Implementing Active/Active Multi-Cluster Deployments with Cloudera Enterprise

Revision History			
Version	**Author**	**Description**	**Date**
0.1	Ted Malaska	Initial Version	3/25/2015
0.1	Ted Malaska	Updated per Alex's Comments	5/14/2015

1 Executive Summary

One of the key benefits of an enterprise data hub (EDH), built on Apache Hadoop, is its ability to scale. A single cluster can scale to thousands of nodes. Additionally, as necessary with any production system, this cluster includes multiple levels of replication and failure safes to ensure data is protected and the system is fault-tolerant. However, many organizations still have a need for multiple clusters, with three of the most prominent reasons being site failure, geographic locality, and/or separate use case clusters.

Site Failure

Site failure is when data access or application uptime needs to be more reliable than the uptime of a single data center in the case of full cluster failure or down-time. This will apply to any system that needs 100% uptime. Sites can go down for scheduled to unscheduled reasons, and to get 99%+ uptime we are going to need a fall back site.

Geographic Locality

These are separate clusters that are designed to meet the real-time needs of a local geographical space (such as US East, US West, Europe, and Asia) but also must share the same data so that request responses in Asia will return the same or as similar as possible results as a request made in the US East. Common examples of this need are payment monitoring systems, gaming platforms, and global financial systems.

Separate Use Case Clusters

Normally, when considering separating use cases by cluster, there is a set of use cases with strong SLAs and a set with weak SLAs. This means one cluster could be tuned and configured to meet the demands of the strong SLA workloads, while allowing weaker SLA workloads to run on the same data on a separate cluster. A common example would be running production jobs on cluster A and running research and sandboxes on cluster B. This setup provides more freedom to the ad hoc jobs, without impacting the production jobs. In addition, if cluster A goes down, then production jobs could migrate to cluster B while temporarily disallowing the weaker SLA jobs to run.

"Active/Active" is a way to address these multi-cluster needs. "Active/Active" can mean different things for different use cases (as evidenced by the archetypes defined later in this paper) but at a high level, when this paper refers to "Active/Active" it refers to the following definition:

- Consists of more than one cluster
- Maintains full desired functionality by migrating to the remaining cluster in case of cluster failure
- Avoids data loss of selected data in the case of single cluster failure
- Able to handle workloads on either cluster

- Supports all of the Apache Hadoop ecosystem, not just HDFS
- Supports selectivity of what is replicated
- Supports administrative controls to prioritize replication and throughput

Cloudera's enterprise data hub, Cloudera Enterprise, lets you easily implement Active/Active multi-cluster deployments using the integrated, automated Backup and Disaster Recovery capability, available with Cloudera Manager.

2 Business Objectives for Active/Active

The objective here is to realize the full power of Hadoop and an enterprise data hub (EDH), while not falling prey to single-site vulnerability and allowing for complete site-separation of workloads. This section will list out the business objectives of why building multi-site into your architecture is worthwhile.

Security from data loss at a site level

While an EDH offers a replication factor of three by default to protect against node and drive failure, many industries require additional protection in the event of site failure, such as from natural disasters, human errors, or even hostile attacks. One example is the Sarbanes-Oxley rule around multi-region datacenter location - and one that most financial companies require to protect their data. This radio gap is put in place to protect corporate data by separating it by a minimum distance to insure the data will survive a wide area disaster.

Cloudera Enterprise allows for site failover (as described through the archetypes in this paper), so enterprises can rest-assured that their critical data can be stored and protected in multiple physical locations.

Quick failover for high SLA workloads:

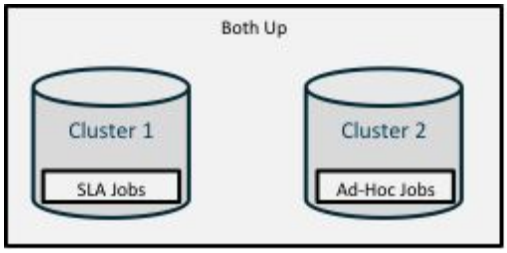
One of the fundamental principles of Hadoop is the ability to move processing and analytics to the data. If the required data is available in any one of the joined clusters, then the job (full-text query, SQL statement, multi-MB jar file, etc) can be sent to any active cluster storing the data to be executed. This is common for clusters where you will have unplanned user activity on the cluster and you cannot have a single site's downtime to affect the work in progress of people working on the data. This use case could include running near real-time (NRT) HBase jobs or even hundreds of users on Impala just running SQL.

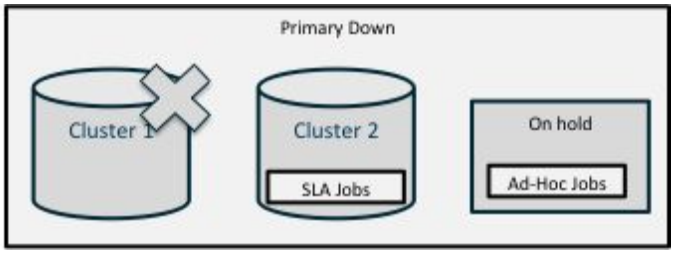
Complete isolation of workloads:

Cloudera Enterprise already has many features built in to protect workloads from other workloads within a single cluster, including controls for CPU, memory, and disk usage. However, even with these controls, there can still be unintended contention if not configured correctly. With cluster-level isolation, not only is there 100% protection from

contention, but also the ability to use all of the resources of the secondary, failover cluster for workloads such as research and exploration. By allowing both clusters to have different resource configurations, you can optimize them for different use cases.

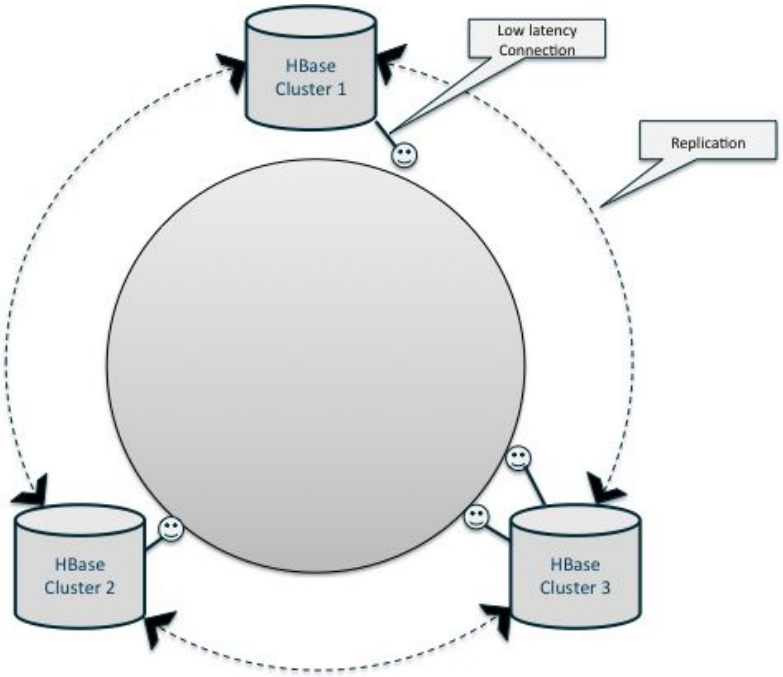
A great example of this is a two-cluster solution where the main cluster has hard SLAs and the backup cluster is used for ad-hoc research. If the primary cluster goes down, you can fail over the SLA jobs to the backup cluster and the ad-hoc jobs are suspended until both clusters are back up again. In this design, the SLA jobs are never at risk from single-site failure or any ad-hoc activity on the cluster.





Global State:

A 360-degree view of customers is one of the common use cases enterprises are building on with Cloudera Enterprise. When these users are traveling around the globe, they expect the same personalized information to follow them, no matter where they are, while maintaining the low latencies from the nearest cluster. By leveraging Active/Active data, enterprises can keep data in sync across all their global clusters – only delayed by the speed of light across the networks between these clusters.



An example of this business case are systems that need to monitor the activity of users anywhere on the globe but need them to directly interact with the cluster nearest to them to avoid unnecessary latency, such as required for credit card fraud.

3 Common Archetypes for Multi-Cluster Active/Active

There are a number of architectures for replication in an EDH. These architecture all have different pros and cons and some may be a better fit depending on the use case. There are a number of considerations for these architectures and the list below includes some of the highest-level design considerations.

- **Size of the data:** With an EDH, a modest cluster can saturate a 10GB line for 24 hours with only 5 minutes of data generation.
- **Compression rate:** While compression can mean getting much more data over a fixed pipe between clusters, it comes with trade offs.
- **Latency requirements:** Consider what the SLA is for the given dataset: seconds, minutes, or hours.
- **Underlying storage system:** The replication requirements and solutions vary significantly from HDFS (large batch), Apache Hive Tables (includes metadata), and Apache HBase (near real-time and event-based).

Below is a look at the high-level archetypes that span the majority of all replication requirements for an EDH. However, there may be some use cases that require modifications or customizations to these archetypes. Cloudera Enterprise allows organizations to push the boundaries of what is possible with data management and processing, and its flexibility and production-ready capabilities make it adaptable as new requirements emerge.

Batch Table/File Replication

Batch movement of data is the easiest form of replication when working with HDFS and/or Hive Tables. With Backup and Disaster Recovery (BDR), included with Cloudera Enterprise, you can easily define the following for a dataset (folder), database, or selection of tables:

- **What triggers replication:** Should replication happen on scheduled intervals, on-demand, or off Cloudera Navigator policies as part of the processing workflow?
- **What is the replication frequency:** This is important to prioritization and SLA management.
- **How much bandwidth to allow for:** This is also important for the prioritization of different datasets and for overall management and control over the throughput of the pipe between the clusters.

- **Do you allow for deletion when replicating:** Turning delete replication off or using the trash delete replication protects against human error.
- **Can you select from and to cluster:** To avoid confusion, replication is defined as one way.
- **What and how to be notified:** There are a number of notification options included with BDR. This allows you to keep track of what data is where so, in the case of failure, you have a complete picture of what data is replicated.

BDR is also built upon Distcp and, therefore, the following functionality is included with the Cloudera Enterprise architecture:

- Parallel copying of data
- Byte-by-byte data copies so compressed data doesn't need to be uncompressed to be transferred
- All permissions replicate with the data
- During the copying process, the file is stored at a TMP location so not to disrupt the workloads working in that table or folder with partly completed data files.

Dual Ingestion

Depending on an organization's requirements, they may require more real-time processing than what batch offers. For these use cases, a simple solution of Apache Kafka and Apache Flume is used to support dual ingestion into both clusters.

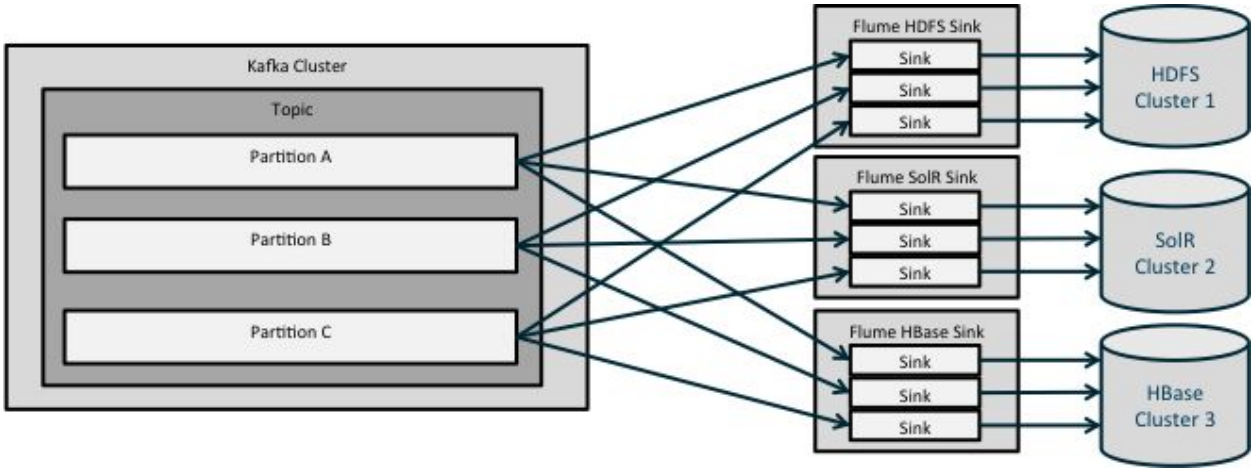
In this design, Kafka is used as the channel or pipe for the data being sent to the cluster and Flume is used to persist the data to HDFS, HBase, and/or Solr.

Kafka is ideal for this use case due to its:

- Horizontal scalability
- Internal replication to avoid data loss
- Cross-site replication to avoid data loss (Kafka's MirrorMaker)
- Compression to reduce over-the-wire bandwidth usage
- Ability to store a configurable amount of days worth of data in case of process error, long outages, or any other reason
- Consumer Groups for the redistribution of data to remaining consumers in case a single consumer fails.

Kafka is just a queue or pipe - it doesn't do any mutations or persisting of the data to final locations. That's where Flume comes in. Flume is ideal for the ingestion portion as it is tuned to HDFS, HBase, and Solr, and has been production-hardened over the years. Working together, the Flume Node can pull data from Kafka as a consumer. Due to Kafka Consumer Groups, if a Flume Node is lost, the remaining Flume Nodes will pick up the

slack. Flume also has a concept of an interceptor and a serializer that allows for mutations before it lands in its final destination.

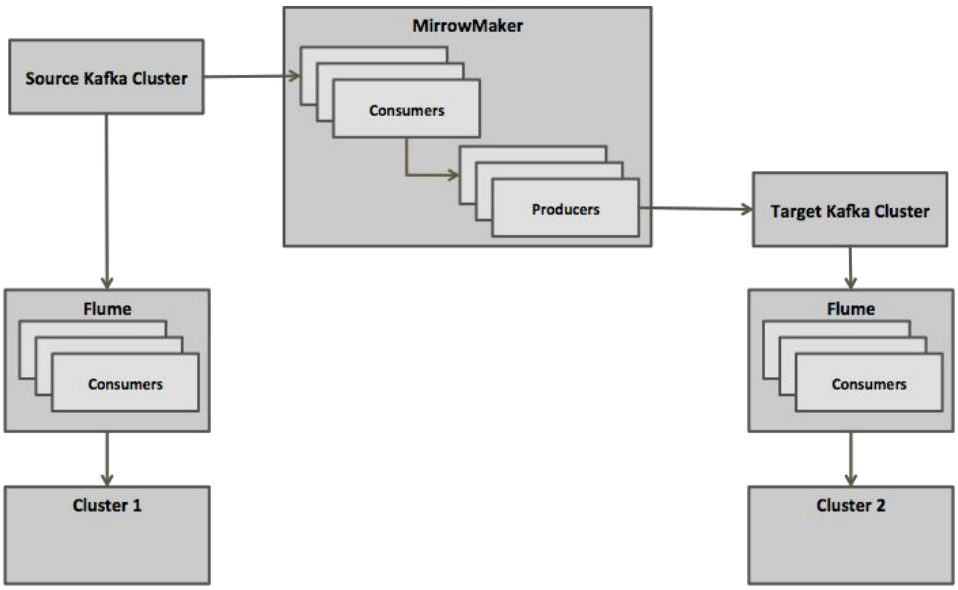


Dual ingestion has some other different attributes to keep in mind when evaluating BDR versus dual ingestion:

- Dual ingestion has a lower latency for a single event to reach both clusters.
- Dual ingestion has lower compression rates than BDR. BDR may have higher throughput depending on the compression codecs and file types used. For example, when comparing Parquet and GZip to Kafka/Flume Snappy over-the-wire, there can be a 2-5x difference in bandwidth required.
- BDR provides notifications that will let you know which files reached which clusters. With dual ingestion, notifications can be a bit trickier. There are solutions such as appending a timestamp to each event, but these require additional architecture decision.

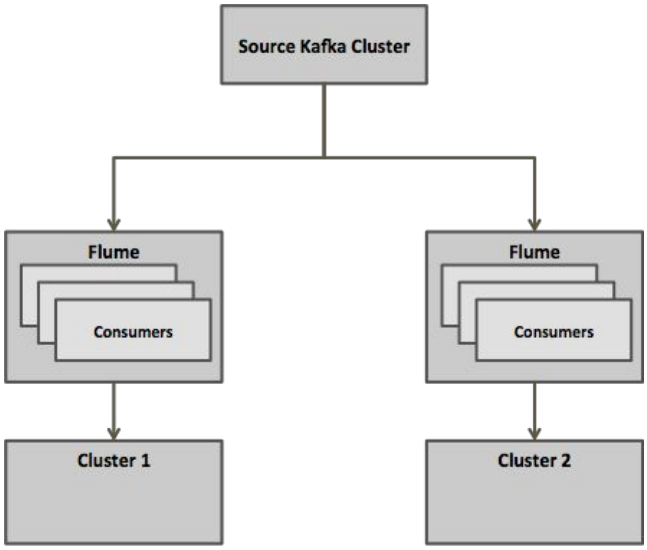
MirrorMaker vs just Dual Consumer Paths

When deciding how to do replication with Kafka you need to decide if you are replicating Kafka or just dual ingesting. The designs are a little different, in that if you are replicating Kafka, you will likely use MirrorMaker, which will give you an architecture diagram similar to the below.



This leads to eventual replication to a second Kafka cluster. In case of loss of the first Kafka cluster, you have a second one to fall back on.

If you are only interested in the dual ingestion of the data then MirrorMaker might not be what you're interested in, and that design would look like the following.

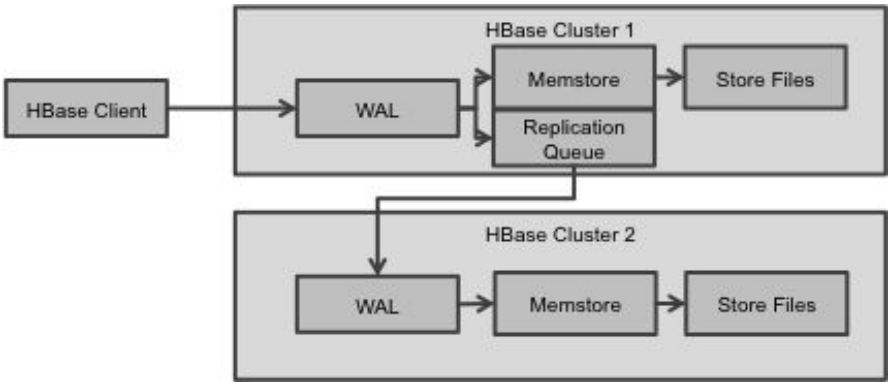


Apache HBase Replication

HBase is the NoSQL storage standard for Hadoop and includes a variety of multi-cluster replication options, including master-slave, master-master, or more than two-cluster replication.

At the core, the design is pretty easy to understand. As data is written to HBase, it is persisted to a Write Ahead Log (WAL) that is replicated three times in HDFS. Normally, without replication, there is a process that reads from the WAL, adds that data to the Memstore to be sorted and ordered, and then writes it again to disk as a sorted indexed store file - a process called Store File Flushing.

When replication is set, simply add the Relocation Queue to this design. This also reads off the WAL and sends the newly added data to the secondary cluster. If the secondary cluster is down, the WAL remains on HDFS until the other cluster comes back up and all the data has been sent over-the-wire.



This design is great for updating remote clusters as fast as possible, but there are some tradeoffs. One such tradeoff is all actions written to the WAL will have to be replicated over-the-wire. If you have more changes per second than can fit over-the-wire, the Replication Queue will fall behind. It will catch up as the actions per second start to slow, but, if it never slows down, the Replication Queue will always be behind.

When using master-master, note that data will be replicated between both clusters and conflicts will be resolved with time stamps, where the last write wins. Also consider that conflicts in HBase are not row conflicts - they can only be column/value conflicts. If there is a requirement to never rely on timestamp conflict resolution, there are schema designs that allow for cluster-specific column value definition and population.

Advanced Designs

While the above archetypes cover the most common use cases, they won't work for every solution. Perhaps you require lower latency than what's available with BDR, but you still need to replicate files. Or perhaps you need to increase compression over-the-wire to better take advantage of the pipe between clusters.

One of the benefits of Cloudera Enterprise is its flexibility to adapt to advanced use cases and designs. With various, production-ready execution engines and frameworks, Cloudera Enterprise is resilient to future replication use cases.

For the low latency/high compression example, you could solve this with a long running Spark application that micro-batches newly dropped files together while compressing them in GZip and Parquet before sending them to the failover cluster. This would avoid the startup and shutdown cost of BDR, as well as increase the compression to allow for more bandwidth over-the-wire.

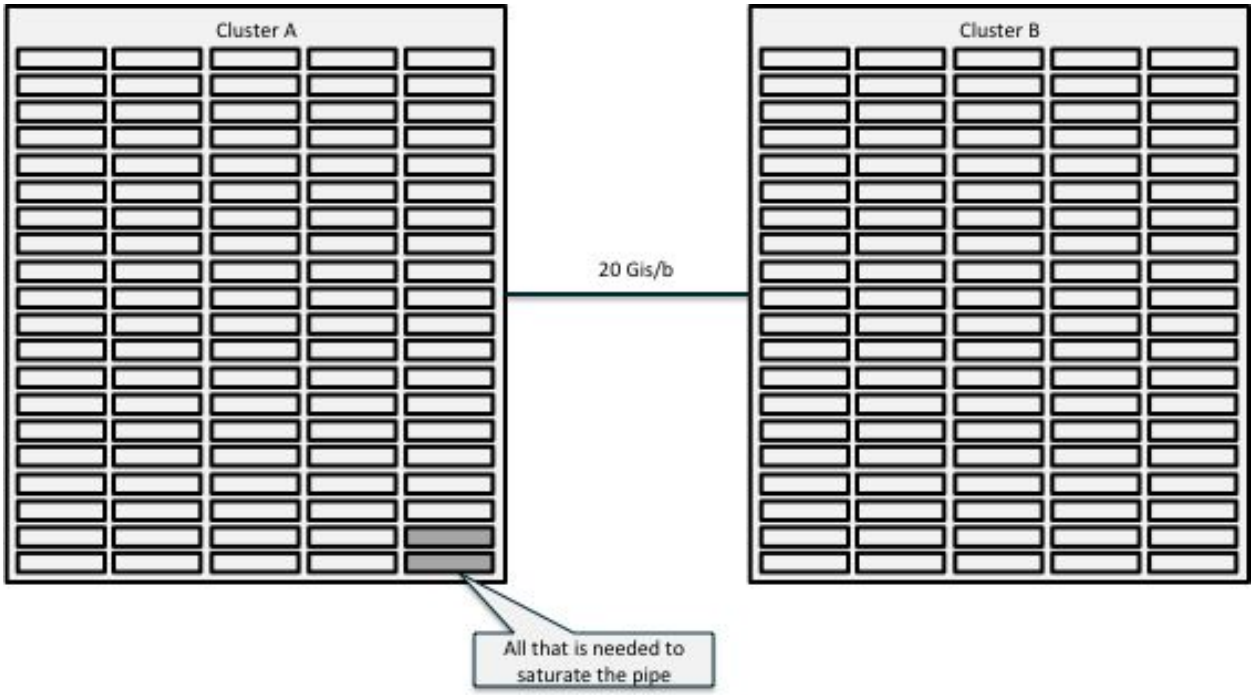
This is just one type of solution for one type of use case. Cloudera will partner with you to help design the best solution for your use case.

4 Architecture Considerations

Pipe Size

One of the most common issues facing companies when implementing a multi-cluster EDH is the size of the pipe between the clusters. No matter how large the pipe (10GB/s, 40GB/s, 80GB/s, or more), it's always possible to saturate the pipe. One example is a 100-node cluster can saturate a 10 GB/s pipe between clusters for 24 hours, if it generated data at full blast for only 10 minutes.

Lets take the image below as an example. Let's say you have two 100-node clusters and you have a 20 GB/s pipe between the clusters and each node has 10 GB/s to and from the switch. Just two nodes could saturate your whole pipe in this scenario.



So what does this mean for the organization?

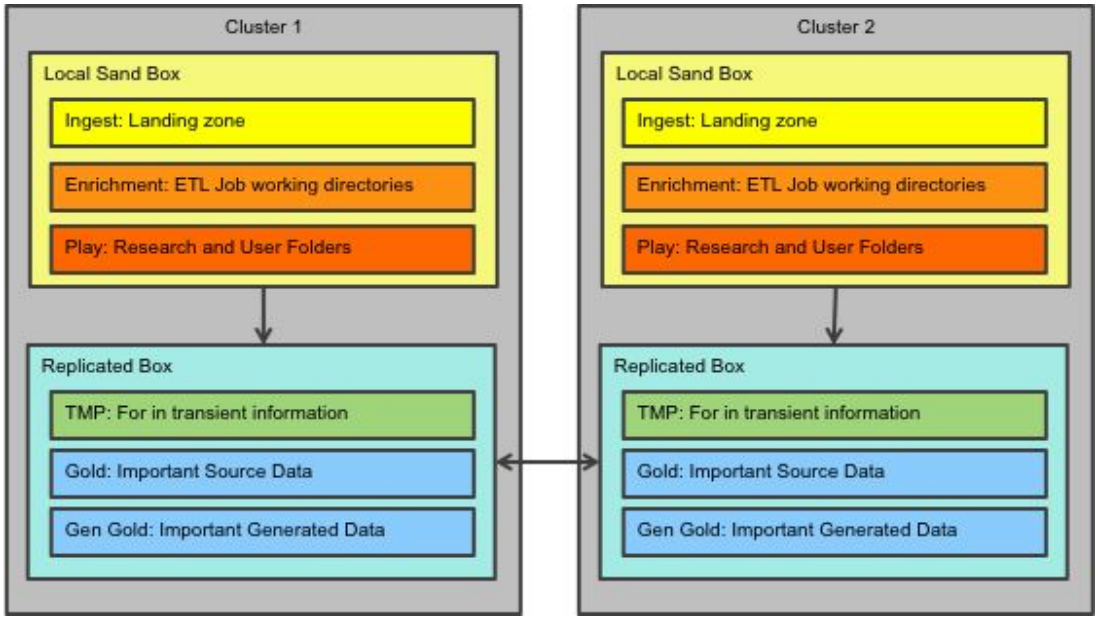
- **Data needs to be prioritized:** This will be explained in more detail in the next section, but, in short, only select data should be replicated
- **Administration should lead replication policy:** Since the pipe is a shared resource, there needs to be an administrator who monitors the use of the pipe and controls what goes over it. The simplest way to manage your policy is through BDR and Cloudera Manager.
- **Compress Compress Compress:** Each of the archetypes have different compression rates, from Snappy compression of data-in-motion to Parquet compressed with GZip. Compression will increase the amount of information that will be able to pass through the pipe by 2-30x depending on how your data compresses.
- **SLA & Latency:** We will go into it in more detail later in this paper, but because of a restricted pipe, only so much can go over the wire and, with that, there is a tradeoff between latency and possible compression rates.

Data Classification

As described above, an EDH can store and generate huge amounts of data. This means we need to be selective and smart when deciding what goes over-the-wire.

For some types of data, it's relatively easy to decide whether or not it will go over-the-wire. Data being used and generated in sandbox environments or for ad hoc research usually isn't worth replicating. It's important that EDH users can do a variety of workloads without the fear of overflowing the pipe between clusters, and we don't want our administrator to ask them to stop experimenting. In addition, initial data loaded into the cluster or

temporary files within a multi-step ETL process may not be selected for replication. It depends on your use cases, requirements, and SLAs for the different data sets.



Once you have selected your data to replicate, be sure to try and compress it. Compression is good for multiple reasons, including: lowering the cost per TB, reducing the resources required to read a data source, and allowing more information to pass through a pipe of fixed bandwidth.

Tradeoff of Latency vs Compression

Compression is critical for replication but, as described, low latency per event and higher compression are often in conflict. When defining the replication strategy for a given dataset, these two options must be weighed carefully.

Compression rate is really a result of which Codec you use and the entropy of your data. When you are streaming data, there is less time to organize your data in a way to lower the entropy, however, in a batch process we can achieve much higher compression - which points to the conflict, low compression and low SLAs versus high compression and larger SLAs.

An example in batch processing is to use the Parquet file format that will organize the data in a column structure, which will lower the overall entropy of the data and lead to much higher compression rates.

In streaming, a Codec like Snappy or LZO is normally used for speed, and those Codecs will offer less compression than a Codec like GZip that is much more CPU intensive on write.

In summary, with streaming compression, we should be somewhere in the range of about 2x compression, while in batch modes, we will most likely see compression rates of about 8-15x.

All compression rates will depend on the data and its natural entropy but the ratios should hold pretty close to the values in this document.

5 Conclusions

Multi site cluster management can be very beneficial for a number of use cases and is possible with Cloudera Enterprise, including the flexibility to adapt to future and advanced use cases. To learn more about Cloudera Enterprise, visit www.cloudera.com. For a free, 60-day trial, visit www.cloudera.com/downloads.