High Availability In YARN Resource Manager using SQL Based StateStore

Lavkesh Lahngir
Inmobi

Agenda

- High Availability in Hadoop
- RM HA
- StateStore Requirements
- StateStore Implementations
- SQL Based StateStore
- InMobi Deployment

High Availability in Hadoop

- Why HA
- Key Services with HA
  - NameNode
  - ResourceManager
- Key Difference
  - Namenode has a hot standby in contrast to Resource Manager

RM HA - Key components

- RM Restart
  - StateStore [YARN-128]
  - Work-preserving recovery[YARN-556]
- RM Failover: YARN-149
  - Leader Election
  - Client Failovers

RM HA

- RM Starts in standby mode
- Leader Election happens between multiple RMs
  - Zookeeper based ElectorService
- If the RM becomes active
  - It fences the other RM.
  - it recovers data from state-store and starts internal services.
- RM HA is not hot standby.

StateStore

- What is Stored
  - Stores data about currently running Applications and AppAttempts
  - Stores delegation tokens
  - Stores master keys
- Runtime
  - Store/update application data, delegation tokens and keys
  - Deletes tokens upon expiry
  - Deletes application data beyond max allowed applications
- Recovery
  - Reads the data from the populates the in memory data structures

StateStore Fencing

- What is fencing ?
  - Active RM gets an exclusive write access to the state store.
  - Only one RM should be able to write to State Store
  - NameNode Comparison: *ssh based fencing*

File Based StateStore

- FileSystemRMStateStore
  - Local/HDFS file systems
  - Stores data in the hierarchical structure
- Fencing
  - No fencing implementation
- Our Experience
  - Reading the state-store very slow
    - 30 min for 2000 apps
  - Number of tokens are more problematic than number of apps

Zookeeper Based StateStore

- ZKRMStateStore
  - Recommended state store for HA
  - Uses ZK node hierarchy
- Fencing
  - Root node will have credential for only one RM
  - While writing to state store check if root node is writable
  - zkclient.multi() is used to ensues that all or none of the operations are succeeded

Zookeeper Based StateStore

- Data stored in ZK was large
  - Crash!

- Philosophically, not the right one
  - Not meant to be used as a data store
  - Meant for functionality such as distributed coordination

SQL Based StateStore: Motivation

- Designed for data storage
- Consistent and Available store
- Fencing ease
- Scale of Updates
- Inhouse usage of databases

SqlRMStateStore: Structure

- Key Tables:
  - credentialtable, applicationtable, attempttable, rmdelegationtable, rmmasterkeystable
- Data is stored in the key,value format with indexing on keys
- Example:
  - Data for appattempt attempt_100_01
    - Table: “attempttable”
    - Key: “application_100/attempt_100_01”

SqlRMStateStore: Structure

- All operation of statestore are SQL transactions
- Transaction level is set to TRANSACTION_REPEATABLE_READ
  - Which means it does not allow dirty reads
  - credentialtable is accessed via a shared lock.
- Retries on SQLExceptions

SqlRMStateStore: Fencing

- Table “credentialtable” contains the *secret* for the active RM
- When RM becomes active it stores the *secret* in the table
- Periodically checks that if state store has the credential for current RM.

SQLStateStore: Fencing

- Transaction semantics
    a. Begin Transaction
    b. Update stuff
    c. Check the credentials (Holds a shared lock on credentialtable)
    d. If the key is same as my secret then commit else abort

SqlRMStateStore: HA for SQL Server

- Zookeeper and hdfs already have builtin redundancy
- For sql we have two choices:Synchronous or Asynchronous replication
- Async replication can be a problem in rare cases
- We adopted for 1 master and 2 slave sync replication

Some numbers

- 5 clusters at inmobi
- 70000 mapreduce plus spark job in a day on biggest cluster
- 2000 max completed applications
- Tested the postgres setup with average of 400 to 500 StateStore operation per second
- For 70k jobs per day, on average we need 200 state-store operations per minute

Questions?