ZooKeeper Dynamic Reconfiguration

by

Table of contents

1 Overview

Prior to the 3.5.0 release, the membership and all other configuration parameters of Zookeeper were static - loaded during boot and immutable at runtime. Operators resorted to "rolling restarts" - a manually intensive and error-prone method of changing the configuration that has caused data loss and inconsistency in production.

Starting with 3.5.0, “rolling restarts” are no longer needed! ZooKeeper comes with full support for automated configuration changes: the set of Zookeeper servers, their roles (participant / observer), all ports, and even the quorum system can be changed dynamically, without service interruption and while maintaining data consistency. Reconfigurations are performed immediately, just like other operations in ZooKeeper. Multiple changes can be done using a single reconfiguration command. The dynamic reconfiguration functionality does not limit operation concurrency, does not require client operations to be stopped during reconfigurations, has a very simple interface for administrators and no added complexity to other client operations.

New client-side features allow clients to find out about configuration changes and to update the connection string (list of servers and their client ports) stored in their ZooKeeper handle. A probabilistic algorithm is used to rebalance clients across the new configuration servers while keeping the extent of client migrations proportional to the change in ensemble membership.

This document provides the administrator manual for reconfiguration. For a detailed description of the reconfiguration algorithms, performance measurements, and more, please see our paper:

> **Shraer, A., Reed, B., Malkhi, D., Junqueira, F. Dynamic Reconfiguration of Primary/Backup Clusters. In *USENIX Annual Technical Conference (ATC)* (2012), 425-437**
> Links: paper (pdf), slides (pdf), video, hadoop summit slides

Note: Starting with 3.5.3, the dynamic reconfiguration feature is disabled by default, and has to be explicitly turned on via reconfigEnabled configuration option.

2 Changes to Configuration Format

2.1 Specifying the client port

A client port of a server is the port on which the server accepts client connection requests. Starting with 3.5.0 the *clientPort* and *clientPortAddress* configuration parameters should no longer be used. Instead, this information is now part of the server keyword specification, which becomes as follows:

```
server.<positive id> = <address1>:<port1>:<port2>[:role];
[<client port address>:]<client port>
```

The client port specification is to the right of the semicolon. The client port address is optional, and if not specified it defaults to "0.0.0.0". As usual, role is also optional, it can be *participant* or *observer* (*participant* by default).

Examples of legal server statements:

- `server.5 = 125.23.63.23:1234:1235;1236`
- `server.5 = 125.23.63.23:1234:1235:participant;1236`
- `server.5 = 125.23.63.23:1234:1235:observer;1236`
- `server.5 = 125.23.63.23:1234:1235;125.23.63.24:1236`
- `server.5 = 125.23.63.23:1234:1235:participant;125.23.63.23:1236`

2.2 The standaloneEnabled flag

Prior to 3.5.0, one could run ZooKeeper in Standalone mode or in a Distributed mode. These are separate implementation stacks, and switching between them during run time is not possible. By default (for backward compatibility) *standaloneEnabled* is set to *true*. The consequence of using this default is that if started with a single server the ensemble will not be allowed to grow, and if started with more than one server it will not be allowed to shrink to contain fewer than two participants.

Setting the flag to *false* instructs the system to run the Distributed software stack even if there is only a single participant in the ensemble. To achieve this the (static) configuration file should contain:

```
standaloneEnabled=false
```

With this setting it is possible to start a ZooKeeper ensemble containing a single participant and to dynamically grow it by adding more servers. Similarly, it is possible to shrink an ensemble so that just a single participant remains, by removing servers.

Since running the Distributed mode allows more flexibility, we recommend setting the flag to *false*. We expect that the legacy Standalone mode will be deprecated in the future.

2.3 The reconfigEnabled flag

Starting with 3.5.0 and prior to 3.5.3, there is no way to disable dynamic reconfiguration feature. We would like to offer the option of disabling reconfiguration feature because with reconfiguration enabled, we have a security concern that a malicious actor can make arbitrary changes to the configuration of a ZooKeeper ensemble, including adding a compromised server to the ensemble. We prefer to leave to the discretion of the user to decide whether to enable it or not and make sure that the appropriate security measure are in place. So in 3.5.3

the reconfigEnabled configuration option is introduced such that the reconfiguration feature can be completely disabled and any attempts to reconfigure a cluster through reconfig API with or without authentication will fail by default, unless **reconfigEnabled** is set to **true**.

To set the option to true, the configuration file (zoo.cfg) should contain:

```
reconfigEnabled=true
```

2.4 Dynamic configuration file

Starting with 3.5.0 we're distinguishing between dynamic configuration parameters, which can be changed during runtime, and static configuration parameters, which are read from a configuration file when a server boots and don't change during its execution. For now, the following configuration keywords are considered part of the dynamic configuration: *server*, *group* and *weight*.

Dynamic configuration parameters are stored in a separate file on the server (which we call the dynamic configuration file). This file is linked from the static config file using the new *dynamicConfigFile* keyword.

Example

zoo_replicated1.cfgzoo_replicated1.cfg

```
tickTime=2000
dataDir=/zookeeper/data/zookeeper1
initLimit=5
syncLimit=2
dynamicConfigFile=/zookeeper/conf/zoo_replicated1.cfg.dynamic
```

zoo_replicated1.cfg.dynamiczoo_replicated1.cfg.dynamic

```
server.1=125.23.63.23:2780:2783:participant;2791
server.2=125.23.63.24:2781:2784:participant;2792
server.3=125.23.63.25:2782:2785:participant;2793
```

When the ensemble configuration changes, the static configuration parameters remain the same. The dynamic parameters are pushed by ZooKeeper and overwrite the dynamic configuration files on all servers. Thus, the dynamic configuration files on the different servers are usually identical (they can only differ momentarily when a reconfiguration is in progress, or if a new configuration hasn't propagated yet to some of the servers). Once created, the dynamic configuration file should not be manually altered. Changed are only made through the new reconfiguration commands outlined below. Note that changing the config of an offline cluster could result in an inconsistency with respect to configuration information stored in the ZooKeeper log (and the special configuration znode, populated from the log) and is therefore highly discouraged.

Example 2

Users may prefer to initially specify a single configuration file. The following is thus also legal:

zoo_replicated1.cfgzoo_replicated1.cfg

```
tickTime=2000
dataDir=/zookeeper/data/zookeeper1
initLimit=5
syncLimit=2
clientPort=2791  // note that this line is now redundant and therefore not
 recommended
server.1=125.23.63.23:2780:2783:participant;2791
server.2=125.23.63.24:2781:2784:participant;2792
server.3=125.23.63.25:2782:2785:participant;2793
```

The configuration files on each server will be automatically split into dynamic and static files, if they are not already in this format. So the configuration file above will be automatically transformed into the two files in Example 1. Note that the clientPort and clientPortAddress lines (if specified) will be automatically removed during this process, if they are redundant (as in the example above). The original static configuration file is backed up (in a .bak file).

2.5 Backward compatibility

We still support the old configuration format. For example, the following configuration file is acceptable (but not recommended):

zoo_replicated1.cfgzoo_replicated1.cfg

```
tickTime=2000
dataDir=/zookeeper/data/zookeeper1
initLimit=5
syncLimit=2
clientPort=2791
server.1=125.23.63.23:2780:2783:participant
server.2=125.23.63.24:2781:2784:participant
server.3=125.23.63.25:2782:2785:participant
```

During boot, a dynamic configuration file is created and contains the dynamic part of the configuration as explained earlier. In this case, however, the line "clientPort=2791" will remain in the static configuration file of server 1 since it is not redundant -- it was not specified as part of the "server.1=..." using the format explained in the section Changes to Configuration Format. If a reconfiguration is invoked that sets the client port of server 1, we remove "clientPort=2791" from the static configuration file (the dynamic file now contain this information as part of the specification of server 1).

3 Upgrading to 3.5.0

Upgrading a running ZooKeeper ensemble to 3.5.0 should be done only after upgrading your ensemble to the 3.4.6 release. Note that this is only necessary for rolling upgrades (if you're fine with shutting down the system completely, you don't have to go through 3.4.6). If you attempt a rolling upgrade without going through 3.4.6 (for example from 3.4.5), you may get the following error:

```
2013-01-30 11:32:10,663 [myid:2] - INFO [localhost/127.0.0.1:2784:QuorumCnxManager
$Listener@498] - Received connection request /127.0.0.1:60876
2013-01-30 11:32:10,663 [myid:2] - WARN [localhost/127.0.0.1:2784:QuorumCnxManager@349] -
 Invalid server id: -65536
```

During a rolling upgrade, each server is taken down in turn and rebooted with the new 3.5.0 binaries. Before starting the server with 3.5.0 binaries, we highly recommend updating the configuration file so that all server statements "server.x=..." contain client ports (see the section Specifying the client port). As explained earlier you may leave the configuration in a single file, as well as leave the clientPort/clientPortAddress statements (although if you specify client ports in the new format, these statements are now redundant).

4 Dynamic Reconfiguration of the ZooKeeper Ensemble

The ZooKeeper Java and C API were extended with getConfig and reconfig commands that facilitate reconfiguration. Both commands have a synchronous (blocking) variant and an asynchronous one. We demonstrate these commands here using the Java CLI, but note that you can similarly use the C CLI or invoke the commands directly from a program just like any other ZooKeeper command.

4.1 API

There are two sets of APIs for both Java and C client.

Reconfiguration API
Reconfiguration API is used to reconfigure the ZooKeeper cluster. Starting with 3.5.3, reconfiguration Java APIs are moved into ZooKeeperAdmin class from ZooKeeper class, and use of this API requires ACL setup and user authentication (see Security for more information.).

Note: for temporary backward compatibility, the reconfig() APIs will remain in ZooKeeper.java where they were for a few alpha versions of 3.5.x. However, these APIs are deprecated and users should move to the reconfigure() APIs in ZooKeeperAdmin.java.

Get Configuration API

Get configuration APIs are used to retrieve ZooKeeper cluster configuration information stored in /zookeeper/config znode. Use of this API does not require specific setup or authentication, because /zookeeper/config is readable to any users.

4.2 Security

Prior to **3.5.3**, there is no enforced security mechanism over reconfig so any ZooKeeper clients that can connect to ZooKeeper server ensemble will have the ability to change the state of a ZooKeeper cluster via reconfig. It is thus possible for a malicious client to add compromised server to an ensemble, e.g., add a compromised server, or remove legitimate servers. Cases like these could be security vulnerabilities on a case by case basis.

To address this security concern, we introduced access control over reconfig starting from **3.5.3** such that only a specific set of users can use reconfig commands or APIs, and these users need be configured explicitly. In addition, the setup of ZooKeeper cluster must enable authentication so ZooKeeper clients can be authenticated.

We also provides an escape hatch for users who operate and interact with a ZooKeeper ensemble in a secured environment (i.e. behind company firewall). For those users who want to use reconfiguration feature but don't want the overhead of configuring an explicit list of authorized user for reconfig access checks, they can set "skipACL" to "yes" which will skip ACL check and allow any user to reconfigure cluster.

Overall, ZooKeeper provides flexible configuration options for the reconfigure feature that allow a user to choose based on user's security requirement. We leave to the discretion of the user to decide appropriate security measure are in place.

Access Control

The dynamic configuration is stored in a special znode ZooDefs.CONFIG_NODE = /zookeeper/config. This node by default is read only for all users, except super user and users that's explicitly configured for write access.

Clients that need to use reconfig commands or reconfig API should be configured as users that have write access to CONFIG_NODE. By default, only the super user has full control including write access to CONFIG_NODE. Additional users can be granted write access through superuser by setting an ACL that has write permission associated with specified user.

A few examples of how to setup ACLs and use reconfiguration API with authentication can be found in ReconfigExceptionTest.java and TestReconfigServer.cc.

Authentication

Authentication of users is orthogonal to the access control and is delegated to existing authentication mechanism supported by ZooKeeper's pluggable authentication schemes. See ZooKeeper and SASL for more details on this topic.

Disable ACL check

ZooKeeper supports "skipACL" option such that ACL check will be completely skipped, if skipACL is set to "yes". In such cases any unauthenticated users can use reconfig API.

4.3 Retrieving the current dynamic configuration

The dynamic configuration is stored in a special znode ZooDefs.CONFIG_NODE = /zookeeper/config. The new `config` CLI command reads this znode (currently it is simply a wrapper to `get /zookeeper/config`). As with normal reads, to retrieve the latest committed value you should do a `sync` first.

```
[zk: 127.0.0.1:2791(CONNECTED) 3] config
server.1=localhost:2780:2783:participant;localhost:2791
server.2=localhost:2781:2784:participant;localhost:2792
server.3=localhost:2782:2785:participant;localhost:2793
version=400000003
```

Notice the last line of the output. This is the configuration version. The version equals to the zxid of the reconfiguration command which created this configuration. The version of the first established configuration equals to the zxid of the NEWLEADER message sent by the first successfully established leader. When a configuration is written to a dynamic configuration file, the version automatically becomes part of the filename and the static configuration file is updated with the path to the new dynamic configuration file. Configuration files corresponding to earlier versions are retained for backup purposes.

During boot time the version (if it exists) is extracted from the filename. The version should never be altered manually by users or the system administrator. It is used by the system to know which configuration is most up-to-date. Manipulating it manually can result in data loss and inconsistency.

Just like a `get` command, the `config` CLI command accepts the `-w` flag for setting a watch on the znode, and `-s` flag for displaying the Stats of the znode. It additionally accepts a new flag `-c` which outputs only the version and the client connection string corresponding to the current configuration. For example, for the configuration above we would get:

```
[zk: 127.0.0.1:2791(CONNECTED) 17] config -c
400000003 localhost:2791,localhost:2793,localhost:2792
```

Note that when using the API directly, this command is called `getConfig`.

As any read command it returns the configuration known to the follower to which your client is connected, which may be slightly out-of-date. One can use the `sync` command for stronger guarantees. For example using the Java API:

```
zk.sync(ZooDefs.CONFIG_NODE, void_callback, context);
zk.getConfig(watcher, callback, context);
```

Note: in 3.5.0 it doesn't really matter which path is passed to the `sync()` command as all the server's state is brought up to date with the leader (so one could use a different path instead of ZooDefs.CONFIG_NODE). However, this may change in the future.

4.4 Modifying the current dynamic configuration

Modifying the configuration is done through the `reconfig` command. There are two modes of reconfiguration: incremental and non-incremental (bulk). The non-incremental simply specifies the new dynamic configuration of the system. The incremental specifies changes to the current configuration. The `reconfig` command returns the new configuration.

A few examples are in: `ReconfigTest.java`, `ReconfigRecoveryTest.java` and `TestReconfigServer.cc`.

4.4.1 General

Removing servers: Any server can be removed, including the leader (although removing the leader will result in a short unavailability, see Figures 6 and 8 in the paper). The server will not be shut-down automatically. Instead, it becomes a "non-voting follower". This is somewhat similar to an observer in that its votes don't count towards the Quorum of votes necessary to commit operations. However, unlike a non-voting follower, an observer doesn't actually see any operation proposals and does not ACK them. Thus a non-voting follower has a more significant negative effect on system throughput compared to an observer. Non-voting follower mode should only be used as a temporary mode, before shutting the server down, or adding it as a follower or as an observer to the ensemble. We do not shut the server down automatically for two main reasons. The first reason is that we do not want all the clients connected to this server to be immediately disconnected, causing a flood of connection requests to other servers. Instead, it is better if each client decides when to migrate independently. The second reason is that removing a server may sometimes (rarely) be necessary in order to change it from "observer" to "participant" (this is explained in the section Additional comments).

Note that the new configuration should have some minimal number of participants in order to be considered legal. If the proposed change would leave the cluster with less than 2 participants and standalone mode is enabled (standaloneEnabled=true, see the section The standaloneEnabled flag), the reconfig will not be processed (BadArgumentsException). If standalone mode is disabled (standaloneEnabled=false) then its legal to remain with 1 or more participants.

Adding servers: Before a reconfiguration is invoked, the administrator must make sure that a quorum (majority) of participants from the new configuration are already connected and synced with the current leader. To achieve this we need to connect a new joining server to the leader before it is officially part of the ensemble. This is done by starting the joining server

using an initial list of servers which is technically not a legal configuration of the system but (a) contains the joiner, and (b) gives sufficient information to the joiner in order for it to find and connect to the current leader. We list a few different options of doing this safely.

1. Initial configuration of joiners is comprised of servers in the last committed configuration and one or more joiners, where **joiners are listed as observers.** For example, if servers D and E are added at the same time to (A, B, C) and server C is being removed, the initial configuration of D could be (A, B, C, D) or (A, B, C, D, E), where D and E are listed as observers. Similarly, the configuration of E could be (A, B, C, E) or (A, B, C, D, E), where D and E are listed as observers. **Note that listing the joiners as observers will not actually make them observers - it will only prevent them from accidentally forming a quorum with other joiners.** Instead, they will contact the servers in the current configuration and adopt the last committed configuration (A, B, C), where the joiners are absent. Configuration files of joiners are backed up and replaced automatically as this happens. After connecting to the current leader, joiners become non-voting followers until the system is reconfigured and they are added to the ensemble (as participant or observer, as appropriate).
2. Initial configuration of each joiner is comprised of servers in the last committed configuration + **the joiner itself, listed as a participant.** For example, to add a new server D to a configuration consisting of servers (A, B, C), the administrator can start D using an initial configuration file consisting of servers (A, B, C, D). If both D and E are added at the same time to (A, B, C), the initial configuration of D could be (A, B, C, D) and the configuration of E could be (A, B, C, E). Similarly, if D is added and C is removed at the same time, the initial configuration of D could be (A, B, C, D). Never list more than one joiner as participant in the initial configuration (see warning below).
3. Whether listing the joiner as an observer or as participant, it is also fine not to list all the current configuration servers, as long as the current leader is in the list. For example, when adding D we could start D with a configuration file consisting of just (A, D) if A is the current leader. however this is more fragile since if A fails before D officially joins the ensemble, D doesn't know anyone else and therefore the administrator will have to intervene and restart D with another server list.

WarningWarning

Never specify more than one joining server in the same initial configuration as participants. Currently, the joining servers don't know that they are joining an existing ensemble; if multiple joiners are listed as participants they may form an independent quorum creating a split-brain situation such as processing operations independently from your main ensemble. It is OK to list multiple joiners as observers in an initial config.

If the configuration of existing servers changes or they become unavailable before the joiner succeeds to connect and learn obout configuration changes, the joiner may need to be restarted with an updated configuration file in order to be able to connect.

Finally, note that once connected to the leader, a joiner adopts the last committed configuration, in which it is absent (the initial config of the joiner is backed up before being rewritten). If the joiner restarts in this state, it will not be able to boot since it is absent from its configuration file. In order to start it you'll once again have to specify an initial configuration.

Modifying server parameters: One can modify any of the ports of a server, or its role (participant/observer) by adding it to the ensemble with different parameters. This works in both the incremental and the bulk reconfiguration modes. It is not necessary to remove the server and then add it back; just specify the new parameters as if the server is not yet in the system. The server will detect the configuration change and perform the necessary adjustments. See an example in the section Incremental mode and an exception to this rule in the section Additional comments.

It is also possible to change the Quorum System used by the ensemble (for example, change the Majority Quorum System to a Hierarchical Quorum System on the fly). This, however, is only allowed using the bulk (non-incremental) reconfiguration mode. In general, incremental reconfiguration only works with the Majority Quorum System. Bulk reconfiguration works with both Hierarchical and Majority Quorum Systems.

Performance Impact: There is practically no performance impact when removing a follower, since it is not being automatically shut down (the effect of removal is that the server's votes are no longer being counted). When adding a server, there is no leader change and no noticeable performance disruption. For details and graphs please see Figures 6, 7 and 8 in the paper.

The most significant disruption will happen when a leader change is caused, in one of the following cases:

1. Leader is removed from the ensemble.
2. Leader's role is changed from participant to observer.
3. The port used by the leader to send transactions to others (quorum port) is modified.

In these cases we perform a leader hand-off where the old leader nominates a new leader. The resulting unavailability is usually shorter than when a leader crashes since detecting leader failure is unnecessary and electing a new leader can usually be avoided during a hand-off (see Figures 6 and 8 in the paper).

When the client port of a server is modified, it does not drop existing client connections. New connections to the server will have to use the new client port.

Progress guarantees: Up to the invocation of the reconfig operation, a quorum of the old configuration is required to be available and connected for ZooKeeper to be able to make progress. Once reconfig is invoked, a quorum of both the old and of the new configurations must be available. The final transition happens once (a) the new configuration is activated, and (b) all operations scheduled before the new configuration is activated by the leader are committed. Once (a) and (b) happen, only a quorum of the new configuration is required. Note, however, that neither (a) nor (b) are visible to a client. Specifically, when a reconfiguration operation commits, it only means that an activation message was sent out by the leader. It does not necessarily mean that a quorum of the new configuration got this message (which is required in order to activate it) or that (b) has happened. If one wants to make sure that both (a) and (b) has already occurred (for example, in order to know that it is safe to shut down old servers that were removed), one can simply invoke an update (`set-data`, or some other quorum operation, but not a `sync`) and wait for it to commit. An alternative way to achieve this was to introduce another round to the reconfiguration protocol (which, for simplicity and compatibility with Zab, we decided to avoid).

4.4.2 Incremental mode

The incremental mode allows adding and removing servers to the current configuration. Multiple changes are allowed. For example:

```
> reconfig -remove 3 -add server.5=125.23.63.23:1234:1235;1236
```

Both the add and the remove options get a list of comma separated arguments (no spaces):

```
> reconfig -remove 3,4 -add
server.5=localhost:2111:2112;2113,6=localhost:2114:2115:observer;2116
```

The format of the server statement is exactly the same as described in the section Specifying the client port and includes the client port. Notice that here instead of "server.5=" you can just say "5=". In the example above, if server 5 is already in the system, but has different ports or is not an observer, it is updated and once the configuration commits becomes an observer and starts using these new ports. This is an easy way to turn participants into observers and vise versa or change any of their ports, without rebooting the server.

ZooKeeper supports two types of Quorum Systems – the simple Majority system (where the leader commits operations after receiving ACKs from a majority of voters) and a more complex Hierarchical system, where votes of different servers have different weights and servers are divided into voting groups. Currently, incremental reconfiguration is allowed only if the last proposed configuration known to the leader uses a Majority Quorum System (BadArgumentsException is thrown otherwise).

Incremental mode - examples using the Java API:

```
List<String> leavingServers = new ArrayList<String>();
leavingServers.add("1");
```

```
leavingServers.add("2");
byte[] config = zk.reconfig(null, leavingServers, null, -1, new Stat());

List<String> leavingServers = new ArrayList<String>();
List<String> joiningServers = new ArrayList<String>();
leavingServers.add("1");
joiningServers.add("server.4=localhost:1234:1235;1236");
byte[] config = zk.reconfig(joiningServers, leavingServers, null, -1, new Stat());

String configStr = new String(config);
System.out.println(configStr);
```

There is also an asynchronous API, and an API accepting comma separated Strings instead of List<String>. See src/java/main/org/apache/zookeeper/ZooKeeper.java.

4.4.3 Non-incremental mode

The second mode of reconfiguration is non-incremental, whereby a client gives a complete specification of the new dynamic system configuration. The new configuration can either be given in place or read from a file:

`> reconfig -file newconfig.cfg` //newconfig.cfg is a dynamic config file, see Dynamic configuration file

`> reconfig -members server.1=125.23.63.23:2780:2783:participant;2791,server.2=125.23.63.24:27`

The new configuration may use a different Quorum System. For example, you may specify a Hierarchical Quorum System even if the current ensemble uses a Majority Quorum System.

Bulk mode - example using the Java API:

```
ArrayList<String> newMembers = new ArrayList<String>();
newMembers.add("server.1=1111:1234:1235;1236");
newMembers.add("server.2=1112:1237:1238;1239");
newMembers.add("server.3=1114:1240:1241:observer;1242");

byte[] config = zk.reconfig(null, null, newMembers, -1, new Stat());

String configStr = new String(config);
System.out.println(configStr);
```

There is also an asynchronous API, and an API accepting comma separated String containing the new members instead of List<String>. See src/java/main/org/apache/zookeeper/ZooKeeper.java.

4.4.4 Conditional reconfig

Sometimes (especially in non-incremental mode) a new proposed configuration depends on what the client "believes" to be the current configuration, and should be applied only to

that configuration. Specifically, the `reconfig` succeeds only if the last configuration at the leader has the specified version.

```
> reconfig -file <filename> -v <version>
```

In the previously listed Java examples, instead of -1 one could specify a configuration version to condition the reconfiguration.

4.4.5 Error conditions

In addition to normal ZooKeeper error conditions, a reconfiguration may fail for the following reasons:

1. another reconfig is currently in progress (ReconfigInProgress)
2. the proposed change would leave the cluster with less than 2 participants, in case standalone mode is enabled, or, if standalone mode is disabled then its legal to remain with 1 or more participants (BadArgumentsException)
3. no quorum of the new configuration was connected and up-to-date with the leader when the reconfiguration processing began (NewConfigNoQuorum)
4. `-v x` was specified, but the version `y` of the latest configuration is not `x` (BadVersionException)
5. an incremental reconfiguration was requested but the last configuration at the leader uses a Quorum System which is different from the Majority system (BadArgumentsException)
6. syntax error (BadArgumentsException)
7. I/O exception when reading the configuration from a file (BadArgumentsException)

Most of these are illustrated by test-cases in `ReconfigFailureCases.java`.

4.4.6 Additional comments

Liveness: To better understand the difference between incremental and non-incremental reconfiguration, suppose that client C1 adds server D to the system while a different client C2 adds server E. With the non-incremental mode, each client would first invoke `config` to find out the current configuration, and then locally create a new list of servers by adding its own suggested server. The new configuration can then be submitted using the non-incremental `reconfig` command. After both reconfigurations complete, only one of E or D will be added (not both), depending on which client's request arrives second to the leader, overwriting the previous configuration. The other client can repeat the process until its change takes effect. This method guarantees system-wide progress (i.e., for one of the clients), but does not ensure that every client succeeds. To have more control C2 may request to only execute the reconfiguration in case the version of the current configuration hasn't changed, as explained in the section Conditional reconfig. In this way it may avoid blindly overwriting the configuration of C1 if C1's configuration reached the leader first.

With incremental reconfiguration, both changes will take effect as they are simply applied by the leader one after the other to the current configuration, whatever that is (assuming that the second reconfig request reaches the leader after it sends a commit message for the first reconfig request -- currently the leader will refuse to propose a reconfiguration if another one is already pending). Since both clients are guaranteed to make progress, this method guarantees stronger liveness. In practice, multiple concurrent reconfigurations are probably rare. Non-incremental reconfiguration is currently the only way to dynamically change the Quorum System. Incremental configuration is currently only allowed with the Majority Quorum System.

Changing an observer into a follower: Clearly, changing a server that participates in voting into an observer may fail if error (2) occurs, i.e., if fewer than the minimal allowed number of participants would remain. However, converting an observer into a participant may sometimes fail for a more subtle reason: Suppose, for example, that the current configuration is (A, B, C, D), where A is the leader, B and C are followers and D is an observer. In addition, suppose that B has crashed. If a reconfiguration is submitted where D is said to become a follower, it will fail with error (3) since in this configuration, a majority of voters in the new configuration (any 3 voters), must be connected and up-to-date with the leader. An observer cannot acknowledge the history prefix sent during reconfiguration, and therefore it does not count towards these 3 required servers and the reconfiguration will be aborted. In case this happens, a client can achieve the same task by two reconfig commands: first invoke a reconfig to remove D from the configuration and then invoke a second command to add it back as a participant (follower). During the intermediate state D is a non-voting follower and can ACK the state transfer performed during the second reconfig comand.

5 Rebalancing Client Connections

When a ZooKeeper cluster is started, if each client is given the same connection string (list of servers), the client will randomly choose a server in the list to connect to, which makes the expected number of client connections per server the same for each of the servers. We implemented a method that preserves this property when the set of servers changes through reconfiguration. See Sections 4 and 5.1 in the paper.

In order for the method to work, all clients must subscribe to configuration changes (by setting a watch on /zookeeper/config either directly or through the `getConfig` API command). When the watch is triggered, the client should read the new configuration by invoking `sync` and `getConfig` and if the configuration is indeed new invoke the `updateServerList` API command. To avoid mass client migration at the same time, it is better to have each client sleep a random short period of time before invoking `updateServerList`.

A few examples can be found in: `StaticHostProviderTest.java` and `TestReconfig.cc`

Example (this is not a recipe, but a simplified example just to explain the general idea):

```
public void process(WatchedEvent event) {
    synchronized (this) {
        if (event.getType() == EventType.None) {
            connected = (event.getState() == KeeperState.SyncConnected);
            notifyAll();
        } else if (event.getPath()!=null &&  event.getPath().equals(ZooDefs.CONFIG_NODE)) {
            // in prod code never block the event thread!
            zk.sync(ZooDefs.CONFIG_NODE, this, null);
            zk.getConfig(this, this, null);
        }
    }
}
public void processResult(int rc, String path, Object ctx, byte[] data, Stat stat) {
    if (path!=null &&  path.equals(ZooDefs.CONFIG_NODE)) {
        String config[] = ConfigUtils.getClientConfigStr(new String(data)).split(" ");   //
 similar to config -c
        long version = Long.parseLong(config[0], 16);
        if (this.configVersion == null){
             this.configVersion = version;
        } else if (version > this.configVersion) {
            hostList = config[1];
            try {
                // the following command is not blocking but may cause the client to close
 the socket and
                // migrate to a different server. In practice its better to wait a short
 period of time, chosen
                // randomly, so that different clients migrate at different times
                zk.updateServerList(hostList);
            } catch (IOException e) {
                System.err.println("Error updating server list");
                e.printStackTrace();
            }
            this.configVersion = version;
} } }
```