High Availability for JobTracker

Devarajulu K

1.0 Introduction

1.0.1 Purpose

In Hadoop cluster, JobTracker is responsible for managing the life cycle of MapReduce jobs. If JobTracker fails, then MapReduce service will not be available until JobTracker is restarted. We propose an automatic failover solution for JobTracker to address such single point of failure. It is based on Leader Election Framework suggested in ZOOKEEPER-1080

1.0.2 Glossary

- **JobClient** – It is an application side interface to JobTracker. In the trunk, the JobClient API is deprecated and Cluster API is added. But still this document refers to the interface for application to JobTracker as JobClient because of the ease of explanation in this document.
- **Active JobTracker** – A JobTracker instance which serves all the requests from JobClient as well as TaskTracker.
- **Standby JobTracker** – A JobTracker instance which does not serve any request from JobClients or TaskTrackers. When Active JobTracker fails, it takes over Active JobTracker role and starts serving all JobClients and TaskTrackers requests.
- **LES –** Leader Election Service. It is a service responsible for electing a leader among multiple master processes, in this case JobTrackers.

1.0.3 JobTracker Unavailability Scenarios

JobTracker can be unavailable due to,

- Hadoop Upgradation
- Hardware failure of Machine
- Crash due to a bug in its code.

1.0.4 Out of Scope

- **Hot Standby Support**

  This solution does not support the state synchronization between Active and Standby JobTracker.

- **Continuing running jobs in case of JobTracker switch**

  This solution does not handle continuation of the execution of Jobs in case the Active JobTracker fails and Standby JobTracker becomes active.

2.0 Design

2.0.1 Requirements

- ✓ Provide automatic failover mechanism for JobTracker
- ✓ Only one Active JobTracker in the cluster at any point of time.
- ✓ Only Active JobTracker serves requests from JobClients and TaskTrackers.
- ✓ Provide transparent redirection mechanism for JobClient and TaskTracker to newly active JobTracker

2.0.2 Leader Election Service (ZOOKEEPER-1080)

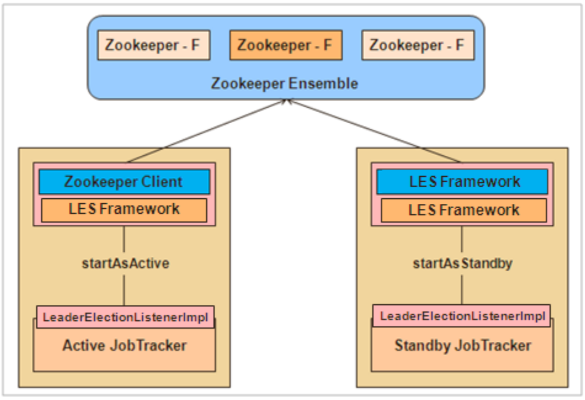


Fig. 1 - JobTracker HA

JobTracker implements LeaderElectionListener interface provided by LES to provide high availability. In rest of the document this LeaderElectionListener interface implementation class is referred to as LeaderElectionListenerImpl. LES triggers following transition methods from LeaderElectionListenerImpl based on which JobTracker is started in Active or Standby mode.

- startInStandby
- startInActive
- standByToActive
- activeToNeutral
- neutralToActive
- standByToNeutral
- neutralToStandBy

2.0.3 JobTracker High Availability

Based on the appropriate transition triggered by LES, JobTracker can be in any of following states.

- ACTIVE
- STANDBY
- NEUTRAL

Following state diagrams explains about the transition of JobTracker into above states based on LES transition events.

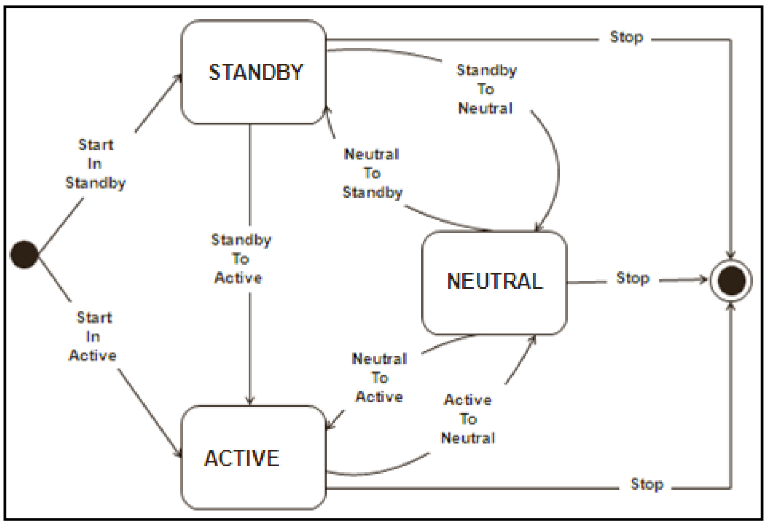


Fig. 2 - JobTracker State Diagram

2.0.3.1 ACTIVE State

JobTracker can go to ACTIVE state based on following scenarios.

- **When multiple JobTrackers are started together**

  LES will only choose one of all started JobTrackers as leader. It will notify that JobTracker’s registered LeaderElectionListenerImpl instance by calling startInActive method on it. LeaderElectionListenerImpl instance will start that JobTracker in ACTIVE mode.

- **When current ACTIVE JobTracker goes down**

  LES will detect that current ACTIVE JobTracker has gone down. It will trigger leader election among all available STANDBY JobTrackers. Only one of those JobTrackers will be elected by LES as a leader. It will notify that JobTracker’s registered LeaderElectionListenerImpl instance by calling standByToActive method on it. LeaderElectionListenerImpl instance will start that JobTracker in ACTIVE mode.

- **When Zookeeper service becomes available again**

  When Zookeeper service is unavailable, all the JobTrackers go to NEUTRAL state. After Zookeeper service becomes available again, LES will trigger leader election among all available NEUTRAL JobTrackers. Only one of those JobTrackers will be elected by LES as a leader. It will notify that JobTracker’s registered LeaderElectionListenerImpl instance by calling neutralToActive method on it. LeaderElectionListenerImpl instance will start that JobTracker in ACTIVE mode.

2.0.3.2 STANDBY State

JobTracker can go to STANDYBY state based on following scenarios

- **When multiple JobTrackers are start together**

  LES will only choose one of all started JobTrackers as leader. It will notify all other JobTracker’s registered LeaderElectionListenerImpl instances by calling startInStandby method on each instance. LeaderElectionListenerImpl instance will start that JobTracker in STANDBY mode.

.

- **When Zookeeper service becomes available again**

  When Zookeeper service is unavailable, all the JobTrackers go to NEUTRAL state. After Zookeeper service becomes available again, LES will trigger leader election among all available NEUTRAL JobTrackers. Only one of those JobTrackers will be elected by LES as a leader. It will notify all other JobTracker's registered LeaderElectionListenerImpl instances by calling neutralToStandby method on each instance. LeaderElectionListenerImpl instance will start that JobTracker in STANDBY mode.

2.0.3.3 NEUTRAL State

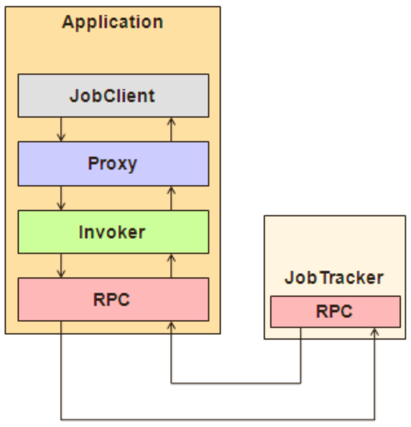
- **When Zookeeper service becomes unavailable**

  This state is mainly to handle Split Brain scenario. If zookeeper service is not available, then we can't determine Active and Standby JobTracker unambiguously. It will notify ACTIVE JobTracker's registered LeaderElectionListenerImpl instance by calling activeToNeutral method and STANDBY JobTrackers registered LeaderElectionListenerImpl instances by calling standByToNeutral method. JobTracker remains in this state until the Zookeeper service is available.

2.0.4 JobClient & TaskTracker Redirection

When currently ACTIVE JobTracker goes down and a STANDBY JobTracker is elected as ACTIVE, JobClients & TaskTrackers should also get redirected to newly ACTIVE JobTracker.

2.0.4.1 Default Invocation Handler



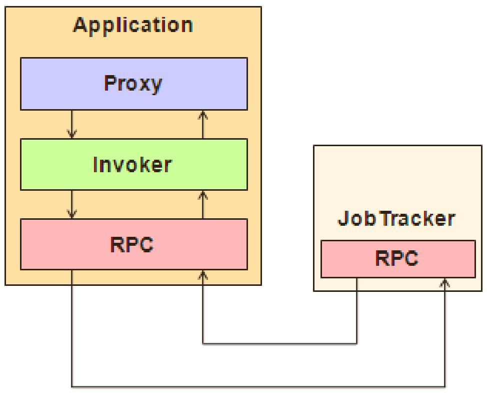
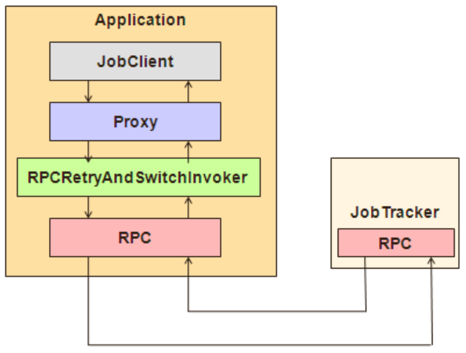


Fig. 3 Default JobClient Call Flow **Fig. 4 Default TaskTracker Call Flow**

When application submits a job, internally JobClient instance is created and initialized.

1. As part of JobClient initialization, a Proxy to JobTracker is created
2. Creation of proxy requires implementation of InvocationHandler. Hadoop's RPC has Invoker API which implements this interface and is responsible for sending/receiving data to/from JobTracker using RPC serialization/de-serialization mechanism.
3. This solution takes advantage of the concept of Invoker and implements its own RPCRetryAndSwitchInvoker.

2.0.4.2 RPCRetryAndSwitchInvoker



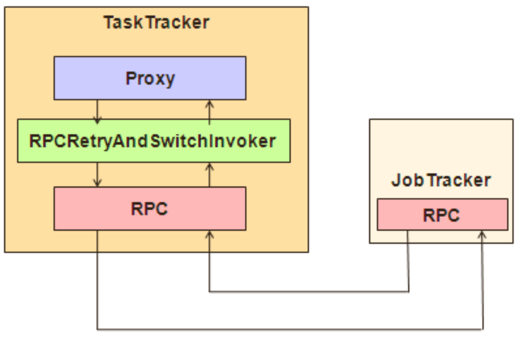


Fig. 3 New JobClient Call Flow **Fig. 4 New TaskTracker Call Flow**

1. RPCRetryAndSwitchInvoker replaces the default Invoker and adds the retry and switch mechanism for JobClient and TaskTracker redirection.
2. It gets the list of all JobTracker addresses using a newly introduced

configuration called as **jobtracker.servers.** Refer to 2.0.5 Configuration section for more information on this configuration

3. It uses this list in a circular queue fashion. Whenever a connection to JobTracker fails, it will try to connect to next address in that queue.
4. Retry and Switch logic will be executed in separate thread than main thread but main thread is made to wait for time mentioned by another configuration called as **jobtracker.await.timeout**. This configuration is to give enough time for RPCRetryAndSwitchInvoker to get connection to ACTIVE JobTracker from a given list of JobTrackers.
5. In case of JobClient, default value of **jobtracker.await.timeout** configuration is kept as 120 seconds based on our tests where there were only 2 JobTrackers. If there is no ACTIVE JobTracker available for some reason or RPCRetryAndSwitchInvoker is not able to get connection to ACTIVE JobTracker then main thread returns after timeout mentioned by **jobtracker.await.timeout** configuration
6. In case of TaskTracker, default value of **jobtracker.await.timeout** configuration is kept as Long.MAX_VALUE seconds. Because TaskTracker should keep trying until it gets connection to ACTIVE JobTracker

2.0.5 Configurations

This solution requires some standard and some new configurations to be set to support some JobTracker High Availability Scenarios. Those configurations are explained in **Table 1** and **Table 2** below.

2.0.5.1 Standard Configurations

Following first table explains required MapReduce standard configurations

Configuration	**Description**
hadoop.job.history.location	Set this location to HDFS e.g. hdfs://myjthost:9000/myhistory

	So that any active JobTracker can access the history of a Job executed by any other JobTracker.
mapred.jobtracker.restart.recover	Set it to true so that in case the Active JobTracker crashes in the middle of Job execution, the newly active JobTracker can recover the Job.
mapred.job.tracker.persist.jobstatus.active	Set it to true so that JobClient will get the status of the Completed Job even after JobTracker switch has happened.
mapred.job.tracker.persist.jobstatus.hours	Set this to a value greater than 0. Make sure the time difference between Active and Standby JobTracker machines is also considered while setting value for this configuration.

Table 1

2.0.5.2 New Configurations

Following first table explains new configurations which are required by this solution to support high availability scenarios.

Configuration	**Description**
jobtracker.servers	This configuration specifies comma separated list of the addresses of all JobTrackers that will participate in HA. JobClient and TaskTracker will use this list to try to connect to newly active JobTracker in case of switch.

jobtracker.await.timeout	The period for which the application thread should wait until connection to active JobTracker is obtained.

Table 2

2.0.6 Test Results

In our test environment with a 100 node cluster and 1000 Jobs, it was measured that STANDBY JobTracker can detect the failure of ACTIVE JobTracker and becomes ACTIVE and starts serving requests in less than 1 minute. This includes failure detection time also.