

Description of how HBase uses ZooKeeper

ZooKeeper recipes that HBase plans to use current and future. By documenting these cases we (zk/hbase) can get a better idea of both how to implement the usecases in ZK, and also ensure that ZK will support these. In some cases it may be prudent to verify the cases (esp when scaling issues are identified). It may also be that new features, etc... might be identified.

Currently, hbase clients find the cluster to connect to by asking zookeeper. The only configuration a client needs is the zk quorum to connect to. Masters and hbase slave nodes (regionservers) all register themselves with zk. If their znode evaporates, the master or regionserver is considered lost and repair begins.

HBase currently will default to manage the zookeeper cluster. It does this in an attempt at not burdening users with yet another technology to figure; things are bad enough for the hbase noob what with hbase, hdfs, and mapreduce. Part of hbase's management of zk includes being able to see zk configuration in the hbase configuration files. Anything that has the hbase.zookeeper prefix will have its suffix mapped to the corresponding zoo.cfg setting (HBase parses its config. and feeds the relevant zk configurations to zk on start).

Below is some more detail on current (hbase 0.20.x) hbase use of zk:

Current Usecases

When I list the /hbase dir in zk I see this.

```
hbase(main):002:0> zk "ls /hbase"
[root-region-server, rs, master, shutdown]
```

root-region-server

This znode holds the location of the server hosting the root of all tables in hbase.

master

This is current master. If more than one master, they fight over who it should be. They all try to grab this znode. When this node evaporates, masters try to grab it again. Whoever wins picks up the master role.

rs

A directory in which there is a znode per hbase server (regionserver) participating in the cluster. They register themselves when they come on line. They name of the znode is a random number, the regions' startcode, so can tell if regionserver has been restarted (We should fix this so server names are more descriptive).

If regionserver session in zk is lost, this znode evaporates. The regionserver will get the disconnect message and shut itself down. Master will start the clean up process gathering its write-ahead logs, splitting them and divvying the edits out per region so they are available when regions are opened in new locations on other running regionserver.

shutdown

If cluster is to shutdown.

Near Future Usecases 0.21 HBase

Case 1

Summary: HBase Table State and Schema Changes

A table has a schema and state (online, read-only, etc.). When we say thousands of RegionServers, we're trying to give a sense of how many watchers we'll have on the znode that holds table schemas and state. When we say hundreds of tables, we're trying to give some sense of how big the znode content will be... say 256 bytes of schema -- we'll only record difference from default to minimize what's up in zk -- and then state I see as being something like zk's four-letter words only they can be compounded in this case. So, 100s of tables X 1024 schema X (2 four-letter words each on average) at the outside makes for about a MB of data that thousands of regionserver are watching. That OK?

Expected scale: Thousands of RegionServers watching ready to react to changes with about 100 tables each of which can have 1 or 2 states and an involved schema

[MS] I was thinking one znode of state and schema. RegionServers would all have a watch on it. 100s of tables means that a schema change on any table would trigger watches on 1000s of RegionServers. That might be OK though because any RegionServer could be carrying a Region from the edited table.

[PDH] My original assumption was that each table has it's own znode (and would still be my advice). In general you don't want to store very much data per znode - the reason being that writes will slow (think of this -- client copies data to ZK server, which copies data to ZK leader, which broadcasts data to all servers in the cluster, which then commit allowing the original server to respond to the client). If the znode is changing infrequently, then no big deal, but in general you don't want to do this. Also, the ZK server has a 1mb max data size by default, so if you increase the number of tables (etc...) you will bump this at some point.


[PDH] Hence my original assumption, and suggestion. Consider having a znode per table, rather than a single znode. It's more scalable and should be better in general. That's up to you though - 1 znode will work too.

[PDH] A single table can change right? Not all the tables necessarily change state at the same time? Then splitting into multiple znodes makes even more sense - you would only be changing (writing) for things that change, even better is that the watchers will know the exact table that changed rather than determining by reading the data and diffing.... I'm no expert on hbase but from a typical ZK use case this is better. Obv this is a bit more complex than a single znode, also there are more (separate) notifications that will fire instead of a single one.... so you'd have to think through your use case (you could have a toplevel "state" znode that brings down all the tables in the case where all the tables need to go down.... then you wouldn't have to change each table individually for this case (all tables down for whatever reason).

General recipe implemented: A better description of problem and sketch of the solution can be found at [Master Rewrite: Table State](#)

[PDH] this is essentially "dynamic configuration" usecase - we are telling each region server the state of the table containing a region it manages, when the master changes the state the watchers are notified

[MS] Is "dynamic configuration" usecase a zk usecase type described somewhere?

[PDH] What we have is  here. Not much to it really - both for name service and dynamic config you are creating znodes that store relevant data, ZK clients can read/write/watch those nodes.

Case 2

Summary: HBase Region Transitions from unassigned to open and from open to unassigned with some intermediate states

Expected scale: 100k regions across thousands of RegionServers

[PDH start]

This sounds like 2 recipes -- "dynamic configuration" ("dynamic sharding", same thing except the data may be a bit larger) and "group membership". Basically you want to have a list of region servers that are available to do work. You also want a master to coordinate the work among the region servers. You also want to ensure that the work handed to the RS is acted upon in order (state transitions) and would like to know the status of the work at any point in time. So really I see two recipes here:

Here's an idea, see if I got the idea right, obv would have to flesh this out more but this is the general idea. I've chosen random paths below, obv you'd want some sort of prefix, better names, etc...

1) group membership:

1. have a /regionservers znode
2. master watches /regionservers for any child changes
3. as each region server becomes available to do work (or track state if up but not avail) it creates an ephemeral node
 - o /regionserver/<host:port> = <status>
4. master watches /regionserver/<host:port> and cleans up if RS goes away or changes status

[MS] Looks good.

2) task assignment (ie dynamic configuration)

1. have a /tables znode
2. /tables/<regionserver by host:port> which gets created when master notices new region server
 - o RS host:port watches this node for any child changes
3. /tables/<regionserver by host:port>/<regionX> znode for each region assigned to RS host:port
 - o RS host:port watches this node in case reassigned by master, or region changes state
4. /tables/<regionserver by host:port>/<regionX>/<state>-<seq#> znode created by master
 - o seq ensures order seen by RS
 - o RS deletes old state znodes as it transitions out, oldest entry is the current state, always 1 or more znode here -- the current state

[MS] ZK will do the increment for us? This looks good too.

[PDH] Right, the "increment" is using the SEQUENTIAL flag on create


Any metadata stored for a region znode (ie to identify)? As long as size is small no problem. (if a bit larger consider a /regions/<regionX> znodes which has a list of all regions and their identity (otw r/o data fine too))

Numbers:

1) 1001 watches by master (1001 znodes)


2) Numbers for this are:

- 1000 watches, one each by RS on /tables (1 znode) -- really this may not be necessary, esp after <self> is created (reduce noise by not setting when not needed)
- 1000 watches, one each by RS on /tables/<self> (1000 znodes)
- 100K watches, 100 for each RS on /tables/<self>/<region[1-100]> (100k znodes total)
- if master wants to monitor region state then we're looking at 100k watches by master

So totally something on the order of 100k watches. No problem. 

[MS] Really? This sounds great Patrick. Let me take a closer look..... Excellent.

[PDH] Obv you need the hw (jvm heap esp, io bandwidth) to support it and the GC needs to be tuned properly to reduce pausing (which cause timeout/expiration) but 100k is not that much.

See  this perf doc for some ideas, 20 clients doing 50k watches each - 1 million watches on a single core standalone server and still << 5ms avg response time (async ops, keep that in mind re implementation time) YMMV of course but your numbers are well below this.

Worst-case scenarios -- say a cascade failure where all RS become disconnected and sessions expire

1. master will get notified of 1000 RS gone
2. master will delete all nodes in 2) - 1000 RS znodes, 100 regions each RS znode, 1 state (typ) each of the 100 reg
 - o 200k nodes deleted (hint: use async) for each RS


Another worst case:

1. some set of RS are flapping (this may actually be much worse than just dying)
 - o consider running some sort of health check on RS before assigning work, in case it just dies
 - o or, slowly ramp up the number of regions assigned to the RS, allow it to prove itself vs dumping a number of regions on it and then have it flap... (donno enough about hbase to comment resonably, but thing about something like this)
2. for each RS master is deleting 200 znodes

[MS] Excellent.

[PDH] think about potential other worst case scenarios, this is key to proper operation of the system. Esp around "herd" effects and trying to minimize those.

[PDH end]

General recipe implemented: None yet. Need help. Was thinking of keeping queues up in zk -- queues per regionserver for it to open/close etc. But the list of all regions is kept elsewhere currently and probably for the foreseeable future out in our .META. catalog table. Some further description can be found here  Master Rewrite: Region State

Far Future Interest

Potential uses going forward

ZooKeeper/HBaseUseCases (last edited 2009-11-20 18:36:03 by 63)