Disaster Recovery Solution for Apache Hadoop

Contributors: Chen Haifeng (haifeng.chen@intel.com), Gangumalla Uma (uma.gangumalla@intel.com), Dey Avik (avik.dey@intel.com), Li Tianyou (tianyou.li@intel.com), Purtell, Andrew (andrew.k.purtell@intel.com)

Contents

Introduction

Apache Hadoop is architected to operate efficiently at scale for normal hardware failures within a datacenter. It is not designed today to handle datacenter failures. Although HDFS is not designed for nor deployed in configurations spanning multiple datacenters, replicating data from one location to another is common practice for disaster recovery and global service availability. There are current solutions available for batch replication using data copy/export tools. However, while providing some backup capability for HDFS data, they do not provide the capability to recover all your HDFS data from a datacenter failure and be up and running again with a fully operational Hadoop cluster in another datacenter in a matter of minutes. For disaster recovery from a datacenter failure, we should provide a fully distributed, zero data loss, low latency, high throughput and secure HDFS data replication solution for multiple datacenter setup.

Targets

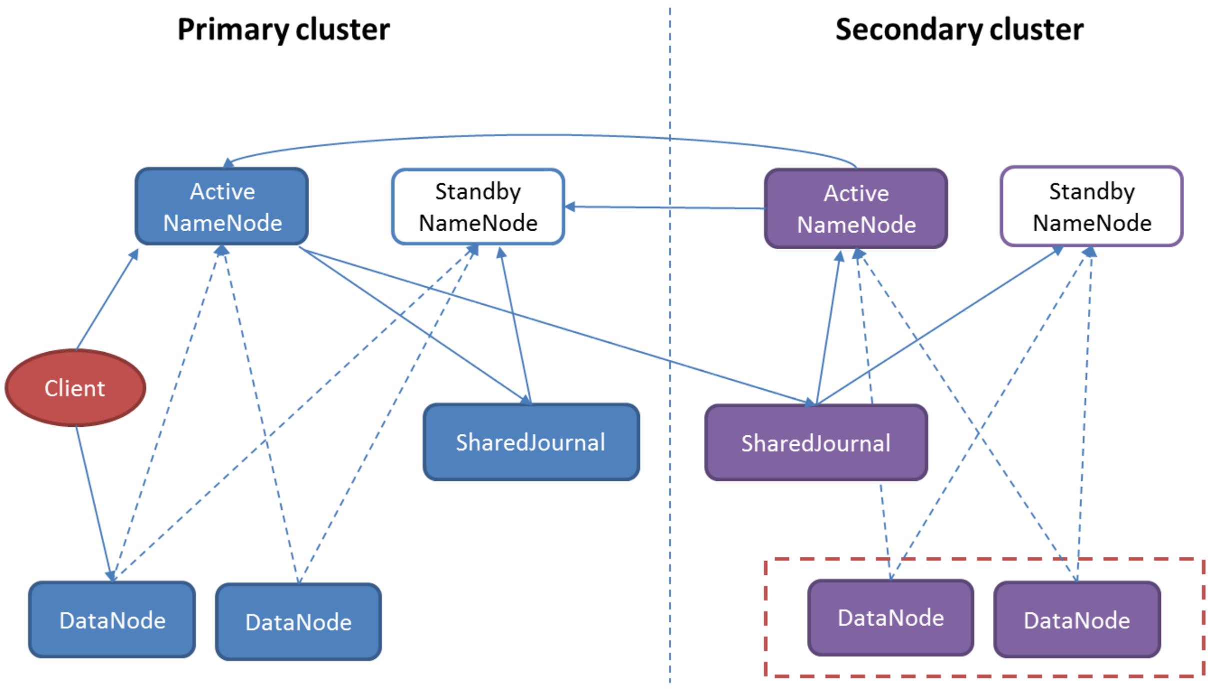
The following list the targets of this design:

1. Support both synchronous writing and asynchronous replication for data and namespace.
2. Configuring and managing of the disaster recovery feature should be simple.
3. All the core disaster recovery functionalities are achieved by using or improving the existing HDFS architecture with nice fits of concepts.

Approach

The basis of this solution is to have one or more secondary (mirror) Hadoop clusters which will be continuously updated with the data from the primary cluster in either a synchronous method or an asynchronous method by utilizing and improving the existing HA feature, data block replication and data block pipelining. In this solution, we support both synchronously writing and asynchronously replication across datacenters for both namespace and data block.

The following architecture diagram shows the overall architecture of this solution.



The following are the key design points,

1. By improving the HDFS to support the concept of secondary (mirror) cluster, we can have a single primary cluster and multiple secondary cluster across multiple datacenters. Each cluster will still has one Active NameNode and one Standby NameNode. They will behave different according to their cluster role.
2. There are DataNodes in both primary cluster and the secondary clusters. The DataNodes will only heartbeat and report blocks to the NameNodes of its local cluster. That's to say, all the DataNodes in the primary cluster heartbeat and report blocks to Active NameNode and Standby NameNode in primary cluster. And all the DataNodes in the secondary clusters heartbeat and report blocks to local Active NameNode and Standby NameNode in secondary cluster.
3. Writing data directly to secondary cluster will have performance drop, but for some users may need more data availability than performance. So, we target to provide two options to the users in configurable way. By default we keep the asynchronous data replications to secondary clusters.
4. To achieve synchronous data writing, we can provide new placement policy in primary cluster which needs to make sure that it is keeping the secondary cluster DataNode in pipeline along with primary DataNodes. Secondary cluster DataNodes always be at the end of the pipeline. So, primary cluster should know about the available DataNodes in secondary cluster. Secondary cluster Active NameNode will heartbeat to primary cluster NameNodes with a special command called DR_DN_AVAILABLE (contains DatanodeInfo with space, load, etc.). Primary NameNodes keep this details and will be used by placement policy while selecting node for pipeline. To satisfy real synchronous data replication, we make sure at least one DataNode selected from secondary cluster. But we will not keep this as strict requirement until user explicitly mention strong replication need. Otherwise replication will happen asynchronously via replication scheduling mechanism.

As a potential optimization, Active NameNode in secondary cluster can heartbeat to only one of the NameNode in the primary cluster. If the information is needed in the other, the NameNode in the primary cluster can share/forward the information to the other. In this way, the cross sites heartbeat communication can be reduced. To keep things simple and impacts small, in the first version, we assume that Active NameNode in secondary cluster will heartbeat to both NameNodes in the primary cluster.

5. To achieve asynchronous data replication, no remote DataNodes will be selected by the block placement policy when the client is writing data in primary cluster. Here secondary cluster Active NameNode schedule the block replications. But the secondary cluster needs to know the block locations from primary to schedule replication. So, the secondary cluster Active NameNode will find blocks for replication and select set of local DataNodes to transfer block. Since secondary cluster not aware of block locations at primary cluster, it will just include batch of replication commands (DR_REPLICATION_REQUEST (contains blocks with targets selected)) to primary cluster NameNodes via heartbeats. On processing the commands at Active NameNode of primary cluster, it will just find the source DataNode and schedule replication commands to it via existing BLOCK_TRANSFER command.

   As a potential and future optimization, we can use Standby NameNode in the primary cluster to process DR_REPLICATION_REQUEST to reduce the workload on Active NameNode in the primary cluster.
6. When the admin wants to configure synchronous namespace replication, he need to keep a new Shared Journal at the secondary cluster and configure Active NameNode in the primary cluster writes edit logs to both the local Shared Journal and Shared Journal in secondary cluster. In this way, the edit logs are guaranteed to be written successfully to the secondary cluster. This is to avoid edit transaction loss when primary cluster met unrecoverable crash in sync mode. Standby NameNode in primary cluster will still tail edit logs from the local Shared Journal. And both NameNodes in the secondary cluster will tail edit logs from secondary Shared Journal.
7. When the admin wants to configure asynchronous namespace replication, Active NameNode in the primary cluster only writes edit logs to the local Shared Journal. Both NameNodes in the secondary cluster will tail edit logs asynchronously from the Shared Journal in primary secondary cluster. There will still be a Shared Journal configured in the secondary cluster for acting writing Journal when secondary cluster failovers to primary cluster role.
8. The primary cluster Active NameNode is same with normal Active NameNode with some additional functionalities, such as handling DR_DN_AVAILABLE and block placement policy to support synchronous block writing and handling DR_REPLICATION_REQUEST for replicates. The secondary cluster Active NameNode is similar with normal Active NameNode but with several significant differences and additional functionalities. The Active NameNode in secondary cluster will not allow any writes operations to the file system through client API. Instead, it will behave like a Standby NameNode to tail from Shared Journal and update its namespace. It also schedule active replications and heart beating the commands (DR_DN_AVAILABLE, DR_REPLICATION_REQUEST) to primary cluster. To simplify the things, current version targets to provide manual switch over to secondary clusters. So, the admin explicitly issues command to secondary cluster to convert the secondary cluster to primary cluster role.

Design

Concept of Region

From the design, we would support interaction between multiple clusters and these clusters are deployed in different datacenters. The NameNodes in primary cluster take different roles with the NameNodes in secondary cluster. And they need to behave differently according to the cluster role.

To help the NameNode distinguish the role of cluster, we use the concept of “region”. Each site or datacenter are conceptually considered as a region and identified by a “regionID”. NameNode knows the region it belongs to and also knows the fact of the current primary region. Based on this information, NameNode can properly act and interact with the components in other regions according to the design.

Also, in cluster node setups, configurations should include all the available regions and the NameNodes and Journal URI of the regions. And this part of information should be same across all setups.

Cluster Joining

When a new disaster recovery cluster needs to join to an existing configuration, we use the same bootstrap process as the current implementation of Standby NameNode joining.

First, the admin needs to bootstrap Shared Journal in the secondary cluster by downloading the edit logs from Shared Journal at primary cluster.

Standby NameNode of the secondary cluster will download and apply the FSImage of a proper checkpoint and tail properly the edit logs from the local Shared Journal.

After the joining process, the asynchronous data block replication can be scheduled by Active NameNode of secondary cluster to localize all the remote data blocks. And the replication process is asynchronous and how long it will last depends on the data size of the primary HDFS cluster. During the catchup period, the secondary cluster may enter into SafeMode as it finds that there are too many missing blocks. We need to make sure that even if the secondary cluster is in SafeMode, it doesn’t prevent tailing the edit logs, replicating of remote blocks and writing synchronous data blocks.

Synchronous Writing

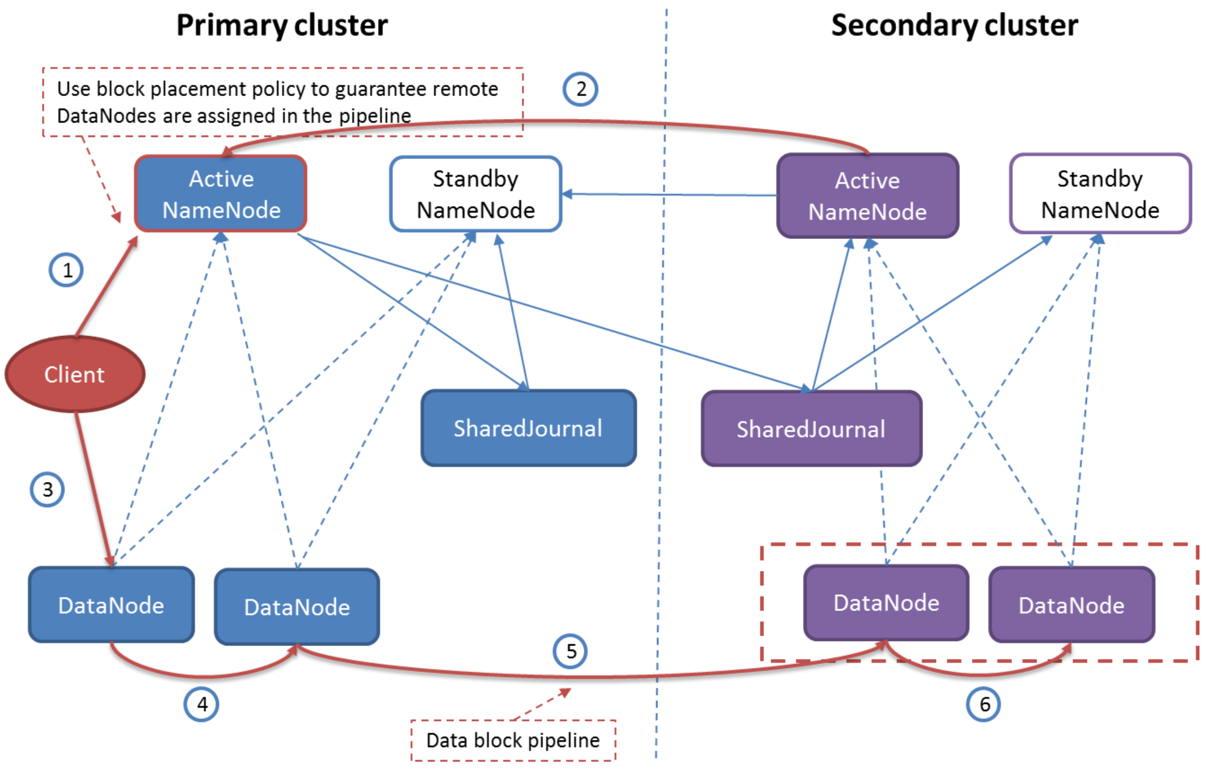
Synchronous writing means two things. First, when the client is writing data to the HDFS in the primary cluster, the data will also write to the secondary cluster. So that when disaster happens in the primary cluster, the data already claimed to be written will not be lost. Second, if one namespace operations is performed successfully in primary cluster, we need it also be in secondary cluster eventually even in the case of primary cluster disaster.

Writing data directly to secondary cluster will have performance drop. But for the users need more data availability than performance on critical data, synchronous namespace and data writing would be the right choice. Please note that in this design, the synchronous data writing and asynchronous data replication can coexist in a single configuration. While for namespace, a single configuration can either support synchronous journaling or asynchronous journaling, but not both.

Synchronous Data Writing

The solution will support both synchronous and asynchronous data writing. There are requirements that any data loss for critical data is not acceptable. For this kind of data, we can configure to use synchronous data writing for achieve “Zero Loss”. When the client is writing block data, the data block is pipelined to both local DataNodes in primary cluster and remote DataNodes in secondary clusters.

The following diagram shows the basic workflow for synchronous data writing.



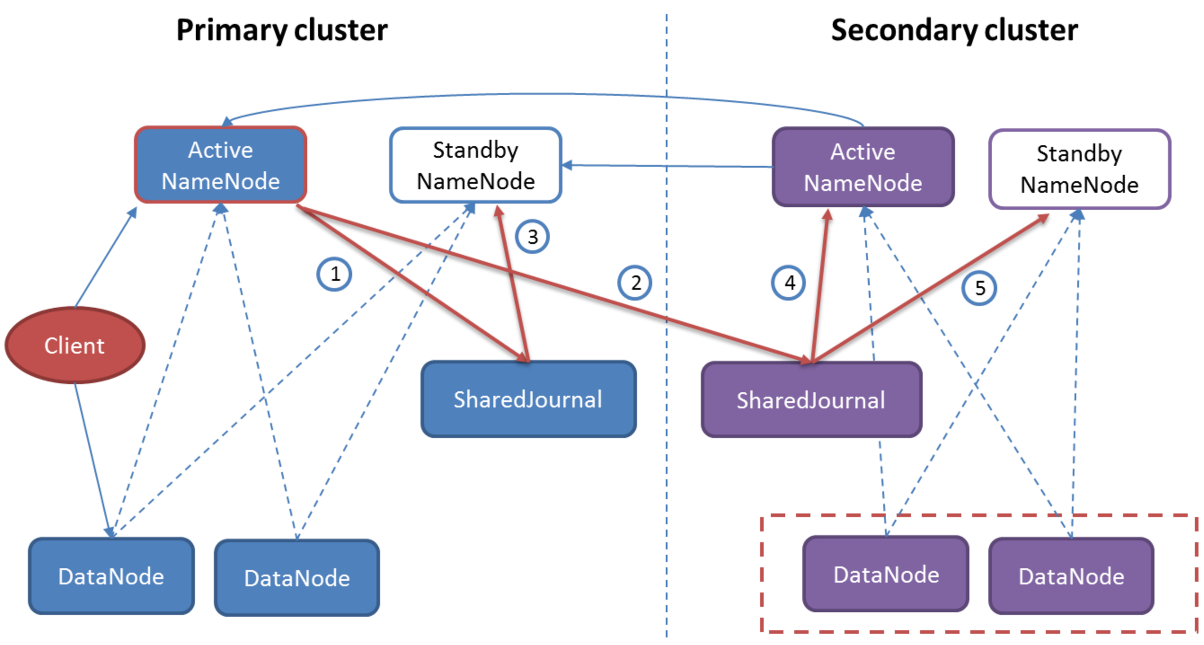
1. When a client is writing a HDFS file, after the file is created, it starts to request a new block. And Active NameNode in primary cluster will allocate a new block and select a list of DataNodes for the client to write to. The new DR block placement policy, Active NameNode can guarantee one or more remote DataNodes from secondary cluster are selected at the end of the pipeline.
2. Primary cluster NameNode knows the available DataNodes in the secondary cluster via heartbeats from secondary cluster’s Active NameNode with the DR_DN_AVAILABLE command. So, latest reported DataNodes will be considered for the secondary cluster pipeline which will be appended to primary cluster pipeline.
3. Upon a successful block allocation, the client will write the block data to the first DataNode in the pipeline and also giving the remaining DataNodes.
4. The first DataNode will continue to write to the following DataNode in the pipeline.
5. The last local DataNode in the pipeline will continue to write the remote DataNode that following.
6. If there are more than one remote DataNodes are selected, the remote DataNode will continue to write to the following DataNode which is local to the remote DataNode. We provide flexibility

to users that they can even configure the secondary cluster replication. Based on the configured replication, secondary nodes will be selected.

Synchronous Namespace Journaling

It is sometimes critical that the namespace edit logs will not be lost when disaster happens. In this solution, it can support synchronous namespace journaling, which means that the primary cluster Active NameNode writes edit logs to multiple Shared Journals and one of them is at the secondary cluster. In this configuration, the secondary cluster Active NameNode and Standby NameNode will tail from Shared Journal in the secondary cluster.

The following shows the basic journaling workflow when multiple Shared Journal is configured.



1. The primary cluster Active NameNode writes the edit logs to Shared Journal in primary cluster.
2. The primary cluster Active NameNode also writes the edit logs to Shared Journal in secondary cluster.
3. The primary cluster Standby NameNode tails the edit logs from Shared Journal of primary cluster.
4. The secondary cluster Active NameNode tails the edit logs from Shared Journal of the secondary cluster.
5. The secondary cluster Standby NameNode tails the edit logs from Shared Journal of the secondary cluster.

Handling Partial Success for Synchronous Namespace Journaling

When doing synchronous namespace journaling, the primary cluster Active NameNode will write both to the local Shared Journal and to remote Shared Journal. We need to deal with the complex situation when the sync to the first Journal succeed but the sync to the second Journal failed, thus leave the two Journal in inconsistent state. In such a situation, gap will happen when tailing from the failed Journal.

Typically, we can handle this with the following steps:

1. The primary cluster Active NameNode will write first to the local Shared Journal before writing to any remote Shared Journals.
2. When a required shared Journal failed, the primary cluster Active NameNode will shut down.
3. When one NameNode in the primary cluster restart or pick up the Active NameNode role, and before doing any log operations, it check with the secondary Journal for the last sync txid.
4. If the last sync txid of secondary Journal matches with the last sync id of primary Journal, it continues as normal.
5. If the last sync txid of secondary Journal doesn't match with the last sync id of primary Journal, it will sync the secondary Journal with the primary Journal and make sure that process is finished before it going on as normal.

If we consider to support only one secondary cluster, there is an alternative approach.

1. The primary cluster Active NameNode will write first to the remote Shared Journal. The reason that we writes to the remote Journal first is we want the remote Journal will always with succeeded edit logs.
2. If remote succeeded, but local shared Journal or local file Journal failed, we shut down Active NameNode.
3. On restart or switch NameNode, it should consider all the Journals to load and roll out the edit logs. The primary would be able see all edits and secondary anyway will have as we ensured to write first to the secondary.

Asynchronous Replication

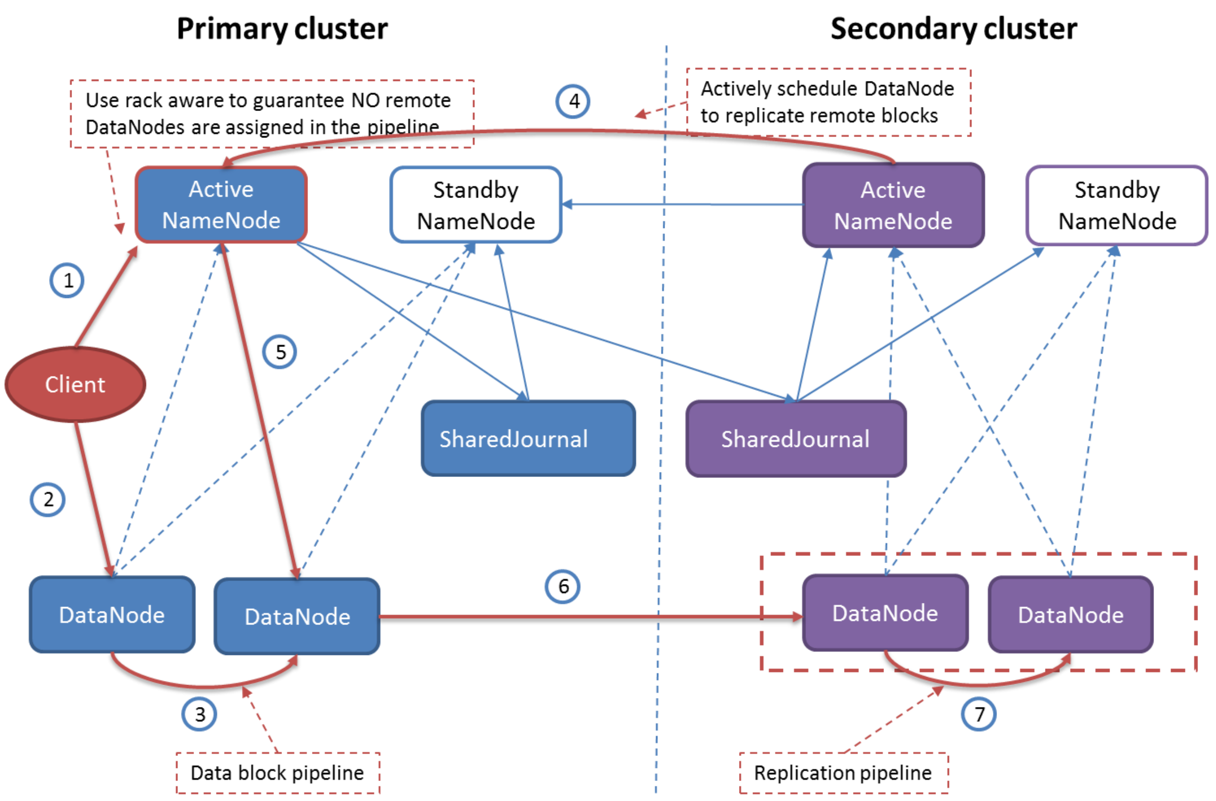
While synchronous data writing and namespace journaling provides data safety, it will give more performance impact. Asynchronous replication can decrease the performance impact to the primary cluster by replicating the data block and namespace asynchronously at the background.

Asynchronous Data Replication

In the real world, limited by the network bandwidth and stability, writing all data synchronously is not possible. Depending on the bandwidth, synchronous data writing to remote clusters may provide unsatisfactory throughput. Synchronous data writing is for only critical data. While for most of other data, asynchronous replication would be common.

Based on configuration, the data that is not so critical can be replicated asynchronous to remote cluster DataNodes.

The following diagram shows the basic workflow for asynchronous data replication.



1. When a client is writing a HDFS file, after the file is created, it starts to request a new block. And the primary cluster Active NameNode will allocate a new block and select a list of DataNodes for the client to write to. For the file which needs only asynchronous data replication, no remote DataNode from secondary cluster is selected for the pipeline at Active NameNode.
2. Upon a successful block allocation, the client will write the block data to the first DataNode in the pipeline and also giving the remaining DataNodes.
3. The first DataNode will continue to write to the following DataNode in the pipeline until the last. But this time the pipeline doesn’t span to the remote cluster.
4. Asynchronously, the secondary cluster Active NameNode will actively schedule to replicate data blocks which are not on any of the local DataNodes. As part of heartbeats it will send DR_REPLICATION_REQUEST which will contain batch of blocks to replicate with target DataNodes selected from secondary cluster. The secondary cluster doesn’t need to aware of real block location in primary cluster.
5. As a result of handling the DR_REPLICATION_REQUEST, the primary cluster Active NameNode takes care of selecting block location and schedules the replication command to corresponding source DataNode at primary cluster.
6. A DataNode will be selected to replicate the data block from one of the DataNodes in primary cluster that hold the block.
7. As a result of the replication pipeline, the local DataNode can replicate the block to other DataNodes in the secondary cluster.

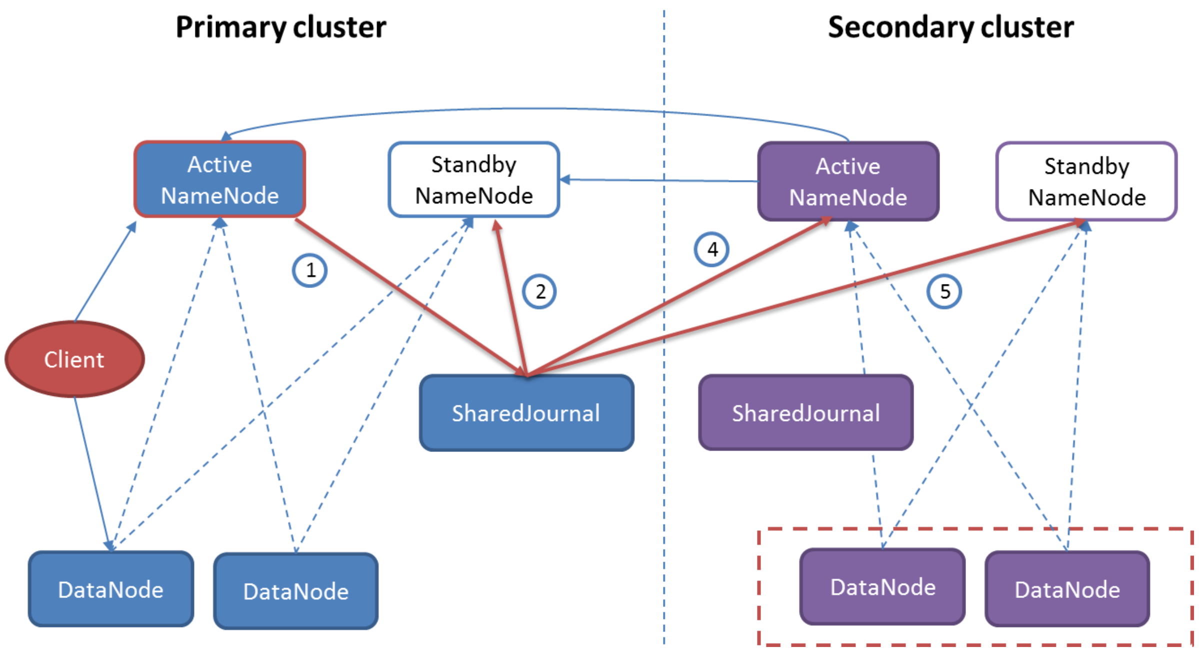
Asynchronous Namespace Journaling

Synchronous journaling to remote clusters means more latency and performance impact. When the performance is critical, the admin can configure an asynchronous edit log journaling.

When the admin configures to use asynchronous namespace replication, the primary cluster Active NameNode only writes edit logs to the local Shared Journal. Then both NameNodes in the secondary cluster will tail edit logs from the Shared Journal in the primary cluster.

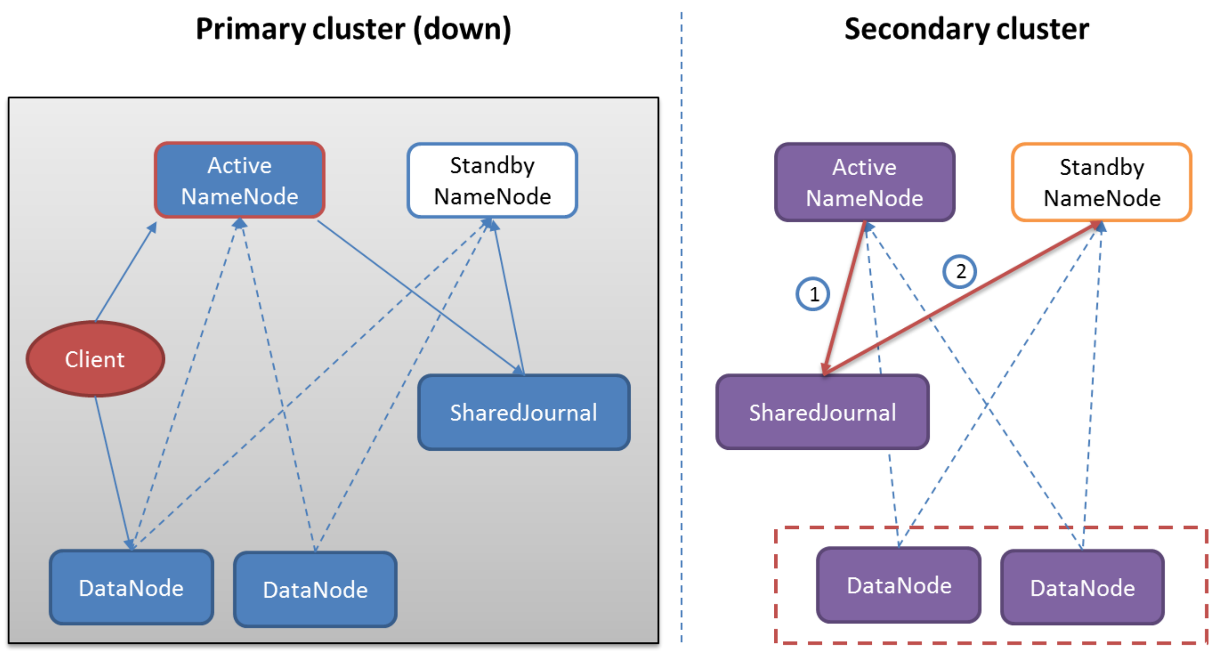
We still keep the Shared Journal in the secondary cluster. When the primary cluster is down and a disaster is declared, one of the secondary cluster NameNodes will take the Active NameNode role. As the process of role switching, the new Active NameNode will initialize to write to the local Shared Journal and other Standby NameNode will tail from the local Shared Journal.

The following diagram shows the basic flow for asynchronous namespace journaling.



1. The primary cluster Active NameNode writes the edit logs to Shared Journal in the primary cluster.
2. The primary cluster Standby NameNode tails the edit logs from Shared Journal of the primary cluster.
3. The secondary cluster Active NameNode tails the edit logs from Shared Journal of the primary cluster.
4. The second cluster Standby NameNode tails the edit logs from Shared Journal of the primary cluster.

The following diagram shows when the primary cluster is down and the primary role failovers to secondary cluster.



1. The secondary cluster Active NameNode writes the edit logs to the local Shared Journal.
2. The secondary cluster Standby NameNode tails the edit logs from the local Shared Journal.

Secondary Cluster Unavailable

For example, the network connectivity between primary and secondary clusters is lost or the secondary cluster is down. The impact of this would vary according to the configuration.

a. In this case, for the files or folders configured to use synchronous data writing and if this is the only secondary cluster, the application that writes to these files will fail. A possible improvement for this is to configure to allow switching to asynchronous mode when secondary cluster is not reachable.
b. For asynchronous data replication, when the connectivity is restored or the secondary cluster is up, the secondary cluster will start fetching the data from where it is left off and continue to synchronize itself with the primary cluster.
c. When admin configures synchronous namespace journaling, the primary cluster will not work since the writing to the secondary cluster Shared Journal will not succeed.
d. When admin configures asynchronous namespace journaling, when the connectivity is restored or the secondary cluster is up, the secondary cluster will start tailing the edit logs from where it is left off and continue to synchronize itself with the primary cluster.

For the case that the secondary cluster was down but cannot be restored, the cluster joining process is followed to make the secondary cluster to join as a fresh cluster. After the bootstrap process, all the existing data will then be replicated asynchronously.

Failover

In this version of document, failover will be manual. The admin has to decide when secondary cluster can become primary cluster.

For a synchronous namespace journaling setup, no namespace edit logs will be lost when disaster happened on the primary cluster. And for the files written by synchronous data writing method, the data will be also guaranteed in the secondary cluster.

The secondary cluster may come up with several situations.

1. If the secondary cluster comes up with all the data and namespace, this would be ideal. No data was lost and the secondary cluster is just the same as the primary cluster with exactly the same data.
2. For files configured for asynchronous replication, the secondary cluster may lags behind some portion of the data blocks. Then the secondary cluster may come up with all the namespace there, but some data blocks may be missing.
    a) If the number of these data blocks are not many, NameNode in secondary cluster will only come up with a few of files corrupted. But it doesn't prevent the HDFS cluster operating.
    b) If the number of these data blocks are too many, then NameNode in secondary cluster will come up in SafeMode because there are too many blocks are missing. This behavior is actually expected as the admin has to deal with the data integrity issue caused by the fact of the data is by far fully replicated. And it is risky to run applications under such state. The admin can have two options under such circumstances.
        i. If there is still a possibility that primary cluster will come up with data in the future and we need to secondary to be up and running right now, the admin can simply force the HDFS out of SafeMode and start to operate with possible many corrupted files.
        ii. If there is no way to recover the data by any means, the admin can do "fsck delete" to remove all the corrupted files and then the HDFS will get out of SafeMode and operate normally.

Data Block Replication

For data block replication in a single network, the existing facility in HDFS may be efficient enough. While for data replication across datacenters, the protocol and channel may be able to be optimized.

The basic idea is to improve the existing block sender and receiver facilities to be able to support pluggable block sender and receiver mechanism so that a potential faster block transport protocol or channel can be plugged in for different network environments.

Performance Considerations

For an ideal disaster recovery solution, the performance impact to primary cluster is minimized and the replication throughput to secondary cluster is maximized. But there is a trade-off between the impaction of primary and the best RPO of disaster recovery. We should allow those parameters which impact primary cluster performance being configurable so that administrator can make such trade-off.

For example, by choosing between synchronous data writing and asynchronous data replication, we can balance the performance impact and data safety. The best practice is using synchronous data writing for critical data only and minimizing the size of such data set.

For asynchronous data replication, by controlling the maximum concurrent replication tasks, we can control the workload added to the nodes in the primary cluster. Also, by implementing different schedule polices, the system can control flexibly the replication workload according to the existing workload on primary cluster. Good examples are scheduling the replication at night or scheduling with priority for different types of data.

Another example is that administrator can choose whether or not to compress or encrypt the data transferring between Primary and Secondary cluster. Data compression provides more efficient data replication through network but adds more work on the nodes running compression algorithms.