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 ........................................................................................................................................... 2
Targets .................................................................................................................................................. 2
Approach ............................................................................................................................................ 2
Design ................................................................................................................................................ 5
  Concept of Region ............................................................................................................................. 5
  Cluster Joining .................................................................................................................................. 5
  Synchronous Writing ......................................................................................................................... 5
    Synchronous Data Writing .............................................................................................................. 6
    Synchronous Namespace Journaling ............................................................................................... 7
  Handling Journaling Failures ........................................................................................................... 8
Asynchronous Replication .................................................................................................................. 8
  Asynchronous Data Replication ........................................................................................................ 9
  Asynchronous Namespace Journaling ............................................................................................... 10
Mirror Cluster Unavailable ................................................................................................................ 11
Failover ............................................................................................................................................... 11
Data Block Replication ..................................................................................................................... 13
Performance Considerations ............................................................................................................ 13
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 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 mirror cluster, we can have a single primary cluster and multiple mirror clusters across multiple datacenters. Each cluster will still have one Active NameNode and one Standby NameNode. The Active NameNode in each cluster will behave differently according to their cluster role.

2. There are DataNodes in both primary cluster and the mirror clusters. As normal, the DataNodes will only heartbeat and report blocks to the NameNodes of its local cluster. That’s to say, all the DataNodes of the primary cluster heartbeat and report blocks to the Active NameNode and Standby NameNode of primary cluster. And all the DataNodes of the mirror clusters heartbeat and report blocks to the Active NameNode and Standby NameNode of mirror cluster.

3. Writing data directly to mirror 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 mirror 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 mirror cluster DataNode in pipeline along with primary DataNodes. The mirror cluster DataNodes always be at the end of the pipeline. So, primary cluster should know about the available DataNodes in mirror cluster. Mirror cluster Active NameNode will heartbeat to the primary Active NameNode with a special command called MIRROR_DANODE_AVAILABLE (contains DatanodeInfo with space, load, etc.). The primary Active NameNode keep this details and will be used by the mirror placement policy while selecting node for pipeline. To satisfy real synchronous data replication, we make sure at least one DataNode selected from mirror 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.

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. Instead, the mirror cluster Active NameNode schedules the block replications from remote site. But the mirror cluster needs to know the block locations from primary to schedule replication. So, the mirror cluster Active NameNode will find blocks for replication and select set of local DataNodes to transfer block. Since mirror cluster not aware of block locations at primary cluster, it will just include batch of replication commands (MIRROR_REPLICATION_REQUEST which contains blocks with targets selected) to primary cluster Active NameNode 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.

6. When the system is configured to do synchronous namespace replication, the Active NameNode will be configured with a new shared Journal which writes the edit logs directly to the Active NameNode of the mirror cluster. When the mirror Active NameNode receives the flushed edit logs from primary cluster, it applies the operations to the in memory namespace and then write the edit logs to its local Shared Journal. In this way, the edit logs are guaranteed to be written successfully to the mirror cluster. This is to avoid edit transaction loss when primary cluster met unrecoverable crash in synchronous mode. The Standby NameNodes of both primary and mirror cluster will still tail edit logs from their local Shared Journal respectively.

7. When the system is configured to do asynchronous namespace replication, the Active NameNode of the primary cluster only writes edit logs to the local Shared Journal. The Active NameNode of mirror cluster will tail edit logs from the Shared Journal of primary cluster. The Active NameNode of mirror cluster applies the tailed edit logs and then writes to its local Shared Journal. The Standby NameNodes of both primary and mirror cluster will still tail edit logs from their local Shared Journal respectively.

8. The primary cluster Active NameNode is same with normal Active NameNode with some additional functionalities, such as handling MIRROR_DATANODE_AVAILABLE and block placement policy to support synchronous block writing and handling MIRROR_REPLICATION_REQUEST for replicates. The mirror cluster Active NameNode is similar with normal Active NameNode but with several significant differences and additional functionalities. The Active NameNode in mirror cluster will not allow any writes operations to the file system through client API. Instead, it will behave to receive or tail edit logs from primary cluster and update its namespace. It still writes edit logs to local Shared Journal. It also schedule active replications and heartbeats the commands (MIRROR_DATANODE_AVAILABLE, MIRROR_REPLICATION_REQUEST) to the Active NameNode of primary cluster. To simplify the things, current version targets to provide manual switch over to mirror clusters. So, the admin explicitly issues command to mirror cluster to convert the mirror 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 Active NameNode in primary cluster take different role with the Active NameNode in mirror 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 accordingly.

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 consistent across all setups.

Cluster Joining
When a new mirror cluster needs to join to an existing configuration, we use the similar bootstrap process as the current implementation of Standby NameNode joining.

First, the admin needs to initialize the Shared Journal of the mirror cluster by writing all the edit logs after the last checkpoint from primary cluster.

Secondly, the Active NameNode of the mirror cluster will be bootstrapped by downloading and applying the FSImage of the last checkpoint of primary cluster.

Finally, the Standby NameNode of the mirror cluster will be bootstrapped to downloading and applying the FSImage of the last checkpoint of mirror cluster.

After the joining process, the asynchronous data block replication can be scheduled by the Active NameNode of mirror 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 mirror cluster may enter into SafeMode as it finds that there are too many missing blocks. We need to make sure that even if the mirror cluster is in SafeMode, it doesn’t prevent mirroring functionality such as 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 of the primary cluster, the data will also write to the mirror 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 mirror cluster eventually even in the case of primary cluster disaster.
Writing data directly to mirror 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 mirror clusters.

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

![Synchronous Data Writing Diagram](image)

1. When a client is writing a HDFS file, after the file is created, it starts to request a new block. And the Active NameNode of primary cluster will allocate a new block and select a list of DataNodes for the client to write to. By using the new mirror block placement policy, the Active NameNode can guarantee one or more remote DataNodes from the mirror cluster are selected at the end of the pipeline.

2. The primary cluster Active NameNode knows the available DataNodes of the mirror cluster via heartbeats from mirror cluster’s Active NameNode with the MIRROR_DATANODEAVAILABLE command. So, latest reported DataNodes will be considered for the mirror cluster pipeline which will be appended to primary cluster pipeline.
3. As usual, 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. As usual, 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 mirror cluster replication. Based on the configured replication, mirror nodes will be selected.

**Synchronous Namespace Journaling**

It is sometimes critical that the namespace edit logs will not be lost when disaster happens. When the system is configured to do synchronous namespace replication, the Active NameNode will be configured with a new shared Journal which writes the edit logs directly to the Active NameNode of the mirror cluster. When the mirror Active NameNode receives the flushed edit logs from primary cluster, it applies the operations to the in memory namespace and then write the edit logs to its local Shared Journal. In this way, the edit logs are guaranteed to be written successfully to the mirror cluster. This is to avoid edit transaction loss when primary cluster met unrecoverable crash in synchronous mode. The Standby NameNodes of both primary and mirror cluster will still tail edit logs from their local Shared Journal respectively.

The following shows the basic journaling workflow in which a plugin Journal Manager is used to write edit logs from the primary Active NameNode to the mirror Active NameNode.
1. As usual, the primary cluster Active NameNode writes the edit logs to Shared Journal of the primary cluster.

2. The primary cluster Active NameNode also writes the edit logs to the mirror cluster Active NameNode by using a new JournalManager.

3. As usual, the primary cluster Standby NameNode tails the edit logs from Shared Journal of the primary cluster.

4. The mirror cluster Active NameNode writes the edit logs to Shared Journal of the mirror cluster after applying the edit logs received from the primary cluster.

5. As usual, the mirror cluster Standby NameNode tails the edit logs from Shared Journal of the mirror cluster.

Handling Journaling Failures

When doing synchronous namespace journaling, the primary cluster Active NameNode will write both to the local Shared Journal and to the mirror Active NameNode (mirror Journal). We need to deal with the complex situation when the sync to the one Journal succeed but the sync to the other Journal failed, thus leave the two Journals in inconsistent state. In such a situation, gap will happen in the system of 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 the mirror Journal standing for the mirror Active NameNode.

2. When a required Shared Journal failed, the primary cluster Active NameNode will shut down.

3. When one NameNode of the primary cluster restart or pick up the Active NameNode role, and before doing any log operations, it check with the mirror Journal for the last sync txid.

4. If the last sync txid of mirror Journal matches with the last sync id of primary Journal, it continues as normal.

5. If the last sync txid of mirror Journal doesn’t match with the last sync id of primary Journal, it will sync the mirror 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 mirror cluster, there is an alternative approach.

1. The primary cluster Active NameNode will write first to the mirror Journal. The reason that we writes to the mirror Journal first is we want the mirror Journal will always with succeeded edit logs.

2. If mirror Journal succeeded, but local shared Journal or local file Journal failed, we shut down the 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 mirror anyway will have as we ensured to write first to the mirror.

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 mirror cluster is selected for the pipeline at Active NameNode.
2. As usual, 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. As usual, 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 mirror cluster.
4. Asynchronously, the mirror 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 MIRROR_REPLICATION_REQUEST which will contain batch of blocks to replicate with target DataNodes selected from mirror cluster. The mirror cluster doesn’t need to aware of real block location in primary cluster.
5. As a result of handling the MIRROR_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 of the mirror 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 system is configured to do asynchronous namespace replication, the Active NameNode of the primary cluster only writes edit logs to the local Shared Journal. The Active NameNode of the mirror cluster will tail edit logs from the Shared Journal of primary cluster. The mirror cluster Active NameNode applies the tailed edit logs and then writes to its local Shared Journal. The Standby NameNodes of both primary and mirror cluster will still tail edit logs from their local Shared Journal respectively.

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

**Asynchronous Namespace Journaling**

1. As usual, the primary cluster Active NameNode writes the edit logs to Shared Journal of the primary cluster.
2. As usual, the primary cluster Standby NameNode tails the edit logs from Shared Journal of the primary cluster.
3. The mirror cluster Active NameNode tails the edit logs from Shared Journal of the primary cluster. And applies the edit logs to its namespace in memory.

4. After applying the edit logs to its namespace, the mirror cluster Active NameNode also writes the edit logs to its local Shared Journal.

5. As usual, the mirror cluster Standby NameNode tails the edit logs from Shared Journal of the mirror cluster.

**Mirror Cluster Unavailable**

For example, the network connectivity between primary and mirror clusters is lost or the mirror 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 mirror 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 mirror cluster is not reachable.

b. For asynchronous data replication, when the connectivity is restored or the mirror cluster is up, the mirror 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 mirror Journal will not succeed.

d. When admin configures asynchronous namespace journaling, when the connectivity is restored or the mirror cluster is up, the mirror 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 mirror cluster was down but cannot be restored, the cluster joining process is followed to make the mirror 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 mirror cluster can become primary cluster.

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

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 mirror cluster.

The mirror cluster may come up with several situations.

1. If the mirror cluster comes up with all the data and namespace, this would be ideal. No data was lost and the mirror cluster is just the same as the primary cluster with exactly the same data.
2. For files configured for asynchronous replication, the mirror cluster may lags behind some portion of the data blocks. Then the mirror 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 the mirror 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 the mirror 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 mirror 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 mirror 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 the 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 of the primary cluster. Also, by implementing different schedule polices, the system can control flexibly the replication workload according to the existing workload on the 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 the primary and mirror cluster. Data compression provides more efficient data replication through network but adds more work on the nodes running compression algorithms.