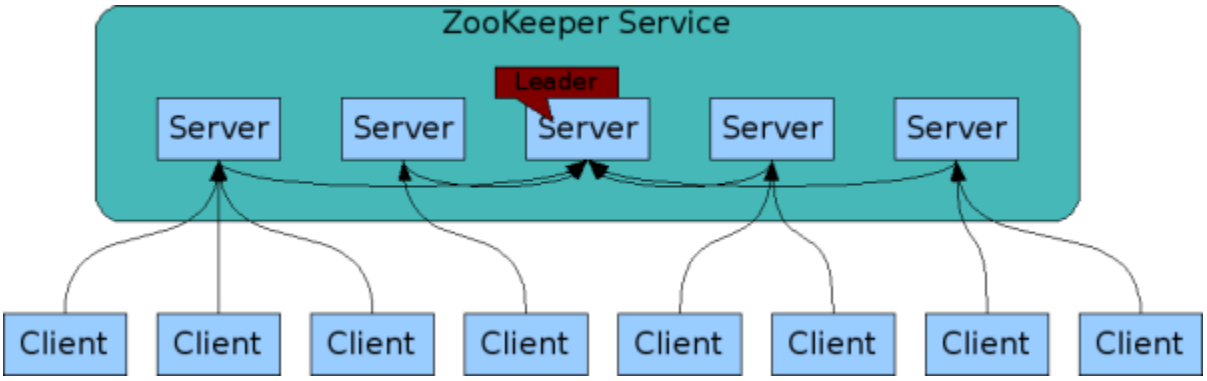
ProjectDescription

ZooKeeper Overview

ZooKeeper allows distributed processes to coordinate with each other through a shared hierarchical name space of data registers (we call these registers znodes), much like a file system. Unlike normal file systems ZooKeeper provides its clients with high throughput, low latency, highly available, strictly ordered access to the znodes. The performance aspects of ZooKeeper allow it to be used in large distributed systems. The reliability aspects prevent it from becoming the single point of failure in big systems. Its strict ordering allows sophisticated synchronization primitives to be implemented at the client.

The name space provided by ZooKeeper is much like that of a standard file system. A name is a sequence of path elements separated by a slash ("/"). Every znode in ZooKeeper's name space is identified by a path. And every znode has a parent whose path is a prefix of the znode with one less element; the exception to this rule is root ("/") which has no parent. Also, exactly like standard file systems, a znode cannot be deleted if it has any children.

The main differences between ZooKeeper and standard file systems are that every znode can have data associated with it (every file can also be a directory and vice-versa) and znodes are limited to the amount of data that they can have. ZooKeeper was designed to store coordination data: status information, configuration, location information, etc. This kind of meta-information is usually measured in kilobytes, if not bytes. ZooKeeper has a built-in sanity check of 1M, to prevent it from being used as a large data store, but in general it is used to store much smaller pieces of data.



The service itself is replicated over a set of machines that comprise the service. These machines maintain an in-memory image of the data tree along with a transaction logs and snapshots in a persistent store. Because the data is kept in-memory, ZooKeeper is able to get very high throughput and low latency numbers. The downside to an in-memory database is that the size of the database that ZooKeeper can manage is limited by memory. This limitation is further reason to keep the amount of data stored in znodes small.

The servers that make up the ZooKeeper service must all know about each other. As long as a majority of the servers are available, the ZooKeeper service will be available. Clients must also know the list of servers. The clients create a handle to the ZooKeeper service using this list of servers.

Clients only connect to a single ZooKeeper server. The client maintains a TCP connection through which it sends requests, gets responses, gets watch events, and sends heartbeats. If the TCP connection to the server breaks, the client will connect to a different server. When a client first connects to the ZooKeeper service, the first ZooKeeper server will setup a session for the client. If the client needs to connect to another server, this session will get reestablished with the new server.

Read requests sent by a ZooKeeper client are processed locally at the ZooKeeper server to which the client is connected. If the read request registers a watch on a znode, that watch is also tracked locally at the ZooKeeper server. Write requests are forwarded to other ZooKeeper servers and go through consensus before a response is generated. Sync requests are also forwarded to another server, but do not actually go through consensus. Thus, the throughput of read requests scales with the number of servers and the throughput of write requests decreases with the number of servers.

Order is very important to ZooKeeper; almost bordering on obsessive–compulsive disorder. All updates are totally ordered. ZooKeeper actually stamps each update with a number that reflects this order. We call this number the zxid (ZooKeeper Transaction Id). Each update will have a unique zxid. Reads (and watches) are ordered with respect to updates. Read responses will be stamped with the last zxid processed by the server that services the read.