ZooKeeper: highly reliable distributed coordination

Weborama Tech Day 2015

September 2015

Elevator pitch

- Server and client library (C, Java) for enabling communication in distributed applications.
- Not a high level tool, more of a framework for writing your own algorithms.
- **Not** a message queue (though you can make one with it)

History

- Originally a component of Hadoop (2005?)
- Split off as a subproject (2008?)when it became apparent that it would be useful on its own
- Currently an Apache "top-level" project with its own community

Users

HBase	Distributed column-oriented data store based on HDFS
Juju	Canonical's service deployment and orchestration framework
Kafka	Distributed pub/sub MQ
Mesos	Cluster management platform for distributed applications (Hadoop MR, Spark, ...)
Neo4j HA components	Master/slave component for the graph database
Solr	Enterprise search engine
...	many others

Users

- eBay
- Rackspace
- TubeMogul
- Yahoo!
- Zynga
- ... many others

How to use it

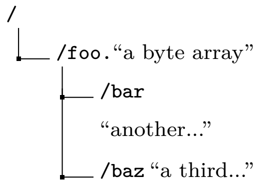
Tree of nodes

Very basic usage: create nodes, write to them, read from them, delete them.

Example

```
ZooKeeper zk =
  new ZooKeeper("localhost:2181", ...);
zk.create("/foo", "a byte array", ...);
zk.create("/foo/bar",
  "another byte array", ...);
zk.create("/foo/baz",
  "a third byte array", ...);
```

On the server:



... so how's this better than NFS anyway?

Adding coordination: nodes

When creating nodes, you can optionally make them

ephemeral	Ephemeral nodes live only as long as the client does.
sequential	Sequential nodes are guaranteed unique (a monotonically increasing integer is appended)
both	
neither	

Adding coordination: watchers

Some operations will allow you to set a watcher callback: when you do a read operation on a thing, you can (*atomically*) notify the server that you're interested in updates on that thing.

Then you do another read operation to check how the thing has changed.

With a watcher set, you can miss some updates, but you are always notified that *something* happened.

Adding coordination: watchers

```
1 zk.exists(path, true, watcher);
2 zk.getData(path, true, watcher);
3 zk.getChildren(path, true, watcher);
```

Whenever the node at *path* is created (1, 2), deleted (1, 2), or its contents are modified (1, 2), or a subnode is created (3) or deleted (3), the watcher will be called.

Adding coordination: watchers

Other things that cause watchers to be called:

- Indirectly when another client dies: all the ephemeral nodes they created disappear (after max. `$heartbeat` seconds), so all clients with watchers on these nodes will be notified
- Connection state changes (`CONNECTED`, `CLOSED`, etc.) also trigger watchers set on the connection itself

Examples

Examples

- Distributed configuration
- Naming service
- Leader election, group membership
- Synchronization

Example

```
ZooKeeper zk =
  new ZooKeeper("localhost:2181", ...);
byte[] jsonString =
  zk.getData("/fetcher", ...);
```

```
{ "throttling": {
      "sandtrapDelay": 0,
      "sandtrapMaxSize": 10000,
      "tickPeriod": 0,
      "logPeriod": 60 },
  "urlCache": {
      "expireAfter": 3600 } }
```

`java-Weborama-Configuration` for BigSea document fetching

Examples

- Distributed configuration
- Naming service
- Leader election, group membership
- Synchronization

Example

```
byte[] protobufRecord =
  zk.getData("/hbase/master", ...);

{ "start_code": 1441897359029,
  "host_name": "betelgeuse",
  "port": 60000 }
```

HBase advertising its “master” node

Examples

- Distributed configuration
- Naming service
- Leader election, group membership
- Synchronization

Example

```
List<String> kids =
  zk.getChildren("/replicas", ...);

# the current caliph
/replicas/member-0000000001
# and his four viziers
/replicas/member-0000000002
/replicas/member-0000000003
/replicas/member-0000000004
/replicas/member-0000000005
```

basic group membership/leader election

algorithm

Examples

- Distributed configuration
- Naming service
- Leader election, group membership
- Synchronization

Example

```
List<String> kids =
  zk.getChildren("/shared-locks", ...);

# this guy has the lock
/shared-locks/write-0000000001
# then it'll be these two
/shared-locks/read-0000000002
/shared-locks/read-0000000003
# then him
/shared-locks/write-0000000004
# finally this one
/shared-locks/read-0000000005
```

globally synchronous shared locking

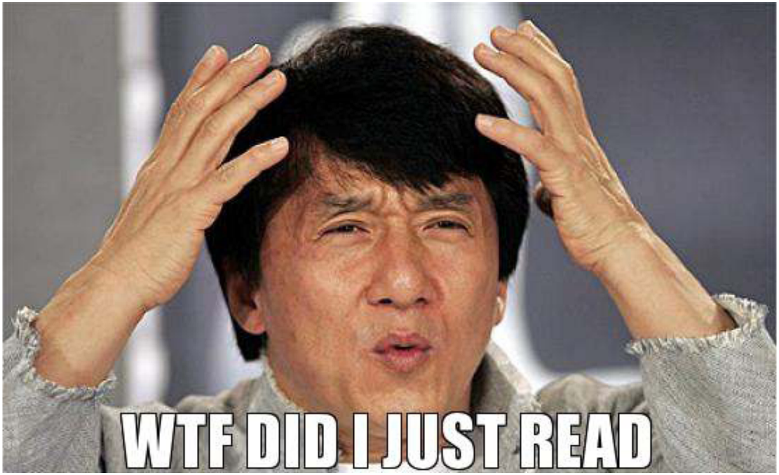
Behind the scenes

Guarantees provided

These are necessary for proper coordination:

- causal ordering
- total ordering of messages
- reliable delivery
- atomic delivery

Guaranteed



in english, please

If a client sees a message, and as a result sends another, everybody will see the answer *after* the question (“causal ordering”). Same even for messages that are not request-reply (“total ordering”).

If any server delivers a message, then eventually every server will deliver it (“reliable delivery”). There are no partially successful operations; either they fail completely, or they succeed completely (“atomic delivery”).

Basic example: Group membership

Group membership algorithm

Assume a well-known node path for the group, e.g. `/replicas`.

1. Announce your membership to the group by creating an ephemeral, sequential subnode of `/replicas`

   ```
   /replicas/member-0000000001, ...
   ```

2. Get the list of members of the group + set a watcher on the group membership list
3. When the watcher callback fires, do previous step again
4. When leaving the group, delete your own node

Features

Dead members
Group desertion (`kill -9 [PID]`) also causes the subnode to disappear after the server notices the client's heartbeat is gone.

New members
Existing members are notified through their watcher, so they know to update the membership list.

Recommended reading

The full version of this presentation with runnable Perl examples is available here: (CC-BY-NC)
`https://bitbucket.org/fgabolde/tech-day-2015-09`

Recommended reading

Consistency, availability, partition tolerance: pick two, one of which is partition tolerance.
URL `https://en.wikipedia.org/wiki/CAP_theorem`.

Series of articles on aphyr.com demonstrating coordination issues in existing distributed software.
URL `https://aphyr.com/tags/Jepsen`.

Paxos consensus protocol.
URL `https://en.wikipedia.org/wiki/Paxos_(computer_science)`.

Zookeeper implementation details.
URL `http://zookeeper.apache.org/doc/r3.3.2/zookeeperInternals.html`.

End credits

Thanks for listening.

Questions?