ZooKeeper Administrator's Guide

A Guide to Deployment and Administration

by

Table of contents

1. Deployment

This section contains information about deploying Zookeeper and covers these topics:

- System Requirements
- Clustered (Multi-Server) Setup
- Single Server and Developer Setup

The first two sections assume you are interested in installing ZooKeeper in a production environment such as a datacenter. The final section covers situations in which you are setting up ZooKeeper on a limited basis - for evaluation, testing, or development - but not in a production environment.

1.1. System Requirements

1.1.1. Supported Platforms

- GNU/Linux is supported as a development and production platform for both server and client.
- Sun Solaris is supported as a development and production platform for both server and client.
- FreeBSD is supported as a development and production platform for clients only. Java NIO selector support in the FreeBSD JVM is broken.
- Win32 is supported as a *development platform* only for both server and client.
- MacOSX is supported as a *development platform* only for both server and client.

1.1.2. Required Software

ZooKeeper runs in Java, release 1.6 or greater (JDK 6 or greater). It runs as an *ensemble* of ZooKeeper servers. Three ZooKeeper servers is the minimum recommended size for an ensemble, and we also recommend that they run on separate machines. At Yahoo!, ZooKeeper is usually deployed on dedicated RHEL boxes, with dual-core processors, 2GB of RAM, and 80GB IDE hard drives.

1.2. Clustered (Multi-Server) Setup

For reliable ZooKeeper service, you should deploy ZooKeeper in a cluster known as an *ensemble*. As long as a majority of the ensemble are up, the service will be available. Because Zookeeper requires a majority, it is best to use an odd number of machines. For

example, with four machines ZooKeeper can only handle the failure of a single machine; if two machines fail, the remaining two machines do not constitute a majority. However, with five machines ZooKeeper can handle the failure of two machines.

Here are the steps to setting a server that will be part of an ensemble. These steps should be performed on every host in the ensemble:

1. Install the Java JDK. You can use the native packaging system for your system, or download the JDK from:

   http://java.sun.com/javase/downloads/index.jsp

2. Set the Java heap size. This is very important to avoid swapping, which will seriously degrade ZooKeeper performance. To determine the correct value, use load tests, and make sure you are well below the usage limit that would cause you to swap. Be conservative - use a maximum heap size of 3GB for a 4GB machine.

3. Install the ZooKeeper Server Package. It can be downloaded from:

   http://hadoop.apache.org/zookeeper/releases.html

4. Create a configuration file. This file can be called anything. Use the following settings as a starting point:

   ```
   tickTime=2000 dataDir=/var/zookeeper/ clientPort=2181
   initLimit=5 syncLimit=2 server.1=zoo1:2888:3888
   server.2=zoo2:2888:3888 server.3=zoo3:2888:3888
   ```

   You can find the meanings of these and other configuration settings in the section Configuration Parameters. A word though about a few here:

   Every machine that is part of the ZooKeeper ensemble should know about every other machine in the ensemble. You accomplish this with the series of lines of the form **server.id=host:port:port**. The parameters **host** and **port** are straightforward. You attribute the server id to each machine by creating a file named `myid`, one for each server, which resides in that server's data directory, as specified by the configuration file parameter **dataDir**. The myid file consists of a single line containing only the text of that machine's id. So `myid` of server 1 would contain the text "1" and nothing else. The id must be unique within the ensemble and should have a value between 1 and 255.

5. If your configuration file is set up, you can start a ZooKeeper server:

   ```
   $ java -cp zookeeper.jar:lib/log4j-1.2.15.jar:conf \
   org.apache.zookeeper.server.quorum.QuorumPeerMain zoo.cfg
   ```

   QuorumPeerMain starts a ZooKeeper server, JMX management beans are also registered which allows management through a JMX management console. The ZooKeeper JMX

document contains details on managing ZooKeeper with JMX.

See the script *bin/zkServer.sh*, which is included in the release, for an example of starting server instances.

6. Test your deployment by connecting to the hosts:
   - In Java, you can run the following command to execute simple operations:

     ```
     $ java -cp
     zookeeper.jar:src/java/lib/log4j-1.2.15.jar:conf:src/java/lib/jline-
     \ org.apache.zookeeper.ZooKeeperMain -server
     127.0.0.1:2181
     ```

   - In C, you can compile either the single threaded client or the multithreaded client: or n the c subdirectory in the ZooKeeper sources. This compiles the single threaded client:

     ```
     $ make cli_st
     ```

     And this compiles the mulithreaded client:

     ```
     $ make cli_mt
     ```

   Running either program gives you a shell in which to execute simple file-system-like operations. To connect to ZooKeeper with the multithreaded client, for example, you would run:

   ```
   $ cli_mt 127.0.0.1:2181
   ```

1.3. Single Server and Developer Setup

If you want to setup ZooKeeper for development purposes, you will probably want to setup a single server instance of ZooKeeper, and then install either the Java or C client-side libraries and bindings on your development machine.

The steps to setting up a single server instance are the similar to the above, except the configuration file is simpler. You can find the complete instructions in the Installing and Running ZooKeeper in Single Server Mode section of the ZooKeeper Getting Started Guide.

For information on installing the client side libraries, refer to the Bindings section of the ZooKeeper Programmer's Guide.

2. Administration

This section contains information about running and maintaining ZooKeeper and covers

these topics:

2.1. Designing a ZooKeeper Deployment

The reliablity of ZooKeeper rests on two basic assumptions.

1. Only a minority of servers in a deployment will fail. *Failure* in this context means a machine crash, or some error in the network that partitions a server off from the majority.
2. Deployed machines operate correctly. To operate correctly means to execute code correctly, to have clocks that work properly, and to have storage and network components that perform consistently.

The sections below contain considerations for ZooKeeper administrators to maximize the probability for these assumptions to hold true. Some of these are cross-machines considerations, and others are things you should consider for each and every machine in your deployment.

2.1.1. Cross Machine Requirements

For the ZooKeeper service to be active, there must be a majority of non-failing machines that can communicate with each other. To create a deployment that can tolerate the failure of F machines, you should count on deploying 2xF+1 machines. Thus, a deployment that consists of three machines can handle one failure, and a deployment of five machines can handle two failures. Note that a deployment of six machines can only handle two failures since three

machines is not a majority. For this reason, ZooKeeper deployments are usually made up of an odd number of machines.

To achieve the highest probability of tolerating a failure you should try to make machine failures independent. For example, if most of the machines share the same switch, failure of that switch could cause a correlated failure and bring down the service. The same holds true of shared power circuits, cooling systems, etc.

2.1.2. Single Machine Requirements

If ZooKeeper has to contend with other applications for access to resourses like storage media, CPU, network, or memory, its performance will suffer markedly. ZooKeeper has strong durability guarantees, which means it uses storage media to log changes before the operation responsible for the change is allowed to complete. You should be aware of this dependency then, and take great care if you want to ensure that ZooKeeper operations aren't held up by your media. Here are some things you can do to minimize that sort of degradation:

- ZooKeeper's transaction log must be on a dedicated device. (A dedicated partition is not enough.) ZooKeeper writes the log sequentially, without seeking Sharing your log device with other processes can cause seeks and contention, which in turn can cause multi-second delays.
- Do not put ZooKeeper in a situation that can cause a swap. In order for ZooKeeper to function with any sort of timeliness, it simply cannot be allowed to swap. Therefore, make certain that the maximum heap size given to ZooKeeper is not bigger than the amount of real memory available to ZooKeeper. For more on this, see Things to Avoid below.

2.2. Provisioning

2.3. Things to Consider: ZooKeeper Strengths and Limitations

2.4. Administering

2.5. Maintenance

Little long term maintenance is required for a ZooKeeper cluster however you must be aware of the following:

2.5.1. Ongoing Data Directory Cleanup

The ZooKeeper Data Directory contains files which are a persistent copy of the znodes stored by a particular serving ensemble. These are the snapshot and transactional log files. As changes are made to the znodes these changes are appended to a transaction log, occasionally, when a log grows large, a snapshot of the current state of all znodes will be written to the filesystem. This snapshot supercedes all previous logs.

A ZooKeeper server **will not remove old snapshots and log files**, this is the responsibility of the operator. Every serving environment is different and therefore the requirements of managing these files may differ from install to install (backup for example).

The PurgeTxnLog utility implements a simple retention policy that administrators can use. The API docs contains details on calling conventions (arguments, etc...).

In the following example the last count snapshots and their corresponding logs are retained and the others are deleted. The value of <count> should typically be greater than 3 (although not required, this provides 3 backups in the unlikely event a recent log has become corrupted). This can be run as a cron job on the ZooKeeper server machines to clean up the logs daily.

```
 java -cp zookeeper.jar:log4j.jar:conf
org.apache.zookeeper.server.PurgeTxnLog <dataDir> <snapDir> -n <count>
```

2.5.2. Debug Log Cleanup (log4j)

See the section on logging in this document. It is expected that you will setup a rolling file appender using the in-built log4j feature. The sample configuration file in the release tar's conf/log4j.properties provides an example of this.

2.6. Monitoring

2.7. Logging

ZooKeeper uses **log4j** version 1.2 as its logging infrastructure. The ZooKeeper default `log4j.properties` file resides in the `conf` directory. Log4j requires that `log4j.properties` either be in the working directory (the directory from which ZooKeeper is run) or be accessible from the classpath.

For more information, see Log4j Default Initialization Procedure of the log4j manual.

2.8. Troubleshooting

2.9. Configuration Parameters

ZooKeeper's behavior is governed by the ZooKeeper configuration file. This file is designed so that the exact same file can be used by all the servers that make up a ZooKeeper server assuming the disk layouts are the same. If servers use different configuration files, care must be taken to ensure that the list of servers in all of the different configuration files match.

2.9.1. Minimum Configuration

Here are the minimum configuration keywords that must be defined in the configuration file:

clientPort

the port to listen for client connections; that is, the port that clients attempt to connect to.

dataDir

the location where ZooKeeper will store the in-memory database snapshots and, unless specified otherwise, the transaction log of updates to the database.

Note:

Be careful where you put the transaction log. A dedicated transaction log device is key to consistent good performance. Putting the log on a busy device will adversely effect performance.

tickTime

the length of a single tick, which is the basic time unit used by ZooKeeper, as measured in milliseconds. It is used to regulate heartbeats, and timeouts. For example, the minimum session timeout will be two ticks.

2.9.2. Advanced Configuration

The configuration settings in the section are optional. You can use them to further fine tune the behaviour of your ZooKeeper servers. Some can also be set using Java system properties, generally of the form *zookeeper.keyword*. The exact system property, when available, is noted below.

dataLogDir

(No Java system property)

This option will direct the machine to write the transaction log to the **dataLogDir** rather than the **dataDir**. This allows a dedicated log device to be used, and helps avoid competition between logging and snaphots.

Note:

Having a dedicated log device has a large impact on throughput and stable latencies. It is highly recommened to dedicate a log device and set **dataLogDir** to point to a directory on that device, and then make sure to point **dataDir** to a directory *not* residing on that device.

globalOutstandingLimit

(Java system property: **zookeeper.globalOutstandingLimit.**)

Clients can submit requests faster than ZooKeeper can process them, especially if there are a lot of clients. To prevent ZooKeeper from running out of memory due to queued requests, ZooKeeper will throttle clients so that there is no more than globalOutstandingLimit outstanding requests in the system. The default limit is 1,000.

preAllocSize

(Java system property: **zookeeper.preAllocSize**)

To avoid seeks ZooKeeper allocates space in the transaction log file in blocks of preAllocSize kilobytes. The default block size is 64M. One reason for changing the size of the blocks is to reduce the block size if snapshots are taken more often. (Also, see **snapCount**).

snapCount

(Java system property: **zookeeper.snapCount**)

Clients can submit requests faster than ZooKeeper can process them, especially if there are a lot of clients. To prevent ZooKeeper from running out of memory due to queued requests, ZooKeeper will throttle clients so that there is no more than globalOutstandingLimit outstanding requests in the system. The default limit is 1,000.ZooKeeper logs transactions to a transaction log. After snapCount transactions are written to a log file a snapshot is started and a new transaction log file is started. The default snapCount is 10,000.

traceFile

(Java system property: **requestTraceFile**)

If this option is defined, requests will be will logged to a trace file named traceFile.year.month.day. Use of this option provides useful debugging information, but will impact performance. (Note: The system property has no zookeeper prefix, and the configuration variable name is different from the system property. Yes - it's not consistent, and it's annoying.)

2.9.3. Cluster Options

The options in this section are designed for use with an ensemble of servers -- that is, when deploying clusters of servers.

electionAlg

(No Java system property)

Election implementation to use. A value of "0" corresponds to the original UDP-based version, "1" corresponds to the non-authenticated UDP-based version of fast leader election, "2" corresponds to the authenticated UDP-based version of fast leader election, and "3" corresponds to TCP-based version of fast leader election. Currently, only 0 and 3 are supported, 3 being the default

initLimit

(No Java system property)

Amount of time, in ticks (see tickTime), to allow followers to connect and sync to a leader. Increased this value as needed, if the amount of data managed by ZooKeeper is large.

leaderServes

(Java system property: zookeeper.**leaderServes**)

Leader accepts client connections. Default value is "yes". The leader machine coordinates updates. For higher update throughput at thes slight expense of read throughput the leader can be configured to not accept clients and focus on coordination. The default to this option is yes, which means that a leader will accept client connections.

Note:

Turning on leader selection is highly recommended when you have more than three ZooKeeper servers in an ensemble.

server.x=[hostname]:nnnnn[:nnnnn], etc

(No Java system property)

servers making up the ZooKeeper ensemble. When the server starts up, it determines which server it is by looking for the file `myid` in the data directory. That file contains the server number, in ASCII, and it should match **x** in **server.x** in the left hand side of this setting.

The list of servers that make up ZooKeeper servers that is used by the clients must match the list of ZooKeeper servers that each ZooKeeper server has.

There are two port numbers **nnnnn**. The first followers use to connect to the leader, and the second is for leader election. The leader election port is only necessary if electionAlg

is 1, 2, or 3 (default). If electionAlg is 0, then the second port is not necessary. If you want to test multiple servers on a single machine, then different ports can be used for each server.

syncLimit

(No Java system property)

Amount of time, in ticks (see tickTime), to allow followers to sync with ZooKeeper. If followers fall too far behind a leader, they will be dropped.

group.x=nnnnn[:nnnnn]

(No Java system property)

Enables a hierarchical quorum construction."x" is a group identifier and the numbers following the "=" sign correspond to server identifiers. The left-hand side of the assignment is a colon-separated list of server identifiers. Note that groups must be disjoint and the union of all groups must be the ZooKeeper ensemble.

weight.x=nnnnn

(No Java system property)

Used along with "group", it assigns a weight to a server when forming quorums. Such a value corresponds to the weight of a server when voting. There are a few parts of ZooKeeper that require voting such as leader election and the atomic broadcast protocol. By default the weight of server is 1. If the configuration defines groups, but not weights, then a value of 1 will be assigned to all servers.

2.9.4. Unsafe Options

The following options can be useful, but be careful when you use them. The risk of each is explained along with the explanation of what the variable does.

forceSync

(Java system property: **zookeeper.forceSync**)

Requires updates to be synced to media of the transaction log before finishing processing the update. If this option is set to no, ZooKeeper will not require updates to be synced to the media.

jute.maxbuffer:

(Java system property: **jute.maxbuffer**)

This option can only be set as a Java system property. There is no zookeeper prefix on it. It specifies the maximum size of the data that can be stored in a znode. The default is

0xfffff, or just under 1M. If this option is changed, the system property must be set on all servers and clients otherwise problems will arise. This is really a sanity check. ZooKeeper is designed to store data on the order of kilobytes in size.

skipACL

(Java system property: **zookeeper.skipACL**)

Skips ACL checks. This results in a boost in throughput, but opens up full access to the data tree to everyone.

2.10. ZooKeeper Commands: The Four Letter Words

ZooKeeper responds to a small set of commands. Each command is composed of four letters. You issue the commands to ZooKeeper via telnet or nc, at the client port.

dump

Lists the outstanding sessions and ephemeral nodes. This only works on the leader.

envi

Print details about serving environment

reqs

List outstanding requests

ruok

Tests if server is running in a non-error state. The server will respond with imok if it is running. Otherwise it will not respond at all.

srst

Reset statistics returned by stat command.

stat

Lists statistics about performance and connected clients.

Here's an example of the **ruok** command:

```
$ echo ruok | nc 127.0.0.1 5111
imok
```

2.11. Data File Management

ZooKeeper stores its data in a data directory and its transaction log in a transaction log directory. By default these two directories are the same. The server can (and should) be configured to store the transaction log files in a separate directory than the data files. Throughput increases and latency decreases when transaction logs reside on a dedicated log

devices.

2.11.1. The Data Directory

This directory has two files in it:

- `myid` - contains a single integer in human readable ASCII text that represents the server id.
- `snapshot.<zxid>` - holds the fuzzy snapshot of a data tree.

Each ZooKeeper server has a unique id. This id is used in two places: the `myid` file and the configuration file. The `myid` file identifies the server that corresponds to the given data directory. The configuration file lists the contact information for each server identified by its server id. When a ZooKeeper server instance starts, it reads its id from the `myid` file and then, using that id, reads from the configuration file, looking up the port on which it should listen.

The `snapshot` files stored in the data directory are fuzzy snapshots in the sense that during the time the ZooKeeper server is taking the snapshot, updates are occurring to the data tree. The suffix of the `snapshot` file names is the *zxid*, the ZooKeeper transaction id, of the last committed transaction at the start of the snapshot. Thus, the snapshot includes a subset of the updates to the data tree that occurred while the snapshot was in process. The snapshot, then, may not correspond to any data tree that actually existed, and for this reason we refer to it as a fuzzy snapshot. Still, ZooKeeper can recover using this snapshot because it takes advantage of the idempotent nature of its updates. By replaying the transaction log against fuzzy snapshots ZooKeeper gets the state of the system at the end of the log.

2.11.2. The Log Directory

The Log Directory contains the ZooKeeper transaction logs. Before any update takes place, ZooKeeper ensures that the transaction that represents the update is written to non-volatile storage. A new log file is started each time a snapshot is begun. The log file's suffix is the first zxid written to that log.

2.11.3. File Management

The format of snapshot and log files does not change between standalone ZooKeeper servers and different configurations of replicated ZooKeeper servers. Therefore, you can pull these files from a running replicated ZooKeeper server to a development machine with a stand-alone ZooKeeper server for trouble shooting.

Using older log and snapshot files, you can look at the previous state of ZooKeeper servers

and even restore that state. The LogFormatter class allows an administrator to look at the transactions in a log.

The ZooKeeper server creates snapshot and log files, but never deletes them. The retention policy of the data and log files is implemented outside of the ZooKeeper server. The server itself only needs the latest complete fuzzy snapshot and the log files from the start of that snapshot. See the maintenance section in this document for more details on setting a retention policy and maintenance of ZooKeeper storage.

2.12. Things to Avoid

Here are some common problems you can avoid by configuring ZooKeeper correctly:

inconsistent lists of servers

The list of ZooKeeper servers used by the clients must match the list of ZooKeeper servers that each ZooKeeper server has. Things work okay if the client list is a subset of the real list, but things will really act strange if clients have a list of ZooKeeper servers that are in different ZooKeeper clusters. Also, the server lists in each Zookeeper server configuration file should be consistent with one another.

incorrect placement of transasction log

The most performance critical part of ZooKeeper is the transaction log. ZooKeeper syncs transactions to media before it returns a response. A dedicated transaction log device is key to consistent good performance. Putting the log on a busy device will adversely effect performance. If you only have one storage device, put trace files on NFS and increase the snapshotCount; it doesn't eliminate the problem, but it should mitigate it.

incorrect Java heap size

You should take special care to set your Java max heap size correctly. In particular, you should not create a situation in which ZooKeeper swaps to disk. The disk is death to ZooKeeper. Everything is ordered, so if processing one request swaps the disk, all other queued requests will probably do the same. the disk. DON'T SWAP.

Be conservative in your estimates: if you have 4G of RAM, do not set the Java max heap size to 6G or even 4G. For example, it is more likely you would use a 3G heap for a 4G machine, as the operating system and the cache also need memory. The best and only recommend practice for estimating the heap size your system needs is to run load tests, and then make sure you are well below the usage limit that would cause the system to swap.

2.13. Best Practices

For best results, take note of the following list of good Zookeeper practices. *[tbd...]*