# Supporting Observer Reads from Routers

Author: Simbarashe Dzinamarira
Contributions from: Owen O'Malley

## Background

### Consistent reads from Standby

HDFS-12943 introduced Observer namenodes. These are Standby namenodes that allow reads. This feature allows reads to be offloaded from the Active namenode to Observer namenodes thereby improving both the throughput and latency of metadata operations.

To make reads from Observers consistent, a client sends the Observer the last state ID that the client received from the Active. The Observer then waits until its state reaches this last seen state before responding to the client's request. To support consistent reads in the presence of third-party communicate, an *msync* call was added for clients. This is used to explicitly query the last seen state ID from the Active namenode instead of getting it implicitly in the response to another request.

The consistency model for Observer reads is formulated as the following:
**If a client c1 sees or modifies an object state at modId1 at timet1, then in any future time t2>t1, c1 will see the state of that object at modId2>=modId1**

### Router based federation

HDFS-10467 added an RPC routing layer that provides a federated view of multiple HDFS namespaces. The routers maintain a mount-table and route clients calls to the appropriate namenodes backing the mount points.

# Interaction between router-based federation and observer reads

Consistency for observer reads is enforced through the state ID that is propagate from an Active namenode, through the client and then to the Observer namenode. Without routers, each client only interacts with one active namenode so a single value is sufficient to capture the state ID.

When Routers are involved, a single client can interact with multiple namenode that are federated behind the router therefore single value in the RPCHeader is not sufficient to capture the last seen states of these namenodes.

Restating the consistency model for Observer reads:
**If a client c1 sees or modifies an object state at modId1 at timet1, then in any future time t2>t1, c1 will see the state of that object at modId2>=modId1**

To maintain this consistency model, there are two main approaches

1) Before a router sends each read to the Observer, it fetches the last seen state ID from the corresponding active namenode.
2) Propagate the last seen state ID for all namespaces to the client in rpc responses, and receive it from the client in requests.

| Approach | Pros | Cons |
|---|---|---|
| (1) MSYNC on every reads | RPC header size unchanged | Extra network roundtrip for every read |
| (2) Propagate map of stated to client | Number of network hops unchanged (expect for msync) | Larger RPC header |

Design decision A

**We choose approach (2), propagating all namespace state IDS to the client.**
The cost of the larger RPC header is less than that incurred from an extra network round-trip, so

Design decision B

**Clients cannot opt-out of observer reads through the routers.**
Routers are meant to provide a unified view of a federated namespace. Clients should not know whether namenodes in a particular namespace have an HA setup with Observer namenodes. We therefore choose to let only the routers enable or disable observer reads.

Allowing clients to opt-out is a small change that can be added later as a performance optimization (we have an implementation with this feature). However, we exclude this optimization from the core design discussion.

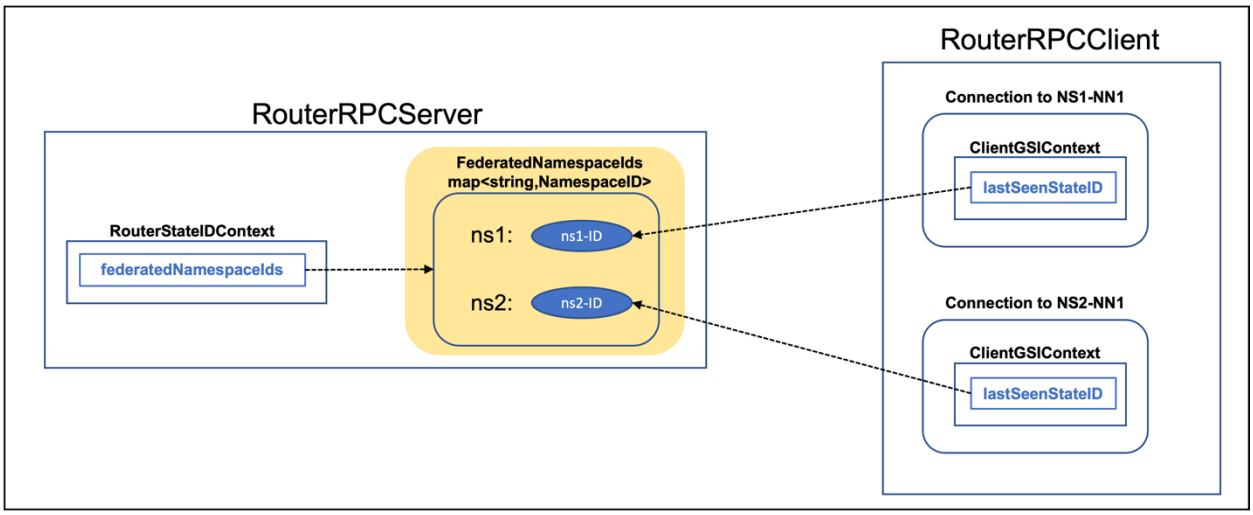
# Summary of proposed changes

1. Include the Observer state in the list of namenode states the Routers track.
2. Extend NamenodeResolver with the ability to prioritize observers so that reads can be first attempted on observers.
3. Add a nameservice to stateID mapping, called **nameserviceStateIds,** to the RPCHeader. Note, this can be just an obscure byte array since the client doesn't need to parse it.
4. Add a composite alignment context, called the **FederatedNamespaceIds,** to the router. This contains a map from nameservices to NamespaceId objects specific to each nameservice.
5. Communication between routers and clients
    a. A router uses the **FederatedNamespaceIds** object to update the **nameserviceStateIds** map in the RPCResponseHeader sent to clients.
    b. A router updates the **FederatedNamespaceIds object** with information in the **nameserviceStateIds** map in the RPCRequestHeader.
6. Communication between routers and namenodes.
    **a.** When communicating with a Namenode, a router uses the a ClientGSIContext linked to a NamespaceId contained within the composite **FederatedNamespaceIds.**
        i. StateID updates received in RPCResponseHeaders from Namenodes are implicitly integrated into the **FederatedNamespaceIds** when applied to the nameservice specific ClientGSIContext.
        ii. Updates to the **FederatedNamespaceIds** will also be implicitly included in the RPCRequestHeaders sent to NameNodes.
7. When a client does an *msync* call to a router, the router fans out this call to all nameservices in order to fully update the **FederatedNamespaceIds.**
8. For old clients which do not have the **nameserviceStateIds** map**,** the router always does an *msync* before each read call so that it obtains the latestSeenStateID for that nameservice

# FederationNamespaceIds

The FederatedNamespaceIds onbject (highlighted in yellow below) stores the last seen stateIDs seen by a router. These stateIDs are received either from clients requests or namenodes responses to the router. Similarly, the map is used to set the RPC head when the router sends RPC requests to namenodes, or sends RPC responses to clients.

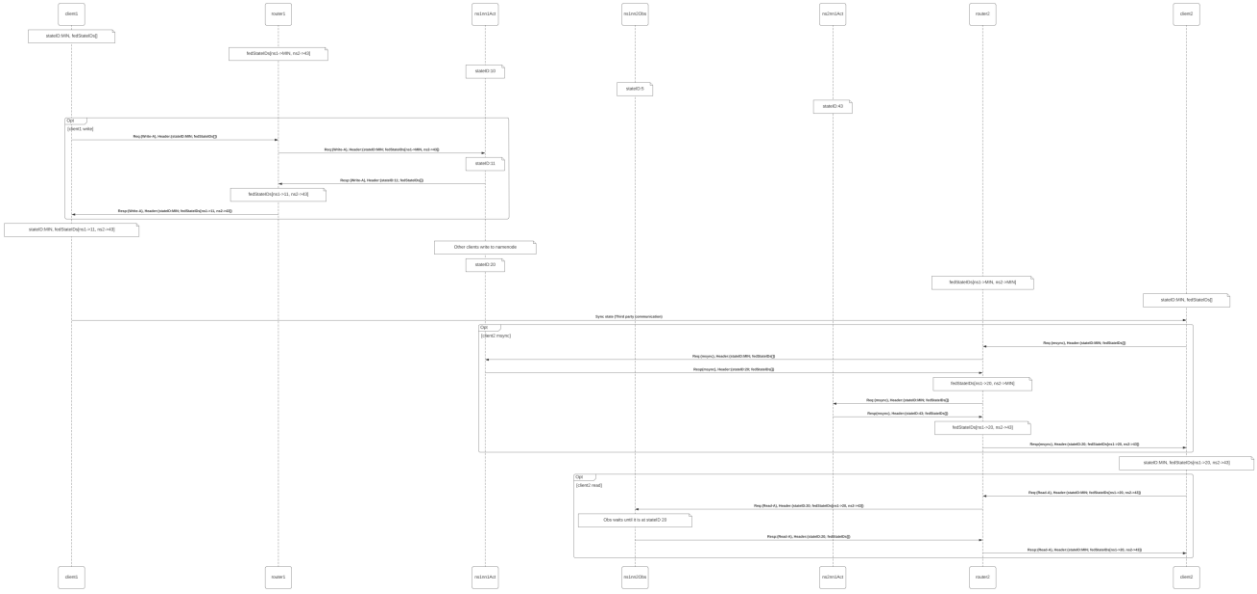
- For Router to Client communication, the FederatedNamespaceIds object is used directly by the RouterStateIDContext.
- For Router to Namenode communication, the FederationNamespaceIds object is used indirectly because the ClientGSIContexts in the routers reference elements of the FederationNamespaceIds' map.




Dotted lines represent object references

# Reads your own writes (pdf)

Third party communication (pdf)



# Implementation breakdown

1. Updates to Hadoop-common: https://github.com/apache/hadoop/pull/4584
    a. Modifies RPCHeader proto
2. IPC changes: https://github.com/apache/hadoop/pull/4311
    a. Creates classes to propagate FederatedState between clients and routers
3. Directing router reads to observers: https://github.com/apache/hadoop/pull/4127
    a. Select observed as target for read operations.
    b. For old clients without federated state, performs msync for every call.
    c. Implements router msync that fans across all namespaces.