Troubleshooting

Troubleshooting ZooKeeper Operating Environment

This page details specific problems people have seen, solutions (if solved) to those issues and the types of steps taken to troubleshoot the issue. Feel free to update with your experiences.

The Hbase troubleshooting page also has insight for identifying/resolving ZK issues.

Monitoring

It is important to monitor the ZK environment (hardware, network, processes, etc...) in order to more easily troubleshoot problems. Otherwise you miss out on important information for determining the cause of the problem. What type of monitoring are you doing on your cluster? You can monitor at the host level – that will give you some insight on where to look; cpu, memory, disk, network, etc... You can also monitor at the process level – the ZooKeeper server JMX interface will give you information about latencies and such (you can also use the four letter words for that if you want to hack up some scripts instead of using JMX). JMX will also give you insight into the JVM workings - so for example you could confirm/ruleout GC pauses causing the JVM Java threads to hang for long periods of time (see below).

Without monitoring troubleshooting will be more difficult, but not impossible. JMX can be used through jconsole, or access the stats through the four letter words, also the log4j log contains much important/useful information.
You can also use SPM for ZooKeeper to see all ZooKeeper, JVM, and system/host metrics.

Troubleshooting Checklist

The following can be useful checklist when you are having issues with your ZK cluster, in particular if you are seeing large numbers of timeouts, sessions expirations, poor performance, or high operation latencies. Use the following on all servers and potentially on clients as well:

- hdparm with the -t and -T options to test your disk IO
- time dd if=/dev/urandom bs=512000 of=/tmp/memtest count=1050
    - time md5sum /tmp/memtest; time md5sum /tmp/memtest; time md5sum /tmp/memtest
    - See ECC memory section below for more on this
- ethtool to check the configuration of your network
- ifconfig also to check network and examine error counts
    - ZK uses TCP for network connectivity, errors on the NICs can cause poor performance
- scp/ftp/etc... can be used to verify connectivity, try copying large files between nodes
- thesehttp://github.com/phunt/zk-smoketest#readme smoke and latency tests can be useful to verify a cluster

Compare your results to some baselines

See the Latency Overview page for some latency baselines. You can also compare the performance of cpu/disk/mem/etc... that you have available to what is used in this test.

A word or two about heartbeats

Keep in mind the the session timeout period is used by both the client and the server. If the ZK leader doesn't hear from the client w/in the timeout (say it's 5 sec) it will expire the session. The client is sending a ping after 1/3 of the timeout period. It expects to hear a response before another 1/3 of the timeout elapses, after which it will attempt to re-sync to another server in the cluster. In the 5 sec timeout case you are allowing 1.3 seconds for the request to go to the server, the server to respond back to the client, and the client to process the response. Check the latencies in ZK's JMX in order to get insight into this. i.e. if the server latency is high, say because of io issues, or jvm swapping, vm latency, etc... that will cause the client/sessions to timeout.

Frequent client disconnects & session expirations

ZooKeeper is a canary in a coal mine of sorts. Because of the heart-beating performed by the clients and servers ZooKeeper based applications are very sensitive to things like network and system latencies. We often see client disconnects and session expirations associated with these types of problems.

Take a look at this section to start.

Client disconnects due to client side swapping

This link specifically discusses the negative impact of swapping in the context of the server. However swapping can be an issue for clients as well. It will delay, or potentially even stop for a significant period, the heartbeats from client to server, resulting in session expirations.

As told by a user:

"This issue is clearly linked to heavy utilization or swapping on the clients. I find that if I keep the clients from swapping that this error materializes relatively infrequently, and when it does materialize it is linked to a sudden increase in load. For example, the concurrent start of 100 clients on 14 machines will sometimes trigger this issue. <...> All in all, it is my sense that Java processes must avoid swapping if they want to have not just timely but also reliable behavior."

As told by a HBase user:

"After looking ganglia
history, it's clear that the nodes in question were starved of memory,
swapping like crazy. The expired scanner lease, the region shutting down,
and as you noted, the Zookeeper session expiry, were not a causal chain, but
all the result of the machine grinding to a halt from swapping. The
MapReduce tasks were allocated too much memory, and an apparent memory leak
in the job we were running was causing the tasks to eat into the
RegionServer's share of the machine's memory. I've reduced the memory
allocated to tasks in hadoop's "mapred.child.java.opts" to ensure that the
HADOOP_HEAPSIZE + total maximum memory allocated to tasks + the
HBASE_HEAPSIZE is not greater than the memory available on the machine."

Hardware misconfiguration - NIC

In one case there was a cluster of 5k ZK clients attaching to a ZK cluster, ~20% of the clients had mis-configured NICs, this was causing high tcp packet loss (and therefore high network latency), which caused disconnects (timeout exceeded), but only under fairly high network load (which made it hard to track down!). In the end special processes were setup to continuously monitor client server network latency. Any spikes in the latencies observed were then correlated to the ZK logs (timeouts). In the end all of the NICs were reconfigured on these hosts.

Hardware - network switch

Another issue with the same user as the NIC issue - a cluster of 5k ZK clients attaching to a ZK cluster. It turned out that the network switches had bad firmware which caused high packet latencies under heavy load. At certain times of day we would see high numbers of ZK client disconnects. It turned out that these were periods of heavy network activity, exacerbated by the ZK client session expirations (they caused even more network traffic). In the end the operations team spent a number of days testing/loading the network infrastructure until they were able to pin down the issue as being switch related. The switch firmware was upgraded and this issue was eventually resolved.

Hardware - ifconfig is your friend

A recent issue we saw extremely poor performance from a 3 server ZK ensemble (cluster). Average and max latencies on operations as reported by the "stat" command on the servers was very high (multiple seconds). Turns out that one of the servers had a NIC that was dropping large numbers of packets due to framing problems. Switching out that server with another (no nic issue) resolved the issue. Weird thing was that SSH/SCP/PING etc reported no problems.

Moral of the story: use ifconfig to verify the network interface if you are seeing issues on the cluster.

Hardware - hdparm is your friend

Poor disk IO will also result in increased operation latencies. Use hdparm with the -t and -T options to verify the performance of persistent storage.

Hardware - ECC memory problems can be hard to track down

I've seen a particularly nasty problem where bad ECC memory was causing a single server to run an order of magnitude slower than the rest of the servers in the cluster. This caused some particularly nasty/random problems that were nearly impossible to track down (since the machine kept running, just slowly). Ops replaced the ECC memory and all was fine. See the troubleshooting checklist at the top of this page – the dd/md5sum commands listed there can help to sniff this out (hint: compare the results on all of your servers and verify they are at least "close").

Virtual environments

We've seen situations where users run the entire zk cluster on a set of VMWare vms, all on the same host system. Latency on this configuration was >>> 10sec in some cases due to resource issues (in particular io - see the link I provided above, dedicated log devices are critical to low latency operation of the ZK cluster). Obviously no one should be running in this configuration in production - in particular there will be no reliability in cases where the host storage fails!

Virtual environments - "Cloud Computing"

In one scenario involving EC2 ZK was seeing frequent client disconnects. The user had configured a timeout of 5 seconds, which is too low, probably much too low. Why? You are running in virtualized environments on non-dedicated hardware outside your control/inspection. There is typically no way to tell (unless you are running on the 8 core ec2 systems) if the ec2 host you are running on is over/under subscribed (other vms). There is no way to control disk latency either. You could be seeing large latencies due to resource contention on the ec2 host alone. In addition to that I've heard that network latencies in ec2 are high relative to what you would see if you were running on your own dedicated environment. It's hard to tell the latency btw the servers and client->server w/in the ec2 environment you are seeing w/out measuring it.

GC pressure

The Java GC can cause starvation of the Java threads in the VM. This manifests itself as client disconnects and session expirations due to starvation of the heartbeat thread. The GC runs, locking out all Java threads from running.

You can get an idea of what's happening wrt GC in your client/server VMs by using the following options when starting the JVM:

-Xloggc:gc.log -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCApplicationConcurrentTime -XX:+PrintGC -XX:+PrintGCTimeStamps -XX:+PrintGCDetails

gchisto is a useful tool for analyzing GC logs https://gchisto.dev.java.net/

Additionally you can use 'jstat' on a running jvm to gain more insight into realtime GC activity, see: http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jstat.html

This issue can be resolved in a few ways:

First look at using one of the alternative GCs, in particular low latency GC:

e.g. the following JVM option: -XX:ParallelGCThreads=8 -XX:+UseConcMarkSweepGC

Secondly you might try the solution used by HBASE, spawn a non-Java (JNI) thread to manage your ephemeral znodes. This is a pretty advanced option however, try the alternative GC first and see if that helps.

Performance tuning

Some things to keep in mind while tuning zookeeper performance.

- Verify that logging isn't at DEBUG.Check your log4j.properties file and change the line log4j.rootLogger=DEBUG, ROLLINGFILE to log4j.rootLogger=WARN ROLLINGFILE. Logging to disk on every action can greatly effect performance.
- Verify that you are using fast local disk for the journal.
- Test with http://github.com/phunt/zk-smoketest. This should identify real performance along latency issues. It is built against 32 bit python.