ZooKeeper Observers

by

Table of contents

1 Observers: Scaling ZooKeeper Without Hurting Write Performance

Although ZooKeeper performs very well by having clients connect directly to voting members of the ensemble, this architecture makes it hard to scale out to huge numbers of clients. The problem is that as we add more voting members, the write performance drops. This is due to the fact that a write operation requires the agreement of (in general) at least half the nodes in an ensemble and therefore the cost of a vote can increase significantly as more voters are added.

We have introduced a new type of ZooKeeper node called an *Observer* which helps address this problem and further improves ZooKeeper's scalability. Observers are non-voting members of an ensemble which only hear the results of votes, not the agreement protocol that leads up to them. Other than this simple distinction, Observers function exactly the same as Followers - clients may connect to them and send read and write requests to them. Observers forward these requests to the Leader like Followers do, but they then simply wait to hear the result of the vote. Because of this, we can increase the number of Observers as much as we like without harming the performance of votes.

Observers have other advantages. Because they do not vote, they are not a critical part of the ZooKeeper ensemble. Therefore they can fail, or be disconnected from the cluster, without harming the availability of the ZooKeeper service. The benefit to the user is that Observers may connect over less reliable network links than Followers. In fact, Observers may be used to talk to a ZooKeeper server from another data center. Clients of the Observer will see fast reads, as all reads are served locally, and writes result in minimal network traffic as the number of messages required in the absence of the vote protocol is smaller.

2 How to use Observers

Setting up a ZooKeeper ensemble that uses Observers is very simple, and requires just two changes to your config files. Firstly, in the config file of every node that is to be an Observer, you must place this line:

```
peerType=observer
```

This line tells ZooKeeper that the server is to be an Observer. Secondly, in every server config file, you must add :observer to the server definition line of each Observer. For example:

```
server.1:localhost:2181:3181:observer
```

This tells every other server that server.1 is an Observer, and that they should not expect it to vote. This is all the configuration you need to do to add an Observer to your ZooKeeper cluster. Now you can connect to it as though it were an ordinary Follower. Try it out, by running:

```
bin/zkCli.sh -server localhost:2181
```

where localhost:2181 is the hostname and port number of the Observer as specified in every config file. You should see a command line prompt through which you can issue commands like *ls* to query the ZooKeeper service.

3 Example use cases

Two example use cases for Observers are listed below. In fact, wherever you wish to scale the numbe of clients of your ZooKeeper ensemble, or where you wish to insulate the critical part of an ensemble from the load of dealing with client requests, Observers are a good architectural choice.

- As a datacenter bridge: Forming a ZK ensemble between two datacenters is a problematic endeavour as the high variance in latency between the datacenters could lead to false positive failure detection and partitioning. However if the ensemble runs entirely in one datacenter, and the second datacenter runs only Observers, partitions aren't problematic as the ensemble remains connected. Clients of the Observers may still see and issue proposals.
- As a link to a message bus: Some companies have expressed an interest in using ZK as a component of a persistent reliable message bus. Observers would give a natural integration point for this work: a plug-in mechanism could be used to attach the stream of proposals an Observer sees to a publish-subscribe system, again without loading the core ensemble.