Agenda

- Coordination Engine abstraction: [HADOOP-10641](#)
- Replicated Namespace for HDFS: [HDFS-6469](#)
  - Overview of Architecture and Implementation
- Technical discussion
- Logistics:
  - Jiras, Branching
  - Interaction with other components, projects
- Sources
  - [ConsensusNode design document](#)
  - [Coordinating Metadata Replication. Hadoop Summit Amsterdam, April 2014](#)
Coordination Engine

- Abstraction based on State Machine Approach
Coordination Engine

Sequencer: *Determines the order in which operations occur*

- **State Machine Approach** *(Wikipedia)*
  1. Receive client requests, interpreted as Inputs *(Proposal)* to the State Machine
  2. Choose an ordering *(Coordination)* for the Inputs
  3. Execute Inputs *(Agreement)* in the chosen order on each server
  4. Respond to clients with the Output from the State Machine

- **Coordination Engine** allows to agree on the order of events submitted to the engine by multiple proposers
  - Anybody can Propose
  - Single Agreement every time
  - Learners observe the same agreements in the same order

- Determinism of agreement execution
Single-Node Coordination Engine

*Simple but lacks reliability*

- Easy to Coordinate
  - Single NameNode is an example of a simple Coordination Engine
  - Performance and availability bottleneck
  - Single point of failure
Distributed Coordination Engine

A reliable approach is to use multiple nodes

- Distributed Coordination Engine consists of nodes
  - Node Roles: Proposer, Learner, and Acceptor
  - Each node can combine multiple roles

- Distributed coordination
  - Multiple nodes submit events as proposals to a quorum of acceptors
  - Acceptors agree on the order of each event in the global sequence of events
  - Learners learn agreements in the same deterministic order
Consensus Algorithms

Consensus is the process of agreeing on one result among a group of participants

- Coordination Engine guarantees the same state of the learners at a given GSN
  - Each agreement is assigned a unique Global Sequence Number (GSN)
  - GSNs form a monotonically increasing number series – the order of agreements
  - Learners start from the same initial state
  - Learners apply the same deterministic agreements in the same deterministic order
  - GSN represents “logical” time in the coordinated system

- Examples of Consensus algorithms
  - PAXOS proven to tolerate a variety of failures
  - Two phase commit
  - Raft
  - ZAB
interface Proposal extends Serializable {}

interface Agreement<L, R> extends Serializable {
    abstract R execute(L callBackObject) throws IOException;
}

abstract class ConsensusProposal<L, R>
    implements Proposal, Agreement<L, R> {
    public Serializable getProposerNodeId();
    public boolean equals(Object o);
    public int hashCode();
    public String toString();
    // To be defined
    void readObject(ObjectInputStream in);
    void writeObject(ObjectOutputStream out);
    R execute(L callBackObject) throws IOException;
}
Interface: CoordinationEngine

```java
public interface CoordinationEngine {

    void initialize(Configuration config);
    void start();
    void stop();

    ProposalReturnCode submitProposal(
        Proposal proposal,
        boolean checkQuorum)
    throws ProposalNotAcceptedException;

    long getGlobalSequenceNumber();
    void checkQuorum() throws NoQuorumException;

    void registerHandler(AgreementHandler<? super handler>);
}
```
HDFS

- State of the Art
HDFS Cluster

Active-Standby Architecture

- Single active NameNode
  Single StandbyNode
- Thousands of DataNodes
- Tens of thousands of HDFS clients
Standard HDFS operations

- **Active NameNode workflow**
  1. Receive request from a client,
  2. Apply the update to its memory state,
  3. Record the update as a journal transaction in persistent storage,
  4. Return result to the client

- **HDFS Client (read or write to a file)**
  - Send request to the NameNode, receive replica locations
  - Read or write data from or to DataNodes

- **DataNode**
  - Data transfer to / from clients and between DataNodes
  - Report replica state change to NameNode(s): new, deleted, corrupt
  - Report its state to NameNode(s): heartbeats, block reports
Consensus Node

- *Coordinated Replication of HDFS Namespace*
Replicated Namespace

*Coordination Engine provides consistency of multiple namespace replicas*

- Replicated NameNode is called a **ConsensusNode** or **CNode**
- ConsensusNodes play equal *active* role on the cluster
  - Provide *write and read access* to the namespace
- The namespace replicas are consistent with each other
  - Each CNode maintains a copy of the same namespace
  - Namespace updates applied to one CNode propagated to the others
- Coordination Engine establishes the global order of namespace updates
  - All CNodes apply the same *deterministic* updates in the same *deterministic* order
  - Starting from the same initial state and applying the same updates = consistency
Coordinated HDFS Cluster

*Multiple active Consensus Nodes share namespace via Coordination Engine*

- Independent CNodes – the same namespace
- Load balancing client requests
- Proposal, Agreement
- Coordinated updates
Coordinated HDFS operations

Updates to the namespace when a file or a directory is created are coordinated

- ConsensusNode workflow
  1. Receive request from a client
  2. Submit proposal to update to the Coordination Engine
     Wait for agreement
  3. Apply the agreed update to its memory state,
  4. Record the update as a journal transaction in persistent storage (optional)
  5. Return result to the client

- HDFS Client and DataNode operations remain the same
Strict Consistency Model

One-Copy-Equivalence as known in replicated databases

- Coordination Engine transforms namespace modification proposals into the *global sequence of agreements*
  - Applied to namespace replicas in the order of their Global Sequence Number

- ConsensusNodes may have different states at a given moment of “clock” time
  - As the rate of consuming agreements may vary

- **CNodes have the same namespace state when they reach the same GSN**

- One-copy-equivalence
  - each replica presented to the client as if it has only one copy
Consensus Node Proxy

*Reads do not modify namespace can be directed to any ConsensusNode*

- **CNodeProxyProvider** – a pluggable substitute of FailoverProxyProvider
  - Defined via Configuration

- **Main features**
  - Randomly chooses CNode when client is instantiated
  - Sticky until a timeout occurs
  - Fails over to another CNode
  - Smart enough to avoid SafeMode

- **Further improvements**
  - Take into account network proximity
Stale Read Problem

A few read requests must be coordinated

1. **Same client fails over** to a CNode, which has an older namespace state GSN
   - CNode1 at GSN 900: mkdir(p) → ls(p) → failover to CNode2
   - CNode2 at GSN 890: ls(p) → directory not found

2. One client modifies namespace, which needs to be seen by other clients
   **MapReduce use case:**
   - JobClient to CNode1: create job.xml
   - MapTask to CNode2: read job.xml → FileNotFoundException

1) Client connects to CNode only if its GSN >= last seen
   - May need to wait until CNode catches up

2) Must coordinate file read
   - Special files only: configuration-defined regexp
   - CNode coordinates read only once per file, then marks it as coordinated
   - Coordinated read: submit proposal, wait for agreement
Block Management

- Block Manager keeps information about file blocks and DataNodes
  - Blocks Map, DataNode Manager, Replication Queues
  - New block generation: `<blockId, generationStamp, locations>`
  - Replication and deletion of replicas that are under- or over-replicated

- Consistent block management problem
  - Collision of BlockIDs if the same ID generated by different CNodes
  - Over-replicated block: if CNodes decide to replicate same block simultaneously
  - Missing block data: if CNodes decide to delete different replicas of the block

- New block generation – all CNodes
  - Assign nextID and nextGenerationStamp while processing the agreement

- Replication and deletion of replicas – a designated CNode (Block Replicator)
  - Block Replicator is elected using Coordination Engine and reelected if fails
  - Only Block Replicator sends DataNode commands to transfer or delete replicas
Low Intrusion Implementation

- ConsensusNode extends NameNode
  - Separate module (subproject) in source code tree
- Subclassing of FSNamesystem and RPCServer may be avoided
  - By restructuring methods in FSNamesystem
  - UniversalNamespaceProposal – coordinate RCP calls without deserialization
- Compatibility
  - No modification to RPC protocol
  - No modification to fsimage, edits
- Client needs to specify CNodeProxyProvider class in configuration
- No change in DataNode: reuse HA configuration
FAQ

- From jiras and communication
Coordination Engine FAQs

- Why Hadoop should care about Coordination? What are the benefits?
- Why this belongs to Hadoop?
- Your design is for Paxos. What about other consensus algorithms?
- Reference implementation using ZooKeeper?
ConsensusNode FAQs

- Complexity of the design?
- What are the advantages for HDFS?
- Impact on performance?
- Determinism requirement while executing agreements?
- Double journaling?
- Active-active vs active+read-only-standby(s)?
- Do you just want to plug in WANdisco coordination engine?
- Are you going to abandon BackupNode?
- Technical Discussion
- Logistics

And Lunch