

ZK Failover Controller Design

Todd Lipcon
todd@cloudera.com

April 3, 2012

Contents

1	Overview	2
1.1	Background	2
1.2	ZooKeeper-based approach	2
2	Design	2
2.1	Components	2
2.2	HealthMonitor design	3
2.3	ActiveStandbyElector design	4
2.4	ZKFC Design	4
2.5	Fencing	5
2.6	ZKFC State machine diagram	6
2.7	Supporting manual failover in conjunction with automatic	6
	2.7.1 Configuration	7
	2.7.2 Manual Failover Operation	7
3	Example scenarios	8
3.1	Active NN JVM crash	8
3.2	Active NN JVM freeze (e.g SIGSTOP)	8
3.3	Active NN machine crash	8
3.4	Active NN goes unhealthy	8
3.5	Active ZKFC crashes	8
3.6	ZooKeeper crash	8
4	Details yet to be ironed out	9
4.1	Integrating manual failover	9
5	Future work	9
5.1	Preferential nodes	9
5.2	Self-fencing	9
5.3	Process supervision	9
6	Test plan	10

1 Overview

1.1 Background

HDFS-1623 and related JIRAs added high availability support to the HDFS NameNode, but rely on an administrator to manually trigger the failover process. As per the original design doc, manual failover addresses many of the scenarios in which cluster downtime was previously unavoidable; for example, the administrator may now upgrade or replace hardware or software on one NameNode machine after failing over to the standby, without any loss of availability.

However, HDFS-1623 did not address *automatic* failover. By contrast, automatic failover relies on a failure detector to automatically decide that the active NameNode has become unavailable and trigger a fail-over to the warm standby.

1.2 ZooKeeper-based approach

This document describes the design for a method of automatic failover based on Apache ZooKeeper. ZooKeeper (henceforth “ZK”) is itself highly available and provides the following features upon which we can build:

- A strongly consistent repository for small amounts of arbitrary information (*znodes*)
- The ability to create a znode which is automatically deleted when the creator’s client fails (an *ephemeral* node).
- The ability to monitor and be asynchronously notified when the state of a znode changes. (*watchers*)

Using the above features, this design uses ZooKeeper in several key ways:

- **Failure detector** - the active NameNode creates an ephemeral node in ZK. If the active should fail, the ephemeral node will be automatically deleted after a configurable timeout.
- **Active node locator** - by writing a small amount of information into ZK, clients or other services can locate the current active node authoritatively
- **Mutual exclusion of active state** - since ZK is consistent and highly available, we can use it to ensure that at most one node is active at any time.

Other designs for automatic failover could substitute different components for some or all of the above. For example, the Linux-HA project can provide some of the same building blocks. Nevertheless, ZK is particularly attractive because it satisfies all aspects in a single system, with no external configuration. Many Hadoop deployments also already deploy ZooKeeper for other applications such as Apache HBase.

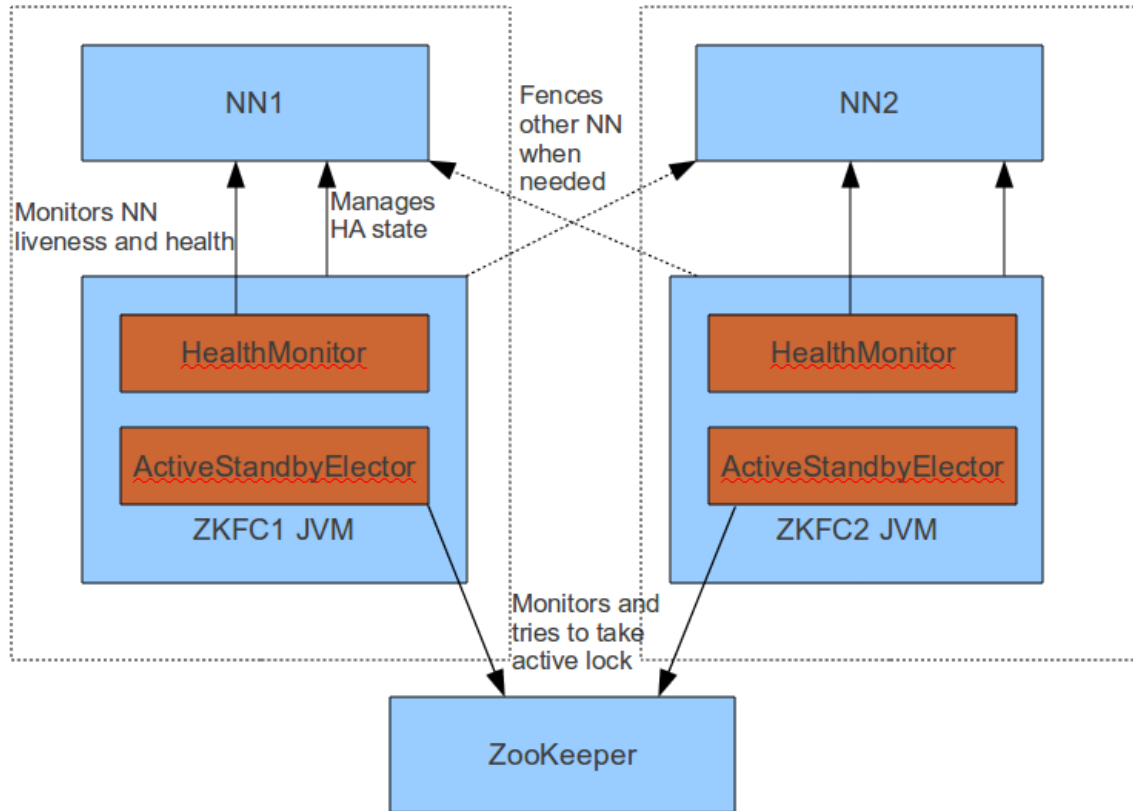
2 Design

2.1 Components

The design for ZK-based automatic failover consists of three main components:

1. A `HealthMonitor` implementation, which watches a `NameNode` to see if the process has become unavailable or entered an unhealthy state
2. A `ActiveStandbyElector` implementation, which manages and monitors state in `ZooKeeper`
3. A `ZKFailoverController` implementation, which subscribes to events from `HealthMonitor` and `ActiveStandbyElector` and manages the state of the `NameNode`. The ZKFC also takes care of fencing the prior `NameNode` if it enters an indeterminate state.

In the current design, all three of the above components run in the same JVM. This JVM runs on the same host as the `NameNode`, but is distinct from the `NameNode` itself. We'll call the JVM as a whole the *ZKFC process*. Thus, in a typical HA cluster with two `NameNodes`, each `NameNode` machine runs its own ZKFC process.



2.2 HealthMonitor design

The `HealthMonitor` (committed in HADOOP-7788) is a thread which is responsible for monitoring the local `NameNode`. It operates in a simple loop, calling the `monitorHealth` RPC. The `HealthMonitor` maintains a view of the current state of the `NameNode` based on the responses to these RPCs. When it transitions between states, it sends a message via a callback interface to the ZKFC. It has the following states:

- **INITIALIZING** - The `HealthMonitor` is starting up and has not yet contacted the NN

- **SERVICE_NOT_RESPONDING** - The health check RPCs are timing out or otherwise not returning either a definitive success or failure
- **SERVICE_HEALTHY** - the health check RPCs are returning success
- **SERVICE_UNHEALTHY** - the health check RPCs are returning a definitive failure (eg the NameNode has itself detected some health issue like being out of local disk space)
- **HEALTH_MONITOR_FAILED** - the health monitor thread has crashed due to an uncaught exception, etc. This is generally a fatal error causing an abort.

2.3 ActiveStandbyElector design

The ActiveStandbyElector (committed in HADOOP-7992 and improved in HADOOP-8163, HADOOP-8212) is responsible for coordinating with ZooKeeper. The ZKFC interacts with it mainly by two main calls:

- **joinElection(...)** - indicate to the ASE that the local NameNode is a candidate to become active.
- **quitElection(...)** - indicate to the ASE that the local NameNode is no longer a candidate to become active (e.g because its health has gone bad)

After the ZKFC calls **joinElection**, the ASE will attempt to acquire a lock in a configurable location in ZooKeeper. This lock consists of an ephemeral znode, such that it will automatically be deleted should the ZKFC process crash, or the node lose its network connection. If the ASE successfully creates the lock, then it will call **becomeActive()** on the ZKFC. Otherwise, it calls **becomeStandby()** and begins to monitor the other node's lock.

In the case that the current lock holder fails, another ZKFC's watch on that node will trigger, causing it to attempt to grab the lock. If it succeeds, the ASE will indicate to the ZKFC that it should now become active with the same **becomeActive()** call.

In the event that the ZooKeeper session expires, the ASE will call **enterNeutralMode** on the local node. It does not, however, call **becomeStandby**, since it has no way of knowing whether another node is ready to take over. The transition of the local node to Standby in this case is handled by the fencing mechanisms (see below).

2.4 ZKFC Design

The ZKFC is itself relatively simple. It runs the following process:

- On startup, initialize the HealthMonitor to monitor the local NameNode. Initialize the ActiveStandbyElector with the configured ZooKeeper quorum information. Do *not* immediately try to join the election.
- When the HealthMonitor's state changes, respond as follows:
 - **SERVICE_HEALTHY** - instruct the elector to join the election if not yet joined
 - **HEALTH_MONITOR_FAILED** - abort the whole ZKFC process, since it can no longer function as designed
 - **INITIALIZING** - in this case, the local node has just restarted, and not ready to service calls. Quits the election, and indicates that fencing is unnecessary, since the NameNode always starts in Standby state.

- Other states - quit the election if currently in the election
- When the ActiveStandbyElector instructs a change, respond as follows:
 - `becomeActive()` - call `transitionToActive()` on the local node. If it fails, quit the election, sleep for a small amount of time, and then rejoin the election. The sleep before rejoining allows other nodes to take a chance at becoming active, if they are able to. When it quits the election in this case, it does *not* delete the breadcrumb node – this ensure that whoever does eventually become active will fence this node, given that it may be left in a partially-active state after the failure.
 - `becomeStandby()` - call `transitionToStandby()` on the local node. If it fails, then the other node will be fencing this node anyway. (see below)
 - `enterNeutralMode()` - currently no response, as there is no such state in the current HA design
 - `fenceOldActive(...)` - see below
 - `notifyFatalError(...)` - abort the ZKFC process, since it is no longer functioning as expected

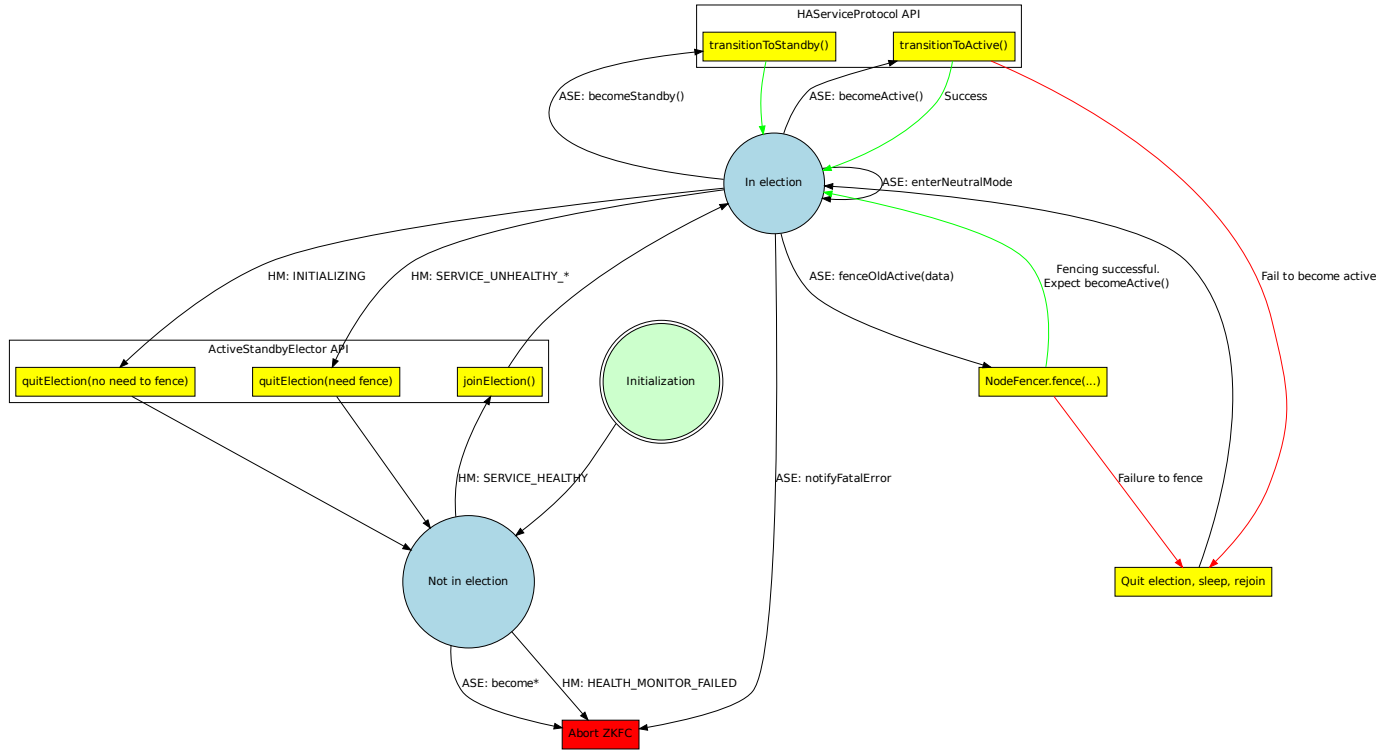
All of the calls are synchronized on the ZKFC object, such that there is a clearly serialized order of events impacting its logic.

2.5 Fencing

HADOOP-8163 enhanced the ActiveStandbyElector design to provide hooks for fencing. The enhancement was as follows:

1. After obtaining the active lock, but before instructing the local process to become active, check for the presence of a *breadcrumb* znode.
 - (a) If it exists, call `fenceOldActive(data)` passing the data from that node. If fencing is successful, delete the breadcrumb node.
 - (b) If fencing fails, log an error, drop the lock, sleep, and rejoin the election. This gives other nodes a chance to try to fence.
 - (c) Create a new breadcrumb node with the local node's identifying data
2. When quitting the election, allow the quitting node to specify whether it may need fencing. If it does not need fencing, delete the breadcrumb node before closing the ZooKeeper session.

2.6 ZKFC State machine diagram



2.7 Supporting manual failover in conjunction with automatic

Even though a cluster may be configured for automatic failover, it is still useful to allow an administrator to trigger a graceful failover. For example, the administrator may want to perform a hardware replacement on one machine, or may want to initiate fail-back to a designated “primary” machine after a failover has taken place.

In the initial implementation, the only means by which a failover can be triggered is a failure. So, the administrator stops the active NN, and allows the automatic behavior to trigger the failover, at which point he or she may restart the prior active. This has the following disadvantages:

- There is no built-in preflight check to ensure that the standby is ready to become active. If the standby is not running, or is not healthy, then the administrator will cause downtime by shutting down the active.
- If an error occurs with the standby as it is becoming active, there is no way to quickly fail back, since the prior active has been shut down.
- Since the prior active fails “un-gracefully”, it will cause fencing to be triggered. This may end up

STONITHing the machine or fencing it from a filer, which is quite inconvenient – especially if the admin just wanted to make a simple configuration change before restarting.

In order to support manual failover in an HA setup, we propose the following changes:

2.7.1 Configuration

Add a new configuration, per namespace, called `dfs.ha.auto-failover.enabled`. If this flag is set to true, the following changes in behavior are enacted:

1. The `haadmin -failover` command line tool no longer makes RPCs directly to NameNodes. Instead, a different implementation (described below) is substituted.
2. The NameNodes enter a mode such that they only accept mutative `HAServiceProtocol` RPCs from ZKFCs.
3. The `start-dfs.sh` script may be augmented to automatically start ZKFC daemons on each node that runs an NN.

The above ensures that a misconfigured client cannot accidentally change the HA state “underneath” an automatic-failover setup.

2.7.2 Manual Failover Operation

When the administrator wishes to initiate a failover, or he or she uses a command line tool: `haadmin -failover -to <namenode>`. We will change the syntax of the non-automatic setup to have the syntax `haadmin -failover -from <namenode> -to <namenode>` for consistency. This tool performs the following operations:

1. HAAdmin makes an RPC `failoverToYou()` to the target ZKFC. The ZKFC performs the following operations:
 - (a) Check that it is in a healthy state (i.e. in the election). If not, throw an exception indicating the bad health.
 - (b) Check whether it is already active. If so, return a success code
 - (c) Send an RPC `concedeLock()` to the currently active ZKFC. That ZKFC performs the following:
 - i. Send a `transitionToStandby()` RPC to its local node.
 - ii. If it succeeds within a short timeout, delete its breadcrumb node. Otherwise, leave the breadcrumb node in tact.
 - iii. Quit the election, and set a timer such that it will not re-join the election for the next 5-10 seconds.
 - (d) When this RPC returns, we expect that the other node has dropped the lock. Wait 5-10 seconds, and then verify that this node has succeeded in taking the lock. Wait for the local node to either become active or to fail to do so. Respond to the client based on the success or failure of the local node’s transition to active.

Note that the initial implementation supposes that there are exactly two nodes taking part in the election. In the future, when we support more than one standby, we can add another znode indicating the target node of the failover, and modify leader election to check this node before trying to take the lock.

3 Example scenarios

3.1 Active NN JVM crash

When the JVM crashes, the HealthMonitor on the same node will fail its `monitorHealth()` call, due to a timeout or a connection-refused error. The HM then triggers `enterState(SERVICE_NOT_RESPONDING)` to the ZKFC. The ZKFC quits the election. The ZKFC on the other node successfully obtains the active lock, initiates fencing, and becomes active.

3.2 Active NN JVM freeze (e.g SIGSTOP)

If the ANN's JVM freezes, but does not crash, the scenario is the same as above. The `monitorHealth()` call will hit an RPC timeout and trigger failover.

Future work: using JVMTI we may be able to determine externally whether the NN JVM is currently experiencing a lengthy garbage collection pause. We could then use a different timeout for failover due to GC pauses.

3.3 Active NN machine crash

When the whole machine crashes, the ActiveStandbyElector will lose its lease in ZooKeeper after the configured session timeout. The other node's ZKFC will notice the deletion of the lock, and proceed the same as above to trigger failover.

3.4 Active NN goes unhealthy

When the health state changes, the HealthMonitor will see a `HealthCheckFailedException` and trigger failover the same as above.

3.5 Active ZKFC crashes

The ZKFC runs in a separate process which is designed to be very simple. However, it is still possible for it to crash (e.g due to a JVM bug, bad memory, etc). In this case, failover will be falsely triggered. Before using aggressive fencing, the failover controller on the other node will issue a `transitionToStandby` to the original node to gracefully give up its active state. This will succeed, since the NameNode itself is healthy. Thus, an unnecessary failover is triggered, but the system proceeds as expected.

3.6 ZooKeeper crash

If ZooKeeper itself crashes, then both ZKFCs will receive a `DISCONNECTED` event at the same time. Upon this event, they call `enterNeutralMode` on the local NameNode, but neither makes any change in state. Thus, the system keeps running while ZooKeeper is down – it is just unable to perform a failover.

When ZooKeeper returns, the clients will immediately reconnect. ZooKeeper's behavior is such that client sessions that were present before the crash are allowed to re-acquire their session so long as they reconnect within their session timeout, starting at the time of restart¹. So, both nodes will regain their session correctly and no unnecessary failovers will be triggered.

¹See <https://cwiki.apache.org/confluence/display/ZOOKEEPER/FAQ>, confirmed by phunt

Future work: the breadcrumb znode could be used in this circumstance to give preference to the node that was active before the ZK outage.

4 Details yet to be ironed out

4.1 Integrating manual failover

Even with automatic failover in place, an administrator may want to trigger manual failover for some other reason (e.g. a hardware upgrade on the active node). Currently, the design does not take administrative input. As a workaround in the initial implementation, the administrator can simply shut down the active NameNode using the usual service management infrastructure. This will initiate a failover within a small number of seconds.

This can be improved by adding a simple RPC interface to the ZKFC which provides an `quiesceActiveState()` call. This call would instruct the NN to leave the election and wait several seconds for the standby node to trigger the failover. If no failover occurs within several seconds, it can re-obtain the active lock and report an error to the administrator.

5 Future work

5.1 Preferential nodes

In some deployments, the administrator may prefer to mark one of their multiple NameNodes as the preferred active. The current design does not allow for that: when the failover controllers start up, they race to take the active lock, and whoever wins the race becomes active. We may want to provide the ability to delay joining the election on the non-preferred node, or even to automatically trigger failback from the non-preferred to the preferred node, even when the non-preferred node remains healthy.

5.2 Self-fencing

When the HM indicates that the local node has gone unhealthy, the ZKFC could initiate a self-fencing step before quitting the election. For example, it can use `fuser -k -9 <ipcport>` to forcefully kill the local NameNode. With this in place, it can avoid many scenarios in which more expensive/complicated fencing mechanisms would otherwise be tried.

5.3 Process supervision

The current design supposes that the ZKFC process and the NameNode process are run independently by service scripts or a cluster management framework. In the event that the NameNode crashes, the ZKFC makes no attempt to re-start it. Instead, it simply continues to monitor the IPC port until the NameNode is restarted by some other means. We assume that existing cluster monitoring is in place to notify the cluster operator, who will take action to restart the NN.

Instead, one could imagine the ZKFC process actually starting (and restarting) the NN. This could make deployment simpler, though it has the downside of adding some complexity to the ZKFC itself, since process management in Java can be messy. Nevertheless, the current design is modular enough that adding this facility would be a straightforward extension of the current work.

6 Test plan

TBD

7 Revision history

commit fecd68fffeef27c7ba44df643da1bc7dbcea0c489
Author: Todd Lipcon <todd@cloudera.com>
Date: Tue Apr 3 22:09:39 2012 -0700

Add section about manual failover in conjunction with automatic

commit cc850453b664109dd55e49e2373991626861493a
Author: Todd Lipcon <todd@cloudera.com>
Date: Sun Apr 1 21:18:53 2012 -0700

Add failure cases to state machine, include more prose about them

commit 1b2f118737b8ac59404ebe14a36d53e2c40a8003
Author: Todd Lipcon <todd@cloudera.com>
Date: Mon Mar 26 18:13:55 2012 -0700

- Add state machine diagram
- Add revision history section

commit 4d207b86f0ee3bfd5d4bc8d5ae377da0ed5d3997
Author: Todd Lipcon <todd@cloudera.com>
Date: Mon Mar 26 14:18:46 2012 -0700

- Fix ZK crash scenario (clients can re-obtain sessions)
- Add INITIALIZING state per atm's feedback

commit 2c7752b75e267b7c1bb1a2920d3bc2c15d9530b5
Author: Todd Lipcon <todd@cloudera.com>
Date: Mon Mar 26 14:16:49 2012 -0700

rev 1