DELL EMC

ACTIVE/ACTIVE HADOOP AND HIVE ON ECS

TECHNICAL SOLUTION GUIDE

Geo-Replicated Hadoop on ECS

ABSTRACT

This technical solution guide provides a solution for running Hadoop and Hive across multiple active data centers. It is based on Dell EMC Elastic Cloud Storage and Hortonworks Data Platform.

January, 2017

TABLE OF CONTENTS

INTRODUCTION

Hadoop is an open source programming framework that supports the processing and storage of extremely large data sets in a distributed computing environment. It is part of the Apache project sponsored by the Apache Software Foundation. Hadoop is designed to span a single cluster that stays within a single data center. This concept works very well when there is only a single data center to be concerned about. However, when enterprises start thinking about how to expand their system to multiple data centers around the world, many difficulties are encountered that need to be solved.

Hive is a key component of any Hadoop infrastructure. First, it provides the metadata that can organize countless directories and files into tables and columns that can be queried using standard SQL. Second, it also provides a SQL engine that can execute a SQL query by converting it into a series of MapReduce or Tez jobs and then execute the jobs. Additionally, other systems such as Spark and HBase use the metadata services provided by Hive to organize files into tables but do their own query processing.

The solution presented in this guide is based on Dell EMC Elastic Cloud Storage (ECS™) which provides a global file system and the Hortonworks Data Platform™ enterprise-ready open source Hadoop distribution. This solution provides a strongly consistent global file system across two to eight sites. All sites are *active* meaning that data can be loaded, read and write SQL queries can be executed, tables can be created and deleted, and ACID transactions can occur, concurrently and consistently across all sites. Further, because ECS can cache often-used files at each site locally (while maintaining consistency, of course), the performance impact is minimized.

SOLUTION PURPOSE

The solution described in this guide provides the following capabilities:

- Supports 2 to 8 sites
- Each site can be in a different data center, city, country or continent
- A common Hive Metastore, Hive Warehouse and Hadoop Compatible File System readable and writable across all sites
- Hive concurrency and ACID transactions (insert, update, delete), even across sites
- Strong consistency
- Asynchronous replication
- Fully recoverable from the failure of a single site
- Supports Hive, Tez, MapReduce, YARN, Spark, Sqoop, Pig, Flume

ALTERNATIVE APPROACHES TO MULTI-SITE HADOOP

There are several alternative approaches to using Hadoop across multiple sites that are often considered.

DISTCP

DistCp (distributed copy) is a MapReduce job provided in standard Hadoop distributions that is designed to copy files in parallel within or between Hadoop clusters. As such, it can be also be used to copy files between Hadoop clusters in different data centers. However, it is not an ideal solution due to the following drawbacks.

- Active/Passive access (not Active/Active)
    - Although the target cluster can be used at any time, access to files and directories that are actively being written can result in inconsistent data, incomplete data, or even the appearance of corruption. Avoiding this inconsistency issue is difficult if not impossible and so the target cluster is often idle except in the case of failure of the primary cluster.
- Very high disk usage (3x replication on each site)
    - Since each site has an independent set of files, each site needs a complete copy of the files. But a single copy is not sufficient because there is no automatic method to retrieve blocks or files from remote sites when a disk failure

occurs. So this means each site still needs a minimum of 2x replication and 3x replication is often used for critical data.

- Periodic transfers (once every few hours)
    - DistCp must check an entire directory structure for changed and new files so running it more often can be very expensive.
- Difficult to build for three or more clusters
    - A DistCp job must be carefully defined for each source directory and target directory combination. If there are three sites with five base directories each, that means creating and managing 30 jobs (3 source sites * 5 source directories * 2 target sites) across three clusters to ensure that the directories are available at all sites.
- As a MapReduce job, it consumes YARN CPU and memory
- No consistency guarantees beyond snapshot and replication
- No Hive concurrency controls on target cluster
- Requires carefully updating the Hive Metastore manually at each site
    - Any new or updated table or partition must be defined consistently at each site.
    - Although there are 3rd party tools that can ease some of the pain with keeping the Hive Metastore in sync, these come with their own sets of limitations and potential issues.

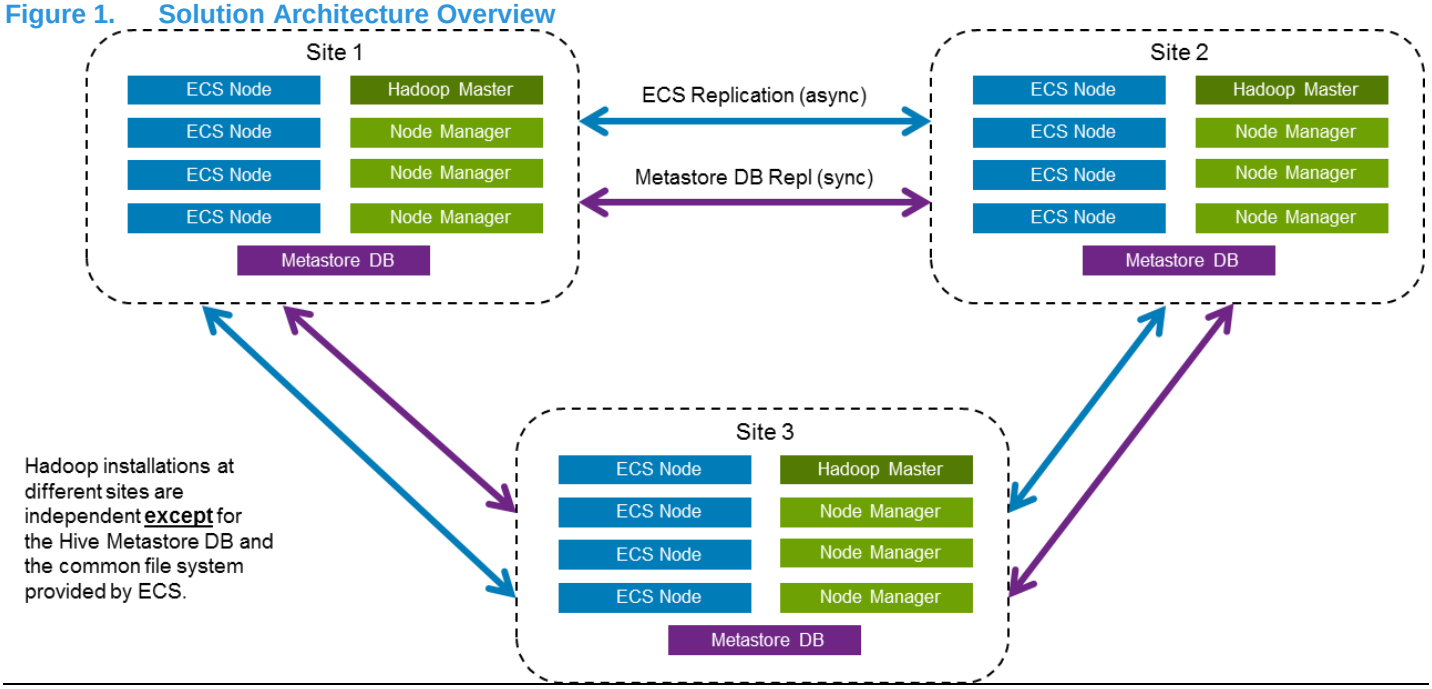
EXTENDING A SINGLE HADOOP CLUSTER ACROSS THE WAN

One approach that might be considered is to simply extend a single Hadoop cluster to other data centers in the same way that a Hadoop cluster can be extended within a data center. This is straight forward to do but will hardly ever work in practice because Hadoop components assume that the communication between all nodes is both low latency and high bandwidth. For instance, the shuffle phase of a typical MapReduce or Spark job will often send huge amounts of intermediate data between all nodes without concern for which nodes are closer. Additionally, ZooKeeper, which is used to coordinate the distributed state of many other applications such as YARN and HBase, requires low latency for acceptable performance.

SOLUTION ARCHITECTURE

SOLUTION ARCHITECTURE OVERVIEW

Figure 1. Solution Architecture Overview



The figure above shows the architecture of the solution. Each site has four or more ECS nodes that provide the common file storage. The ECS nodes at a single site are grouped into a Virtual Data Center (VDC). The VDCs are grouped together into an ECS federation, providing geo-replication and geo-caching. Replication in an ECS federation is asynchronous which provides low latency writes even with a world-wide federation. However, ECS is still able to provide strong consistency when all VDCs are online.

A clustered relational database named Galera Cluster™ for MySQL is installed on several nodes across the sites. This provides the Hive Metastore Database which contains table and column metadata as well as locks for concurrency control. Galera Cluster for MySQL provides synchronous replication and thus exhibits latency that is higher than a local-only MySQL installation but provides strong consistency and maximum availability for this critical component.

At each site, there is a Hortonworks Data Platform (HDP) Hadoop cluster. These Hadoop clusters are independent except for the shared Hive Metastore Database.

Data which is to be shared and protected across all sites (e.g. important Hive tables) will be stored on ECS in a bucket that is configured to be replicated. Data which does not need to be shared and protected can stay local at each site in a non-replicated ECS bucket.

SOLUTION COMPONENTS

DELL EMC ELASTIC CLOUD STORAGE (ECS)

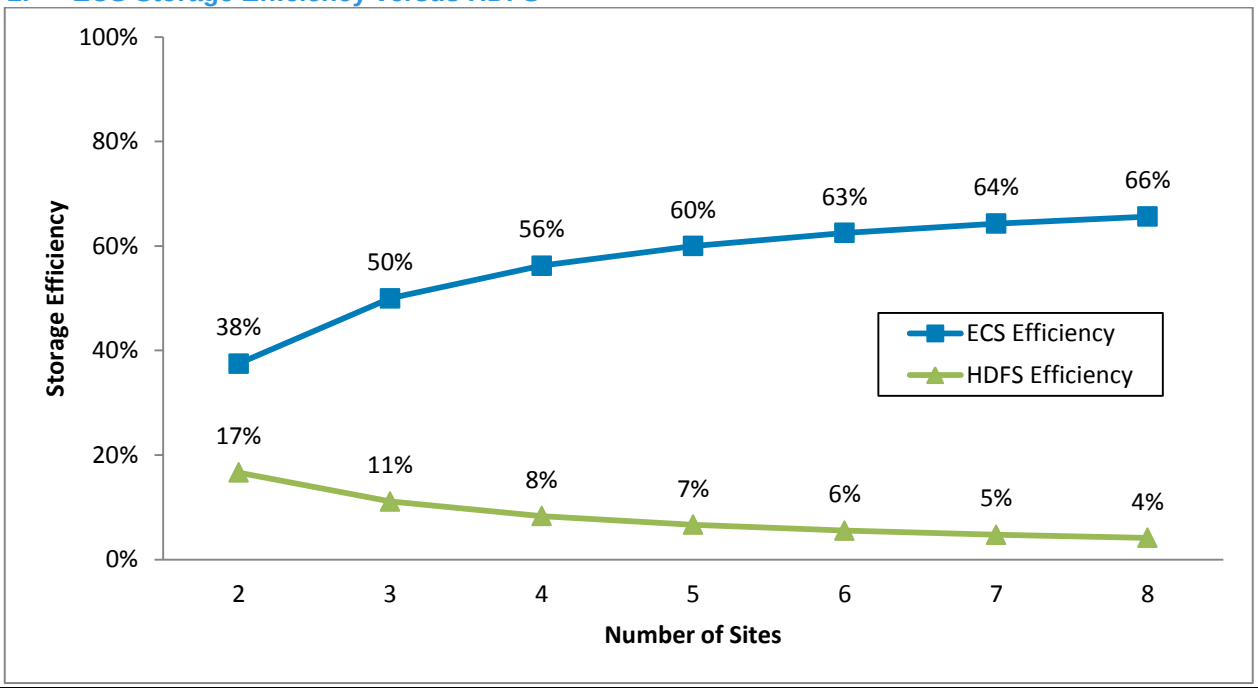
The third generation object platform from Dell EMC, Elastic Cloud Storage (ECS) is designed for traditional and next-generation applications with unmatched storage efficiency, resiliency and simplicity. ECS has been true software-defined storage that is available as a turnkey appliance, as ECS Software that could be deployed on supported industry-standard hardware, or as a fully EMC-managed ECS-as-a-Service. ECS supports standard object protocols such as S3 as well as file system protocols such as NFS and it provides a Hadoop Compatible File System (HCFS) interface.

ECS provides the following key features relevant to this solution.

- ECS supports 2 to 8 sites in a federation. Each site can be in a different data center anywhere in the world.
- Strong consistency is guaranteed when all sites are online. This is achieved even when an object has not been completely replicated by forwarding read/write requests to the site that owns the object.

- Replication occurs asynchronously which provides low latency updates even with a world-wide federation.
- RPO (Recovery Point Objective) can be less than one minute. This means that writes and other updates are replicated to other sites usually in less than one minute. Of course, large updates will take longer and the time will depend on the size of the updates, the WAN speed and several other factors.
- Replication is configured per bucket. A bucket on ECS is the root of a particular file system. An ECS cluster can have multiple buckets and each bucket can be configured with different replication properties. For instance, one bucket can be configured to replicate between three specific sites while another bucket can be configured to only exist at a single site.
- Data Protection
    - Data is protected *across* sites (VDCs) using either XOR erasure coding or mirrors. When a bucket is replicated across three sites, for example, 1/3rd of the stored bytes are XOR parity. With four sites, it becomes 1/4th, and so on. All data is fully recoverable from the complete failure of any single site.
    - Data is protected *within* a site (VDC) using Reed Solomon erasure coding. For each 12 blocks of data, 4 additional blocks of parity are calculated and stored. Each of these 16 blocks is distributed on different disks within the site. All data is fully recoverable from the failure of any four drives or a single node within the site.
    - All data is fully recoverable from the complete failure of a single site plus four drives in each site.
    - All data is fully recoverable from the complete failure of a single site plus one node in each site.
    - The two layers of data protection (across sites and within a site) work together to provide an efficiency starting at 38% for two sites and increasing up to 66% for eight sites. Storage efficiency is the effective percentage of raw disk bytes that are usable by your data.
    - In contrast, using a traditional HDFS system with the standard 3x HDFS replication within a site and full mirrors across sites provides an efficiency of 17% for two sites and only 4% for eight sites.

Figure 2. ECS Storage Efficiency versus HDFS



ECS efficiency is 4.5 times higher than HDFS with 3 sites and improves with more sites. HDFS efficiency assumes 3x replication within sites and mirroring across sites.

- Frequently used data is automatically cached at sites (geo-caching), avoiding WAN delays and reducing WAN traffic. Of course, strong consistency is still enforced and any out-of-date cached copies are not used after the original file has been updated.
- ECS supports a variety of storage protocols, including S3, CAS, and NFS. For access from Hadoop, a Hadoop Compatible File System (HCFS) client-side Java library is provided.
- Hadoop URIs for files in ECS are the same regardless of the requesting (client) site or owning site. This allows ECS buckets to be located or relocated anywhere within the ECS federation without the URI changing. It also allows a Hive table to reference a single consistent location for its data files.

GALERA CLUSTER FOR MYSQL (HIVE METASTORE DATABASE CLUSTER)

The Hive Metastore database is used for the following.

- Table, partition and column definitions (metadata)
- Active table locks for concurrency control
- Active transactions

In a default HDP installation, the Hive Metastore is a single-node MySQL database. Of course, Hive supports multiple database backends as long as there is a Java JDBC connector for it. In this solution, Galera Cluster for MySQL is used for the Hive Metastore database.

Galera Cluster for MySQL (http://galeracluster.com/) can be used to provide the following.

- Synchronous replication
- Active-active multi-master topology
- Read and write to any cluster node
- Direct client connections, native MySQL look and feel

This means that read and write SQL operations can be sent to any Galera Cluster node in the cluster and the system will be 100% consistent at all times. It can also automatically handle a failure of multiple nodes. It uses the standard MySQL client which means that the standard MySQL JDBC is all that is needed for Hive to use it.

With Galera Cluster for MySQL, a table created in any site will be immediately accessible from all other sites. A table lock or transaction created by any site will be immediately recognized by any other site. These are features that Hive has been designed to support within a single site but when Hive is used with a distributed synchronous database cluster, these features extend to multiple Hadoop clusters.

Although Galera Cluster for MySQL was selected for this guide, other RDBMs can be used as well. In particular, MariaDB Server™ is powered by Galera and enterprise support is offered by MariaDB™ (https://mariadb.com). However, only Galera Cluster for MySQL was tested as part of this solution.

HORTONWORKS DATA PLATFORM (HDP) HADOOP CLUSTER

Each site has an independent HDP Hadoop installation. This allows each site to have different types and amounts of resources, different administrators and administrative policies, etc.

Hive is configured to use one of the nodes running Galera Cluster for MySQL for the common Hive Metastore database that is shared between all sites. Hive concurrency must be enabled. Table locks and transactions, instead of being stored in ZooKeeper (the default), must be stored in the Hive Metastore database. This is what allows concurrency to work across sites as well as within sites. Hive ACID transactions can be enabled if required. Just as with a single-site installation, only a single instance of HiveServer across all sites should be designated as the Hive compactor initiator.

CONSISTENCY GUARANTEES

When considering this solution (or any distributed system for that matter) in the enterprise, special care must be taken to understand the consistency guarantees (semantics) offered by the system. ECS is an object storage system and provides guarantees that are generally better than other object storage systems but less strict than a typical file system such as NFS or HDFS. Of particular note is the fact that on ECS, the order of the asynchronous file replication between sites is *not* guaranteed. To provide a simple example, if a 1 GB file is saved to ECS, and then a 1 KB file is saved to ECS, it is possible that the 1 KB file will finish replication before the 1 GB file. Under normal operating conditions, when all sites are online, this would be invisible to users as I/O on a file that hasn't been replicated locally will simply result in remote I/O to the site that owns the file. This means that strong consistency is maintained even with replication delays, as long as all sites are online.

However, when some sites are offline or otherwise inaccessible from other sites, the details of the asynchronous replication are important to understand. In a default ECS installation, the temporary failure of a site will result in I/O errors on some reads and all writes. In terminology of the CAP theorem, under a partition, the system will sacrifice availability to maintain strong consistency.

For use cases where availability is more important than strong consistency, a bucket may be enabled for Access During Outage (also known as Temporary Site Outage or TSO). With Access During Outage enabled, when a local VDC recognizes that a remote VDC has been down for a specific amount of time (approximately 15 minutes), it will answer read requests based on the data that has been replicated to it and the remaining VDCs. It will do so even though the data may not be up-to-date. Again in the terminology of the CAP theorem, under a partition, Access During Outage sacrifices strong consistency in order to provide increased availability for reads.

Although the replication order of different files is not guaranteed, multiple updates to the same file are guaranteed to be replicated in order.

Galera Cluster for MySQL provides synchronous replication for the Hive Metastore so consistency is guaranteed at all times and availability is guaranteed as long as two of three nodes are online. When all systems are online, ECS and the Hive Metastore are 100% consistent. Even if failures occur, it is still guaranteed that the Hive Metastore is *never older* than the data on ECS.

The specific impact of these consistency properties is described in more detail in the *Failure Handling* section of this guide.

HADOOP CLIENT ACCESS TO ECS

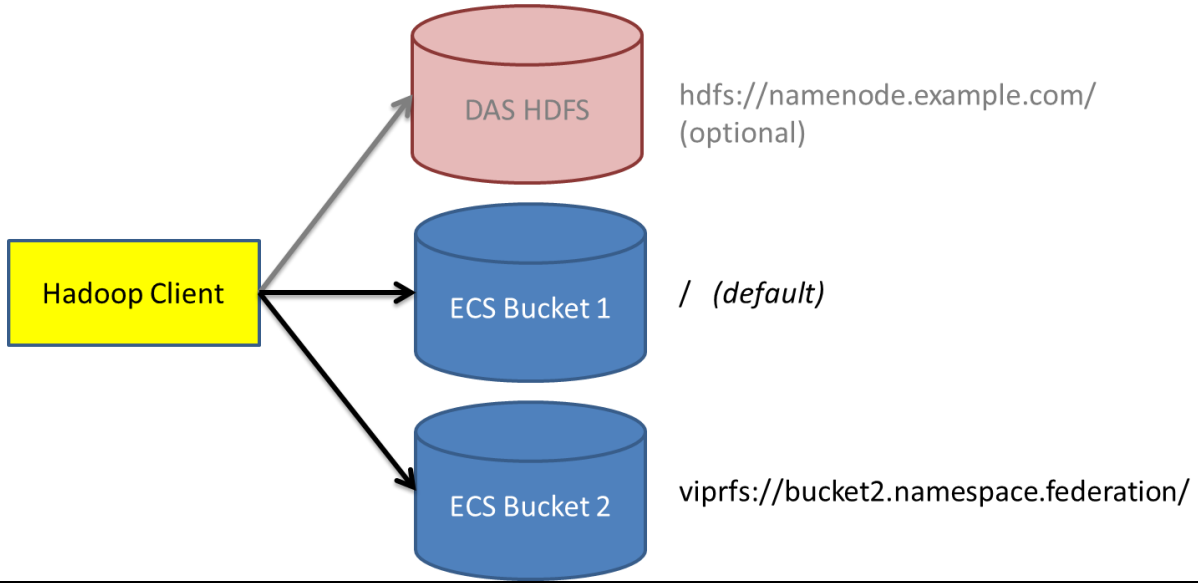
A traditional HDFS URI is in the format *hdfs://namenode.example.com:8020*. To access a bucket on ECS using the Hadoop Compatible File System (HCFS) client library, the URI is in the format *viprfs://bucket.namespace.installation*. *namespace* and *bucket* refer to the ECS bucket. *installation* must match one of the installations listed in the fs.vipr.installations setting in core-site.xml. An ECS installation identifies a single ECS federation consisting of one or more VDCs (sites). The setting fs.vipr.installation.*installation*.hosts identifies the list of site-local ECS nodes that are in the ECS federation. In effect, when a Hadoop client resolves viprfs://bucket.namespace.installation, it looks up fs.vipr.installation.*installation* for a list of site-local ECS nodes to contact. It will spread the load across each of these nodes. If an object read request can't be satisfied from the local VDC, then the local VDC will forward the request to the remote VDC that owns the object.

DEPLOYMENT OPTIONS

ECS AS THE DEFAULT HADOOP FILE SYSTEM

An ECS bucket can be used as the default file system in Hadoop, in which all Hadoop paths without an explicit URI prefix such as *hdfs://* refer to objects in a single ECS bucket. In this scenario, fs.defaultFS in core-site.xml is set to *viprfs://bucket.namespace.installation/*. Data on the traditional HDFS NameNode and DataNodes will be accessible by using the fully-qualified URI such as *hdfs://namenode.example.com/*. Some applications that require the Apache HDFS implementation will not work in this Hadoop cluster.

Figure 3. AN ECS BUCKET AS THE DEFAULT FILE SYSTEM



A single ECS bucket is the default Hadoop file system for a specific site. This bucket can be replicated to other sites or not. Other ECS buckets (replicated or not) can be accessed by a fully-qualified URI beginning with *viprfs://*. A traditional DAS HDFS is optional and could be accessed with the fully-qualified URI beginning with *hdfs://*.

AUTHENTICATION (KERBEROS)

The default authentication in Hadoop is called standard authentication. Standard authentication provides minimal security and is intended to run in an environment where everyone with network access to the Hadoop nodes can be trusted. Environments with untrusted users should use Kerberos to protect access to sensitive data.

Although Kerberos authentication is supported by ECS, in the current version of this guide, only standard authentication is considered.

PRODUCTS AND VERSIONS

Below is the list of products and versions that this guide is based on.

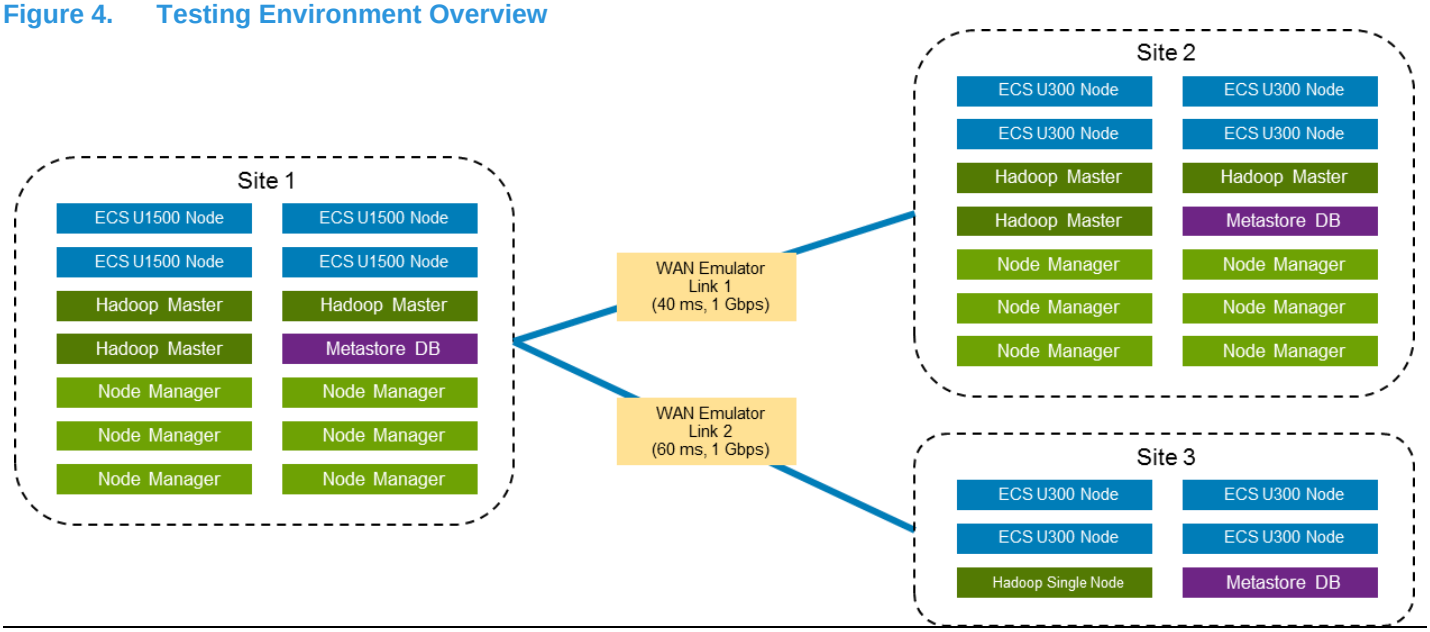
Table 1. Products and Versions

PRODUCT / COMPONENT	VERSION	DETAILS
ECS OS	3.0.0.0-1422.d46985b.663	ecs-os-setup-target.x86_64-3.0.0.0-1422.d46985b.663.install.iso
ECS Fabric and Object	3.0.0.0.85807.98632a9	ecs-3.0.0.0-2398.9f9f451.582-production.tgz
ECS Client	3.0.0.0.85807.98632a9	hdfsclient-3.0.0.0.85807.98632a9.zip
Galera Replication Plugin (galera-3)	25.3.19-2.el7	
MySQL server with wsrep	5.6.34-25.18.el7	
Hortonworks Hadoop Data Platform (HDP)	2.3.4.7-4	Includes: Hadoop 2.7.1.2.3 Hive 1.2.1.2.3 Spark 1.5.2.2.3
Apache Ambari	2.2.2.0	
CentOS Linux	7.2.1511 x64	Used for all hosts except ECS nodes

TESTING ENVIRONMENT

The environment used to perform the testing related to this guide was designed to simulate a typical environment with three sites around the world. A WAN emulator was used in order to emulate a specific latency and bandwidth between the simulated sites. All hosts in this testing environment were connected to a single core switch but each site had a different VLAN and the VLANs were bridged via the WAN emulator. By using a WAN emulator instead of physically separating the hosts in different data centers, latencies could be easily changed or even reduced to zero in order to understand the impact of WAN latency on the workloads.

Figure 4. Testing Environment Overview



Round trip latencies are shown. Communication between Site 2 and Site 3 travels through both links for a round trip ping latency of 100 ms.

Table 2. List of resources and technologies used to support the testing process

DESCRIPTION	VERSION	DETAILS & SPECIFICATIONS
(1) ECS U1500 rack	3.0.0.0	(4) nodes per rack Each node has: (60) 6 TB, 7200 RPM drives (1) Intel® Xeon® CPU E5-2609 v2 @ 2.50GHz 64 GB RAM (2) 10 GbE Arista 7124 switch (4) 10 Gbps uplinks to Node Managers 1440 TB raw capacity
(2) ECS U300 rack	3.0.0.0	(4) nodes per rack Each node has: (15) 6 TB, 7200 RPM drives (1) Intel® Xeon® CPU E5-2609 v2 @ 2.50GHz 64 GB RAM (2) 10 GbE Arista 7124 switch (4) 10 Gbps uplinks to Node Managers 360 TB raw capacity
(12) Node Managers		2x Intel Xeon E5-2650 v3 Processor (2.3 GHz, 10 Cores, 10x 256KB L2 and 25MB L3 Cache) 256 GB RAM 2x 10GbE (Intel X520-DA2), 4x 1GbE (AF1 Slot), 4x 1GbE (AF2 Slot) [Only a single 10 GbE NIC was used for data] 1x 1Gb OOB (Motherboard) 1x 32 GB SSD [used for OS and application binaries] 12x 600GB, SAS, 10K, 2.5 inch [all 12 used for shuffle/intermediate data] CentOS 7.2
(6) Hadoop Master Nodes		VMware Virtual Machine 32 GB Memory 4 CPU Cores 128 GB hard disk CentOS 7.2
(1) Hadoop Single Node		VMware Virtual Machine 64 GB Memory 4 CPU Cores 128 GB hard disk CentOS 7.2
(3) Metastore DB Nodes		VMware Virtual Machine 4 GB Memory 1 CPU Cores 50 GB hard disk CentOS 7.2
(1) Apposite Technologies Linktropy 8510 WAN emulator	4.4	4 Links with (2) 1 Gbps ports each Link 1 configured for 20 ms constant delay in each direction for a round trip ping latency of 40 ms. Link 2 configured for 30 ms constant delay in each direction for a round trip ping latency of 60 ms. Communication between Site 2 and Site 3 travels through both links for a round trip ping latency of 100 ms.
(1) Cisco Nexus 7718 switch		Each ECS rack connects to this core switch via their Arista 7124 uplink switches (four 10 Gbps link each). All other hosts connect directly to this core switch (single 10 Gbps link for each Node Manager). Hosts at different sites are placed into separate VLANs that are bridged together using the Linktropy WAN emulator.

FUNCTIONALITY EXAMPLE

The example script below demonstrates what is possible with this active/active Hive solution.

Table 3. Functionality Example

STEP	SITE 1	SITE 2
1	hive> use db1;	hive> use db1;
2	hive> drop table if exists demotab1; hive> create table demotab1 (id int, s string) partitioned by (site int) clustered by (id) into 4 buckets stored as orc tblproperties ('transactional'='true');	
3	hive> insert into demotab1 partition (site=1) values (11, 'ant'), (12,'bear');	
4		hive> select * from demotab1; 12 bear 1 11 ant 1
5		hive> insert into demotab1 partition (site=2) values (21, 'cat'), (22,'dog');
6	hive> select * from demotab1; 12 bear 1 11 ant 1 21 cat 2 22 dog 2	
7	hive> delete from demotab1 where site=2 and id=21;	
8		hive> select * from demotab1 where site=2; 12 bear 1 11 ant 1 22 dog 2

A new transactional table is created (step 2) and populated (3) from site 1. Then the table is read from site 2 (4). Since the system guarantees consistency of the metadata and data, no waiting is needed before reading the table, even if the actual replication of the data is delayed. Next, another record is added to the table from site 2 (5) and it is immediately read from site 1 (6). Finally, a record is deleted from site1 (7) and the table is immediately read from site 2 (8).

FUNCTIONALITY DETAILS

The functionality of the solution depends on the overall health of the system. For details of the tests performed to validate this section, refer to the *Functional Testing Procedure* in the appendix.

The functionality explained in this section only applies to data which is stored on the replicated ECS bucket referred to as *hive-warehouse-1*.

STEADY STATE BEHAVIOR (ALL SITES ONLINE)

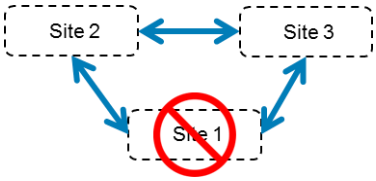
When all sites are online and communicating with each other, everything is fully functional.

- File creates, updates, and deletes from Hadoop at any site are immediately effective when accessed by Hadoop at any other site (file system consistency).
- Table creation and loading from any site is immediately effective at any other site.
- Loading of a new table partition from any site is immediately effective at any other site.
- For tables enabled for transactions, inserts, deletes, and updates from any site are immediately effective at any other site.
- Hive table compaction occurs only on the primary site but is immediately effective at any other site.
- Hive concurrency across sites works as expected as table locks are in the shared Metastore DB.
- During testing, there were no inconsistencies encountered between the Hive Metastore and the ECS bucket across any of the sites.

- Supports concurrent loading into different partitions from different sites
    - e.g. Site 1 loads into the partition (date=2016-09-04, site=1) while Site 2 loads into the partition (date=2016-09-04, site=2) concurrently
- Supports concurrent select (read) queries across partitions created from different sites
- Repeated select queries on tables loaded from a remote site will improve in performance over time while the files are cached on the local ECS VDC.
- TPC-DS data generation and queries work as expected (tested with scale factor of 3 TB).

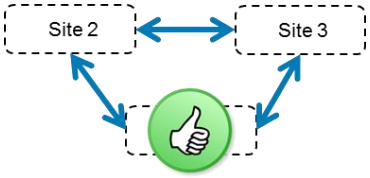
FAILURE HANDLING

The following occurs upon a complete outage of a single site (e.g. site disaster, power failure or WAN link failure).



- The Hive Metastore DB nodes at the two remaining sites will continue to be fully functional (read/write, active/active, synchronous).
- During the first 15 minutes after such a failure, some read requests to ECS will return an error, depending on which site owns the object. Additionally all write requests to ECS will return an error.
- After 15 minutes, an optional feature called TSO (temporary site outage) will become active.
    - The optional TSO feature sacrifices some consistency to provide availability.
    - TSO can be enabled and disabled on a per-bucket basis. In the ECS administrative UI, TSO is referred to as *Access During Outage*.
    - Read requests will be satisfied from the asynchronously replicated data across the remaining two sites.
    - Read-only Hive select queries will generally work.
    - All write requests will continue to fail.
- Because replication occurs asynchronously and the order of replication is not guaranteed, the Hive Metastore, file listings and file contents may not be 100% consistent.
- The following anomalous conditions are possible if TSO becomes active:
    - A recently created file appears to be missing.
    - A recently created file is returned in a directory listing but its contents cannot be read.
    - A recently updated file reverts to an older version.
    - A recently deleted file reappears.
    - The output directory of a recently completed MR or Spark job may have a _SUCCESS file but some output files may be missing.
    - A recently loaded Hive table partition may be defined in the Hive Metastore but all or some of the data files may be missing.
    - A recently deleted Hive table partition may have been removed from the Hive Metastore but all or some of the data files may still exist.
    - A recent Hive transaction consisting of multiple files may be partially or completely lost.
    - A recent Hive concatenation or compaction may leave the partition in an inconsistent state.

- If TSO is disabled, any I/O which could possibly return an inconsistent or anomalous result will instead return an error.



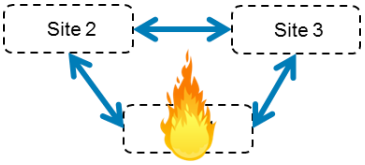
RECOVERY FROM A TEMPORARY FAILURE

The following occurs upon the site being restored (e.g. power restored or WAN link restored).

- All read and write I/O requests will succeed and strong consistency will be enforced
- Asynchronous replication will resume
- Any anomalous conditions that were visible during TSO will no longer exist

RECOVERY FROM A PERMANENT FAILURE

In the event that the failed site can't be recovered, the ECS VDC should be removed from the federation, and then the following will occur.



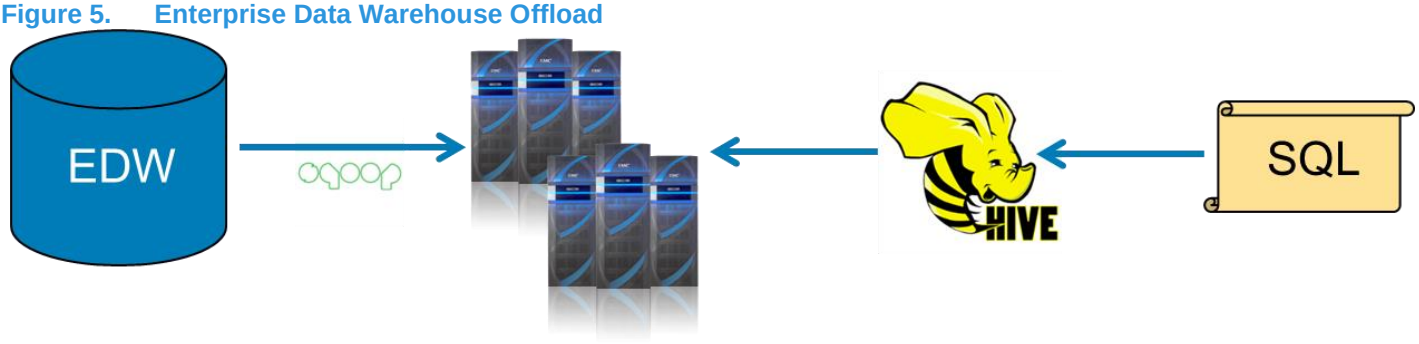
- All read and write I/O requests will succeed and strong consistency will be enforced.
- Asynchronous replication will resume.
- Any anomalous conditions will continue to exist and may need to be cleaned up, e.g.
    - Drop and reload recently loaded or modified table partitions
    - Manually delete directories corresponding to recently deleted table partitions
    - Rerun recent MR or Spark jobs
- If the failed site contained the Hive Compactor Initiator, enable the Hive Compactor Initiator on one of the remaining sites.
- To go back to a 3-site federation, install a new ECS VDC and add it to the replication group.

USE CASES

This solution can be an integral part of a variety of use cases. Below are two such use cases.

ENTERPRISE DATA WAREHOUSE OFFLOAD

Figure 5. Enterprise Data Warehouse Offload



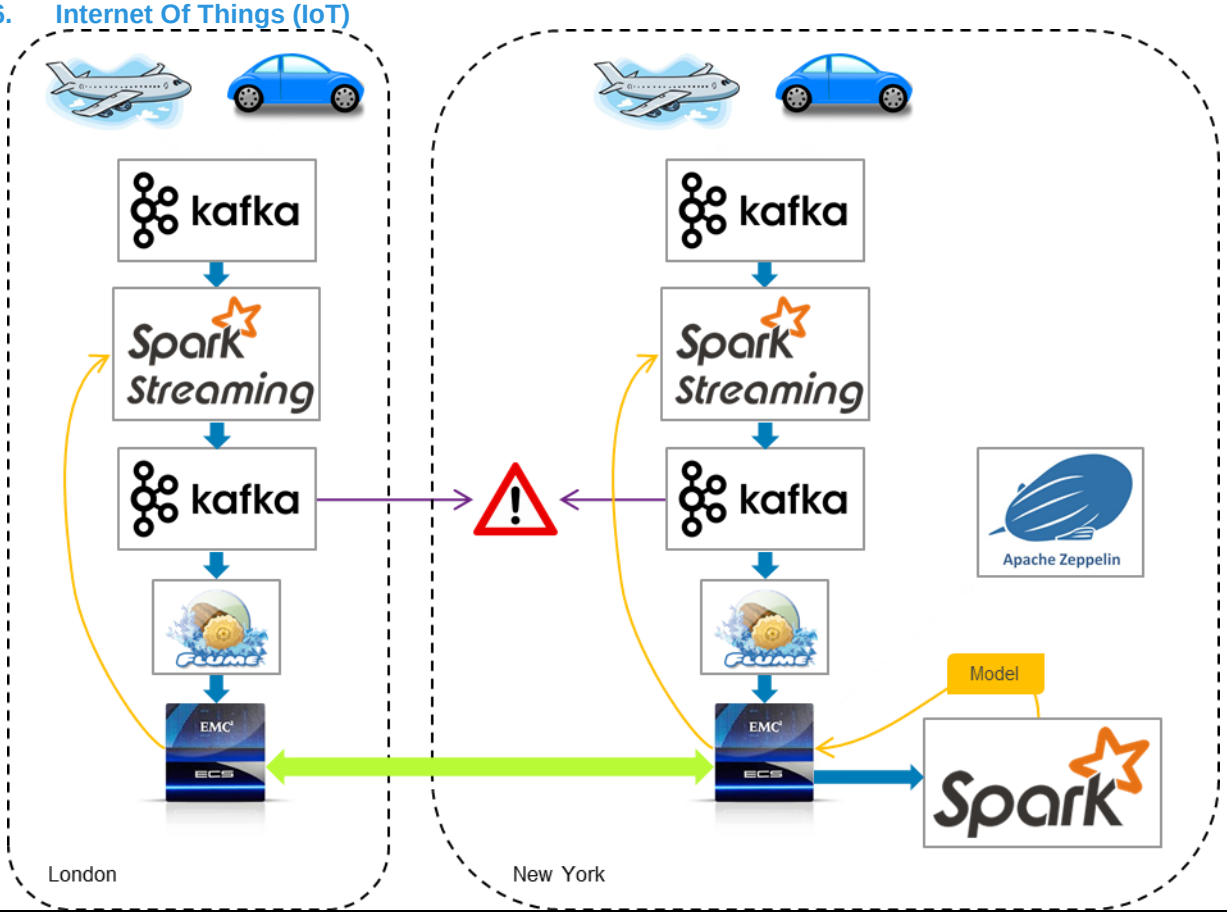
Tables and records are exported from EDWs at any site to Hive on ECS with Apache Sqoop. Data on ECS can be queried in-place from any site using Hive SQL.

- Tables and records can be exported from an EDW (e.g. Teradata) to Hive on ECS with Apache Sqoop or similar tools

- Data will be efficiently distributed and protected across multiple sites
- Data can be queried in place using Hive SQL from any site
- If desired, exported records can be deleted from the EDW
- Simple recovery from site failures: Rerun recent Sqoop export jobs

INTERNET-OF-THINGS (IOT)

Figure 6. Internet Of Things (IoT)



An example of an IoT use case utilizing an active/active Hadoop system. For details, visit http://bit.ly/1QcYwst.

- IoT messages are received by the nearest data center and placed in a local Kafka cluster
- HDF, Flume, Spark, or Storm pull messages from Kafka and place them into Hive table partitions which are located in a geo-replicated ECS bucket
- The table is partitioned by hour and site and can be loaded in parallel
- Adhoc or operational SQL queries can be executed against the partitions of a single site or multiple sites
- Repeated queries on remotely loaded partitions will cause the partition's data to get cached by the local ECS VDC, providing fast yet strongly consistent access to the data

For details on this particular use case, visit http://bit.ly/1QcYwst.

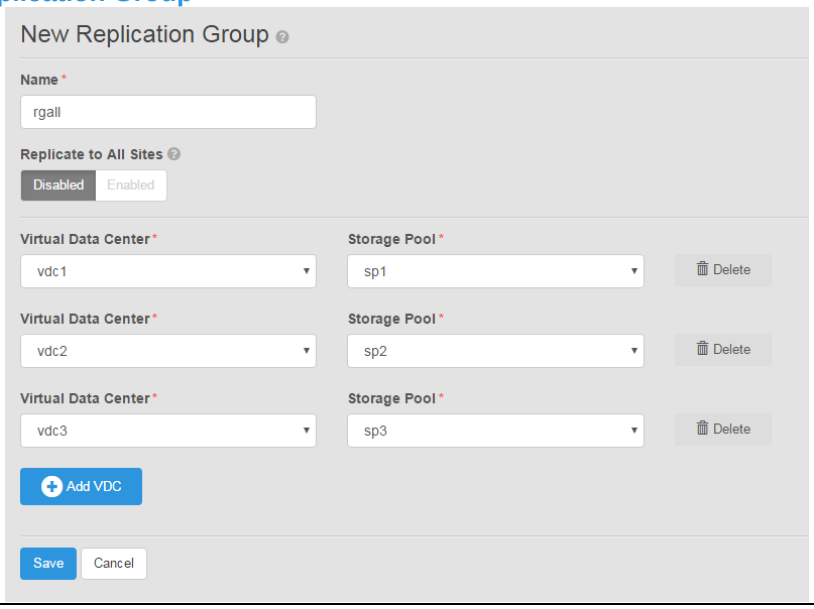
INSTALLATION AND CONFIGURATION

ECS

Follow the procedures in the *ECS Administrator's Guide* to perform the steps below.

1. Configure storage pools.
2. Configure Virtual Data Centers (VDCs). There should be one or more VDCs for each site and/or data center.
3. Configure the shared-data replication group. Replication groups are used to define the VDCs that buckets are replicated to. In general, the bucket containing the Hive warehouse should be in a replication group that includes all sites. This document assumes this replication group is named *rgall* but any valid name can be used.

Figure 7. Create a Replication Group



A replication group is created with three Virtual Data Centers.

4. Configure the site-local replication groups. For each VDC, create a replication group that only exists at the single VDC. This document assumes these replication groups are named *rg1*, *rg2*, and *rg3*. but any valid name can be used.
5. Configure a namespace. This document will use the namespace *ns1* but any valid name can be used.
6. Configure users and tenants. At a minimum, create a user named *hdfs*.
7. For each Hadoop cluster, create a default file system (root) bucket.
    a. Create a bucket named *mycluster1-root*. A different name can be used if desired.
    b. Replication Group: *rg1* (This should match the replication group that spans just this site's VDC.)
    c. Bucket Owner: *hdfs*
    d. File System: Enabled
    e. Default Bucket Group: *hdfs*
    f. Group File Permissions: Read, Write checked
    g. Group Directory Permissions: Read, Write, Execute checked

h. Access During Outage: Enabled

i. Bucket ACLs Management: The ACL below will provide full access to all users.

i. Group Name: all users

ii. Permissions: Full Control

8. Create the Hive warehouse bucket.

a. Create a bucket named *hive-warehouse-1*. A different name can be used if desired.

b. Replication Group: *rgall*

c. Bucket Owner: *hdfs*

d. File System: Enabled

e. Default Bucket Group: *hdfs*

f. Group File Permissions: Read, Write checked

g. Group Directory Permissions: Read, Write, Execute checked

h. Access During Outage: Enabled

i. Bucket ACLs Management: The ACL below will provide full access to all users.

i. Group Name: all users

ii. Permissions: Full Control

GALERA CLUSTER FOR MYSQL (HIVE METASTORE DATABASE CLUSTER)

It is recommended to have exactly three hosts running Galera Cluster for MySQL. With three or more sites, place one host at each of the three most used or reliable sites. Using more than three hosts is unlikely to provide significantly more protection or performance and may actually reduce performance in some cases. With only two sites, place two hosts on the "primary" site and one host on the other site. With three hosts, Galera Cluster will provide read/write synchronous database access as long as two hosts are communicating with each other.

Complete documentation for installing Galera Cluster for MySQL is available at http://galeracluster.com/documentation-webpages/.

For convenience, the minimal set of steps is shown below.

1. Create the YUM repository file */etc/yum.repos.d/galera.repo* with the following contents.

```
[galera]
name = Galera
baseurl = http://releases.galeracluster.com/centos/7/x86_64
gpgkey = http://releases.galeracluster.com/GPG-KEY-galeracluster.com
gpgcheck = 1
```

2. Install Galera Cluster for MySQL.

```
# yum upgrade -y mysql-wsrep-shared-5.6
# yum install -y galera-3 mysql-wsrep-5.6
```

3. Create the MySQL configuration file */etc/my.cnf* with the following contents. Edit the parameter *wsrep_cluster_address* with the list of FQDNs for the three hosts. This file will be identical on all hosts.

```
[mysqld]
```

```
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
binlog_format=ROW
bind-address=0.0.0.0
default_storage_engine=innodb
innodb_autoinc_lock_mode=2
innodb_flush_log_at_trx_commit=0
innodb_buffer_pool_size=122M
wsrep_provider=/usr/lib64/galera-3/libgalera_smm.so
wsrep_provider_options="gcache.size=300M; gcache.page_size=300M"
wsrep_cluster_name="mydbcluster"
wsrep_cluster_address="gcomm://mydbcluster-0.example.com,mydbcluster-1.example.com,mydbcluster-
2.example.com"
wsrep_sst_method=rsync
[mysql_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

4. Repeat steps 2 - 4 on the two remaining hosts.

5. On the first host, start MySQL Server and initialize replication from any existing databases stored on this node.

```
# service mysql start --wsrep-new-cluster
```

6. View the MySQL password, login to MySQL, and change the password.

```
# cat ~/.mysql_secret
# The random password set for the root user at Tue Jul 26 21:53:28 2016 (local time): c9oI1_nKXwgq0iKs
# mysql --password=c9oI1_nKXwgq0iKs
mysql> SET PASSWORD=PASSWORD('MyNewPassword');
```

7. View the replication status. wsrep_cluster_size is the number of active hosts in the Galera Cluster for MySQL.

```
# mysql --password
mysql> SHOW STATUS LIKE 'wsrep_cluster_size';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 1     |
+--------------------+-------+
```

8. On each of the two remaining hosts, start MySQL Server.

```
# service mysql start
```

9. View the replication status. There should now be three active hosts.

```
# mysql --password
mysql> SHOW STATUS LIKE 'wsrep_cluster_size';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 3     |
+--------------------+-------+
```

10. Create the account for the Hive Metastore. Note that the Hive database will be created during the HDP installation process.

```
mysql> CREATE USER 'hive'@'%' IDENTIFIED BY 'mypassword';
mysql> GRANT ALL PRIVILEGES ON hive.* TO 'hive'@'%'  WITH GRANT OPTION;
mysql> SHOW GRANTS FOR 'hive'@'%';
```

Tip: To "cold start" Galera Cluster when there are no active hosts already running, execute "service mysql start --wsrep-new-cluster" on one of the nodes and then "service mysql start" on the two other hosts. However, if the hosts were not shut down properly and you suspect that some of these hosts have an older version of the database, carefully read the Galera Cluster documentation for the proper recovery procedure.

HORTONWORKS DATA PLATFORM (HDP)

HDP must be independently installed at each site. At each site, use one of the methods described below to install HDP.

INSTALLING HDP WITH THE AMBARI HADOOP ECS STACK

To perform a new installation of HDP with an ECS bucket as the default Hadoop file system, follow the instructions in the *ECS Data Access Guide* but with the following changes.

1. After installing Ambari but before logging into the Ambari UI, install the MySQL connector for Ambari. This will allow Ambari to provision and manage the Hive Metastore on Galera Cluster for MySQL.

```
# yum install -y mysql-connector-java
# ambari-server setup --jdbc-db=mysql --jdbc-driver=/usr/share/java/mysql-connector-java.jar
```

2. Customize the Hive database.

   2.1 Hive Database: Existing MySQL Database

   2.2 Database Host: This should be the FQDN of the Galera Cluster for MySQL host that resides at this site.

   2.3 Database Password: This should match the password for the hive user previously created in Galera Cluster for MySQL.

   2.4 Click *Test Connection* and confirm that the connection is OK.

INSTALLING HDP WITHOUT THE AMBARI HADOOP ECS STACK

This section describes the procedure to deploy a standard HDP installation and then adds the ECS client.

Complete details for this procedure can be found at http://docs.hortonworks.com/HDPDocuments/Ambari-2.2.2.0/bk_Installing_HDP_AMB/content/_download_the_ambari_repo_lnx7.html. A concise list of steps is shown below.

1. Install Ambari.

```
# wget -nv http://public-repo-1.hortonworks.com/ambari/centos7/2.x/updates/2.2.2.0/ambari.repo -O
/etc/yum.repos.d/ambari.repo
# yum install -y ambari-server
# ambari-server setup -s
# ambari-server start
```

2. Install the MySQL connector for Ambari. This will allow Ambari to provision and manage the Hive Metastore on Galera Cluster for MySQL.

```
# yum install -y mysql-connector-java
# ambari-server setup --jdbc-db=mysql --jdbc-driver=/usr/share/java/mysql-connector-java.jar
```

3. Login to the Ambari UI by browsing to http://site1-master-0.example.com:8080/.
4. Follow the wizard to complete the installation process. In general, choose the defaults unless specified below.
5. Choose HDP version 2.3.
6. Choose Services. At a minimum, select the following:

6.1 HDFS

6.2 YARN + MapReduce2

6.3 Tez

6.4 Hive

6.5 Pig

6.6 ZooKeeper

6.7 Ambari Metrics

7. Customize the Hive database.

7.1 Hive Database: Existing MySQL Database

7.2 Database Host: This should be the FQDN of the Galera Cluster for MySQL host that resides at this site.

7.3 Database Password: This should match the password for the hive user previously created in Galera Cluster for MySQL.

7.4 Click *Test Connection* and confirm that the connection is OK.

8. Complete the HDP installation.

9. Enable Hive concurrency and transactions. Use Ambari to configure the following settings.

Table 4. Hadoop Configuration to Enable Hive Concurrency and ACID Transactions

LOCATION	PROPERTY	VALUE
Hive Settings	ACID Transactions (hive_txn_acid)	true
Hive Settings	Run Compactor (hive.compactor.initiator.on)	If compaction is required, this should be true on one node of one site only. The site that has the bulk of the data should perform the compactions.
Hive Settings	Number of threads used by Compactor (hive.compactor.worker.threads)	Set to 0 on sites that should never perform compaction.
Hive Advanced General	Transaction Manager (hive.txn.manager)	org.apache.hadoop.hive.ql.lockmgr.DbTxnManager
Hive Advanced General	Enforce bucketing (hive.enforce.bucketing)	true
Hive Advanced General	Allow all partitions to be Dynamic (hive.exec.dynamic.partition.mode)	nonstrict
Advanced hive-site	Use Locking (hive.support.concurrency)	true

CONFIGURE ECS CLIENT ON HDP NODES

DEPLOY ECS CLIENT JAR

If you did not already deploy the ECS client jar to all Hadoop nodes, the steps below can be used.

1. Create a text file named masters that contains a list of IP addresses or FQDNs for all Hadoop master nodes, one per line.

2. Create a text file named workers that contains a list of IP addresses or FQDNs for all Hadoop worker nodes, one per line.

3. Create the directory /usr/lib/hadoop/lib on all nodes.

```
# cat masters workers | xargs -i -n 1 ssh root@{} mkdir -p /usr/lib/hadoop/lib
```

4. Copy the ECS client jar to all nodes.

```
# cat masters workers | xargs -i -n 1 scp viprfs-client-3.0.0.0-hadoop-2.7.jar root@{}:/usr/lib/hadoop/lib/
```

CONFIGURE ECS CLIENT PROPERTIES

Use Ambari to set the following configuration properties that are required by the ECS client.

Table 5. Hadoop Configuration to Enable ECS Access

LOCATION	PROPERTY	VALUE
core-site	fs.viprfs.impl	com.emc.hadoop.fs.vipr.ViPRFileSystem
core-site	fs.AbstractFileSystem.viprfs.impl	com.emc.hadoop.fs.vipr.ViPRAbstractFileSystem
hdfs-site	fs.permissions.umask-mode	022
core-site	fs.viprfs.auth.identity_translation	NONE
core-site	fs.viprfs.auth.anonymous_translation	LOCAL_USER
core-site	fs.vipr.installations	federation1 (this can be any name and will be referred to as $FEDERATION). If you have multiple independent ECS federations, enter multiple values separated by commas.
core-site	fs.vipr.installation.$FEDERATION.hosts	(comma-separated list of FQDN or IP address of each ECS host in the local site)
core-site	fs.vipr.installation.$FEDERATION.hosts.resolution	dynamic
core-site	fs.vipr.installation.$FEDERATION.resolution.dynamic.time_to_live_ms	900000
yarn-site	yarn.application.classpath	Append the following: /usr/lib/hadoop/lib/*
mapred-site	mapreduce.application.classpath	Append the following: /usr/lib/hadoop/lib/*
tez-site	tez.cluster.additional.classpath.prefix	Append the following: /usr/lib/hadoop/lib/*
HDFS	hadoop-env template	Append the following: export HADOOP_CLASSPATH=${HADOOP_CLASSPATH}:/usr/lib/hadoop/lib/*
Spark	spark-env template	Append the following: export SPARK_DIST_CLASSPATH="${SPARK_DIST_CLASSPATH}:/usr/lib/hadoop/lib/*:/usr/hdp/current/hadoop-client/client/guava.jar"

CHECK HADOOP ACCESS TO ECS

Once all Hadoop services have started (actually just the client configurations need to be refreshed), ensure that the ECS bucket can be accessed using the Hadoop CLI. The URI will be of the form viprfs://*bucket*.*namespace*.*federation*/. In the example commands below, first, a directory listing is attempted. A new bucket will be empty and nothing will be returned. Then an empty file is created, followed by another directory listing.

```
[root@mycluster1-master-0 ~]# hdfs dfs -ls viprfs://hive-warehouse-1.ns1.federation1/
[root@mycluster1-master-0 ~]# hdfs dfs -touchz viprfs://hive-warehouse-1.ns1.federation1/THISIS_hive-warehouse-1
[root@mycluster1-master-0 ~]# hdfs dfs -ls viprfs://hive-warehouse-1.ns1.federation1/
Found 1 items
-rw-------   1 root          0 2015-08-26 19:05 viprfs://hive-warehouse-1.ns1.federation1/THISIS_hve-warehouse-1
```

RELOCATE THE DEFAULT FILE SYSTEM FROM HDFS TO AN ECS BUCKET

This section does not apply if the Ambari Hadoop ECS stack was used to install HDP.

Although the system is now usable and will appear to work well, a configuration with HDFS as the default file system is not supported by Dell EMC. It is therefore recommended to relocate the default file system from HDFS to the cluster's root ECS bucket. This procedure will copy all files from the HDFS file system to an ECS bucket and then set the ECS bucket as the default file system.

1. Use Ambari to stop all services except HDFS, YARN, and Zookeeper.
2. Next, copy all existing files on the DAS HDFS file system to the ECS bucket. Even for a new installation of Hadoop, there are critical directories that must exist in the default Hadoop file system. Use DistCp to perform the file copy.

   ```
   [hdfs@mycluster1-master-0~]$ hadoop distcp -skipcrccheck -update -pugp -i / viprfs://mycluster1-root.ns1.federation/
   ```

3. Use Ambari to configure the following settings.

Table 6. Hadoop Configuration to Enable Hive Concurrency and ACID Transactions

LOCATION	PROPERTY	VALUE
HDFS Advanced core-site	fs.defaultFS	viprfs://mycluster1-root.ns1.federation1
Spark Advanced spark-defaults	spark.eventLog.dir	viprfs://mycluster1-root.ns1.federation1/spark-history
Spark Advanced spark-defaults	spark.history.fs.logDirectory	viprfs://mycluster1-root.ns1.federation1/spark-history

4. Use Ambari to stop and start all services.
5. Ensure proper directory permissions. If DistCp encounters any errors, the necessary permissions may not have been applied to critical directories. The following commands will set the correct permissions.

```
[hdfs@mycluster1-master-0~]$
hadoop fs -chmod 777 /apps/hive/warehouse
hadoop fs -chown hive:hdfs /apps/hive/warehouse
hadoop fs -chmod -R 770 /user/ambari-qa
hadoop fs -chown -R ambari-qa:hdfs /user/ambari-qa
```

TEMPORARY HIVE FILES

During the execution of some queries, Hive may need to create temporary files locally on each Node Manager and/or on the distributed file system (e.g. HDFS). Although these files could be created on a replicated ECS bucket, doing so creates unnecessary load on ECS and the WAN. It is best to ensure that these temporary files intended for HDFS actually get stored in a non-replicated ECS bucket. Hive uses the directory defined by the parameter hive.exec.scratchdir for temporary files intended to be stored on HDFS. The default value of this parameter is */tmp/hive* which is a relative path that gets appended to the fs.defaultFS parameter. If the fs.defaultFS parameter references a site-local ECS bucket as recommended in the above procedures, then no change is required. However, if fs.defaultFS has been changed to use a replicated ECS bucket, then hive.exec.scratchdir should be changed to a fully-qualified path to a site-local ECS bucket such as *viprfs://site1tmp.ns1.federation1/tmp/hive*.

USING HADOOP WITH ECS

HIVE AND BEELINE CLI

The original CLI for executing interactive SQL commands on Hive is the Hive CLI and it is started by running *hive* with no parameters. The Hive CLI connects to the Hive Metastore service for metadata operations and launches MapReduce or Tez jobs to execute queries.

```
# hive
hive> select * from db1.tab3;
hive> exit;
```

A newer and more flexible SQL CLI is called Beeline. It connects to HiveServer2 using a Thrift API and all SQL queries and results travel through this interface.

```
# beeline -u jdbc:hive2://site1-master-0.example.com:10000
0: jdbc:hive2://site1-master-0.example.com> select * from db1.tab3;
0: jdbc:hive2://site1-master-0.example.com> !quit
```

USING HIVE WITH ECS

Each table in Hive has a location which identifies the URI to its data files. The "show create table" command will display the location.

```
hive> show create table ptab2;
OK
CREATE TABLE `ptab2`(
```

```
  `id` int,
  `col_1` boolean,
  `col_2` double,
  `col_3` timestamp)
PARTITIONED BY (
  `part` int)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  'viprfs://hive-warehouse-1.ns1.federation1/apps/hive/warehouse/db1.db/ptab2'
TBLPROPERTIES (
  'transient_lastDdlTime'='1478822665')
```

When a new table is created, the location can be set explicitly using the LOCATION option or it will use the current database's default location. A database's default location can be displayed with the "describe database" command.

```
hive> describe database db1;
OK
db1             viprfs://hive-warehouse-1.ns1.federation1/apps/hive/warehouse/db1.db       hive    USER
```

When a new database is created, the default location for new tables within the database can be set explicitly using the LOCATION option. If not specified, the default location is set to a subdirectory within the Hive warehouse directory that is defined with the hive.metastore.warehouse.dir settings in hive-site.xml.

```
hive> CREATE DATABASE db1 LOCATION 'viprfs://hive-warehouse-1.ns1.federation1/apps/hive/warehouse/db1.db';
```

If you want all new databases to be stored in a specific ECS bucket by default, simply use Ambari to set the parameter hive.metastore.warehouse.dir to the URI to the bucket (e.g. viprfs://hive-warehouse-1.ns1.federation1/apps/hive/warehouse). For the change to take effect, use Ambari to restart all Hive services.

With this flexibility, each Hive table can be stored in a different ECS bucket or HDFS Name Node. You may choose to locate commonly used dimension tables in an ECS bucket that is mirrored to all sites while the large fact tables can be located in an ECS bucket that is efficiently XOR erasure coded across sites. Any temporary tables or data that does not need to be protected can be stored in non-replicated ECS buckets or in a traditional HDFS directory. Even with the tables located in different places, Hive can still transparently join them together in a SQL query.

There is limited support for changing the location of an existing table or database. Refer to the *Hive MetaTool* section in the Appendix for details.

HIVE ACID COMPACTION

A standard non-transactional Hive table only allows whole partitions to be loaded. Inserts, updates, and deletes of only some rows in a partition are not possible. However, if ACID transaction support is enabled, specific tables can be marked as transactional. Each transaction is stored as a new directory containing delta files. After many transactions, the large number of delta files can reduce the performance of queries. To solve this problem, Hive has a table compactor that can merge together the delta files into fewer files. A Hive major compaction will read the entire partition (base + deltas), write new base files, and then delete the old files. A Hive minor compaction will only rewrite the delta files, leaving the base alone. For more information on compaction and Hive transactions in general, see https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions.

When all sites are online, the Hive compactor works as designed on the system described in this guide. However, there are several factors to consider when using compaction.

First, the Hive compactor must necessarily rewrite all of the data (or just the delta files for a minor compaction). Writing a new file on ECS involves additional WAN traffic to replicate the data to remote sites. Additionally, the primary copy of the rewritten data will now be located at the VDC that ran the compactor job instead of the VDC that wrote the initial data. This may not be desired in some cases.

Second, as described in the *Failure Handling* section of this guide, it is possible that a Hive partition that is actively being compacted gets only partially replicated to another VDC when a catastrophic failure occurs at the compacting site. Because there is no guarantee of replication order, this can leave the partition in an inconsistent state when TSO takes effect or the VDC is removed from the replication group.

Due to these issues, compaction needs to be handled carefully. In general, automatic compaction should be disabled by setting the table parameter NO_AUTO_COMPACTION to true. For tables where inserts, updates, and deletes are rare, this will have an insignificant impact on performance. When transactions are more common, perhaps on the last partition of a table collecting streaming data, a well-defined procedure to recover from a site disaster should include reloading this entire partition. This might entail completely reprocessing the last day's messages from a Kafka queue, for example. A compaction could then be initiated manually at the end of the day.

The following command shows how to disable automatic compaction of a table. It should be repeated for all tables stored in a replicated ECS bucket.

```
ALTER TABLE table_name SET TBLPROPERTIES ("NO_AUTO_COMPACTION"="true");
```

To initiate manual compaction of a single partition of a table, execute the following command.

```
ALTER TABLE table_name PARTITION (date=20161121) COMPACT 'MINOR';
```

USING SPARK SQL WITH HIVE

In addition to using MapReduce or Tez to execute SQL queries on Hadoop with Hive, Spark SQL can also be used. This can be accomplished in three ways. In each of these methods, the Hive Metastore metadata and Hive libraries are used to locate and retrieve the source table data but the Spark execution engine performs the query processing. Note that Spark SQL does not currently support transactional tables.

USING PYTHON WITH SPARK SQL

Below shows how to use PySpark to execute SQL against the data in a Hive table.

```
# pyspark --master yarn --num-executors 6
>>> from pyspark.sql import HiveContext
>>> sqlContext = HiveContext(sc)
>>> r = sqlContext.sql("select * from db1.tab3")
>>> r.show()
>>> exit()
```

This method allows the entire library of Spark functionality to be applied directly to Hive tables. This includes user-defined functions, machine learning algorithms, graph processing, and lots more.

USING SCALA WITH SPARK SQL

Of course, Scala applications can also be used with the same functionality as PySpark (and more).

```
# spark-shell --master yarn --num-executors 6
scala> val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
scala> val r = sqlContext.sql("select * from db1.tab3")
scala> r.show()
scala> exit()
```

USING BEELINE WITH SPARK SQL

As shown earlier, the Beeline CLI can connect to HiveServer2 to execute SQL using Hive with Tez. However, Beeline can also be used to execute SQL using Spark SQL. To enable this, the Spark Thrift Server must be installed on one of the servers in the cluster using Ambari. Then start Beeline as shown below. Note that the host and port number must match that of the Spark Thrift Server, not HiveServer2.

```
# beeline -u jdbc:hive2://site1-master-0.example.com:10015
0: jdbc:hive2://site1-master-0.example.com> select * from db1.tab3;
0: jdbc:hive2://site1-master-0.example.com> !quit
```

SPARK AND MAPREDUCE

Spark and MapReduce applications can also work with non-Hive files stored on ECS. By non-Hive file, this means any file that is not tabular in nature, such as unstructured text, XML, JSON, images, audio and video. This is straight forward to accomplish and requires 1) that the Java classpath includes the ECS Hadoop client jar file (viprfs) and 2) using a URI in the format *viprfs://bucket.namespace.federation/directory/file*. The necessary classpath changes are documented in the ECS installation section of this guide.

Here is an example of using Spark to count the lines of an unstructured text file on ECS.

```
# pyspark --master yarn --num-executors 6
>>> r = sc.textFile("viprfs://hive-warehouse-1.ns1.federation1/data/data1.txt")
>>> r.count()
>>> exit()
```

CONCLUSION

Dell EMC ECS and Hortonworks Data Platform can be used very effectively to provide a geo-distributed active/active Hadoop and Hive system. The system's strong consistency along with geo-caching enables all sites to read and write Hive tables without limitations. This makes it an ideal system for use by Enterprise Data Warehouse offload and Internet-of-Things.

REFERENCES

Table 7. References

DESCRIPTION	DETAIL / LINKS
ECS Product Page	http://www.emc.com/ecs
ECS 3.0 Product Documentation	https://community.emc.com/docs/DOC-53956
Hortonworks Data Platform	http://hortonworks.com/
Galera Cluster for MySQL	http://galeracluster.com/
Real-Time Global Anomaly Detection in IoT with EMC ECS	http://bit.ly/1QcYwst
Hive MetaTool	https://cwiki.apache.org/confluence/display/Hive/Hive+MetaTool
Hive Transactions	https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions
Spark Documentation	https://spark.apache.org/docs/1.5.2/
Hive 0.14 Benchmarking	http://hortonworks.com/blog/cost-based-optimizer-makes-apache-hive-0-14-more-than-2-5x-faster/
TPC-DS	http://www.tpc.org/tpcds/

DOCUMENT VERSION HISTORY

Table 8. Document Version History

VERSION #	REVISION DATE	REVISION AUTHOR	REVISION DETAIL
1.0	12/9/2016	Claudio Fahey	Initial document
1.1	1/12/2017	Claudio Fahey	Added procedure for relocating HDFS to an ECS bucket. Various clean up.

APPENDIX LIST

Table 9. Appendix List

ID	TITLE
1	Moving An Existing MySQL Database To Galera Cluster
2	Hive MetaTool
3	Functional Testing Procedure

APPENDIX 1 - MOVING AN EXISTING MYSQL DATABASE TO GALERA CLUSTER

To move the Hive Metastore from an existing MySQL database to a new MySQL database (e.g. Galera Cluster for MySQL), the steps below can be used.

Create a backup of the database.

```
[root@site1-master-0 ~]# mysqldump hive > hive.sql
```

Transfer the backup file hive.sql to the target system (not shown). Then restore it.

```
[root@mydbcluster-0 ~]# mysql --password
mysql>
create database hive;
CREATE USER 'hive'@'%' IDENTIFIED BY 'mypassword';
GRANT ALL PRIVILEGES ON hive.* TO 'hive'@'%' WITH GRANT OPTION;
exit;
[root@mydbcluster-0 ~]# mysql --password mypassword < hive.sql
```

Test connectivity from a Hadoop node.

```
[root@site1-master-0 ~]# mysql --host mydbcluster-0.example.com --user=hive --password=mypassword
```

APPENDIX 2 - HIVE METATOOL

Hive MetaTool can be used to make bulk changes to the Hive Metastore. For instance, it can be used to change the location of tables and databases from HDFS to ECS (viprfs). Refer to https://cwiki.apache.org/confluence/display/Hive/Hive+MetaTool for details.

Before using Hive Tool, the file /etc/hive/conf/hive-site.xml needs to contain the Hive Metastore database connection password. Add the following to this file before using Hive Tool and remove it when done so that the plain text password is not unnecessarily exposed.

```
<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>mypassword</value>
</property>
```

The following command will list all file systems referenced in the Hive Metastore.

```
[root@site1-master-0 ~]# hive --service metatool -listFSRoot
```

The next command will change the location of tables and databases from HDFS to ECS (viprfs). Note that DistCp could be used prior to this to copy the actual data files from HDFS to ECS.

```
hive --service metatool -updateLocation viprfs://hive-warehouse-1.ns1.federation1 hdfs://site1-master-1.example.com:8020
```

APPENDIX 3 - FUNCTIONAL TESTING PROCEDURE

OVERVIEW

To begin the functional testing, start with all systems online and functioning normally. This is referred to as the steady state. In this state, perform the steps in the Standard Test Suite. Next, simulate a network outage and perform the steps in the Standard Test Suite. This will be repeated for various failure modes and recovery from them.

ENVIRONMENT

The environment should be configured as described in the *Testing Environment* section of this guide. Ensure that the ECS bucket is replicated to all three sites and that Access During Outage is enabled.

STANDARD TEST SUITE

This section describes the standard suite of tests that will be performed when the system is in various states (e.g. steady state, network outage, power failure, etc.).

HIVE - ONE-TIME CONFIGURATION

The steps in this section only need to be performed once after initial installation. The steps will generate a small sample dataset and some Hive metadata.

Site 1 - [ambari-qa@site1-master-0 ~]$
Set an environment variable to the URI of the ECS bucket to use. This allows the following steps to be typed as-is. In this step, you must specify the URI that is specific to your system.

```
FS=viprfs://hive-warehouse-1.ns1.federation1
```

Create a small CSV file.

```
cat - > tab1.csv
1,true,123.123,2012-10-24 08:55:00
2,false,1243.5,2012-10-25 13:40:00
3,false,24453.325,2008-08-22 09:33:21.123
4,false,243423.325,2007-05-16 22:32:21.33454
5,true,243.325,1953-04-22 09:11:33
```

Type <Control+D>.

Upload the file to ECS.

```
hadoop fs -mkdir -p $FS/user/ambari-qa/sample_data/tab1
hadoop fs -put -f tab1.csv $FS/user/ambari-qa/sample_data/tab1
hadoop fs -ls $FS/user/ambari-qa/sample_data/tab1
```

Start Hive. The configuration parameter FS is defined to refer to the URI to the ECS bucket to use.

```
hive -hiveconf FS=$FS
```

Create the Hive database *db1*. The default location for new tables created in this database is explicitly set. Otherwise, Hive will use the default Hive warehouse directory.

```
CREATE DATABASE DB1 LOCATION '${hiveconf:FS}/apps/hive/warehouse/db1.db';
USE DB1;
```

Create an external table *tab1*.

```
DROP TABLE IF EXISTS TAB1;
CREATE EXTERNAL TABLE TAB1
(
 ID INT,
 COL_1 BOOLEAN,
 COL_2 DOUBLE,
 COL_3 TIMESTAMP
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '${hiveconf:FS}/user/ambari-qa/sample_data/tab1';
SHOW CREATE TABLE TAB1;
SELECT * FROM TAB1;
```

Create a partitioned table *ptab2*.

```
DROP TABLE IF EXISTS PTAB2;
CREATE TABLE PTAB2
(
 ID INT,
 COL_1 BOOLEAN,
 COL_2 DOUBLE,
 COL_3 TIMESTAMP
```

```
)
PARTITIONED BY(PART INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

Create a partitioned table *acidptab1* that supports ACID transactions.

```
DROP TABLE IF EXISTS ACIDPTAB1;
CREATE TABLE ACIDPTAB1
(
 ID INT,
 COL_1 BOOLEAN,
 COL_2 DOUBLE,
 COL_3 TIMESTAMP
)
PARTITIONED BY(PART INT)
CLUSTERED BY (ID) INTO 5 BUCKETS
STORED AS ORC TBLPROPERTIES ('transactional'='true');
```

FILE SYSTEM

Sites 1, 2, and 3 - - [hdfs@siteX-master-0 ~]$

Set an environment variable to the URI of the ECS bucket to use. This allows the following steps to be typed as-is. In this step, you must specify the URI that is specific to your system. Repeat on each site.

```
FS=viprfs://hive-warehouse-1.ns1.federation1
```

List the root directory of the ECS bucket. Repeat on each site.

```
date -u ; time hadoop fs -ls $FS/
```

Create a new empty file. In below commands, replace ***xx*** with a unique number to ensure that a new file is created. Replace ***n*** with the site number. Repeat on each site.

```
date -u ; time hadoop fs -touchz $FS/EMPTY_FROM_SITEn_xx
```

Delete a file. In below commands, replace ***xx*** with the same unique number used above. Replace ***n*** with the site number. Repeat on each site.

```
date -u ; hadoop fs -rm -skipTrash $FS/EMPTY_FROM_SITEn_xx
```

Update the timestamp (touch) an existing file. Run below commands at least twice to test file overwrite. Replace ***n*** with the site number. Repeat on each site.

```
date -u ; time hadoop fs -touchz $FS/EMPTY_FROM_SITEn
```

Create a new file whose contents contain a timestamp, then read it. In below commands, replace ***xx*** with a unique number to ensure that a new file is created.

Site 1:

```
date -u | hadoop fs -put - $FS/FROM_SITE1_xx ; hadoop fs -cat $FS/FROM_SITE1_xx
```

Site 2 and 3:

```
date -u ; hadoop fs -cat $FS/FROM_SITE1_xx
```

Create a new file whose contents contain a timestamp, then list it. In below commands, replace ***xx*** with a unique number to ensure that a new file is created.

Site 1:

```
date -u | hadoop fs -put - $FS/FROM_SITE1_xx ; date -u ; hadoop fs -ls $FS/FROM_SITE1_xx
```

Ensure that the new file is listed.

Site 2 and 3:

```
date -u ; hadoop fs -ls $FS/FROM_SITE1_xx
```

Ensure that the new file is listed.

Overwrite an existing file, then read it. The put command below will first delete the target file and then write a new one with the same name.

Site 1:

```
date -u | hadoop fs -put -f - $FS/FROM_SITE1 ; hadoop fs -cat $FS/FROM_SITE1
```

Site 2 and 3:

```
date -u ; hadoop fs -cat $FS/FROM_SITE1
```

Repeat above with Site 1 and 2 swapped.

Repeat above with Site 1 and 3 swapped.

HIVE - BASIC TABLE OPERATIONS

Site 1 - [root@mydbcluster-0 ~]$

Check the status of the Galera Cluster.

```
mysql --password
SHOW STATUS LIKE 'wsrep_cluster_status';
SHOW STATUS LIKE 'wsrep_cluster_size';
```

Site 1 - [ambari-qa@site1-master-0 ~]$

Start Hive.

```
hive
```

Create a new table using a select statement.

```
SHOW DATABASES;
USE DB1;
SHOW TABLES;
DROP TABLE TAB3;
CREATE TABLE TAB3 AS SELECT ID, COL_1, COL_2, MONTH(COL_3), DAYOFMONTH(COL_3) FROM TAB1 WHERE YEAR(COL_3) =
2012;
SELECT * FROM TAB3;
```

Execute a query that will run as a Tez job.
`SELECT COUNT(ID) FROM TAB3;`
Site 2 - [ambari-qa@site2-master-0 ~]$
Start Hive.
`hive`
View the contents of the table. Ensure that the contents shown match the contents shown in the previous step.

`USE DB1;`
`SHOW TABLES;`
`SELECT * FROM TAB3;`

Site 3 - [ambari-qa@site3-master-0 ~]$
Start Hive.
`hive`
View the contents of the table. Ensure that the contents shown match the contents shown in the previous step.

`USE DB1;`
`SHOW TABLES;`
`SELECT * FROM TAB3;`

Repeat the steps in this section but with Site 1 and Site 2 swapped.
Repeat the steps in this section but with Site 1 and Site 3 swapped.

HIVE - PARTITIONED TABLE OPERATIONS

Site 1 - [ambari-qa@site1-master-0 ~]$ - hive>
Load into a single partition of a partitioned table.

`USE DB1;`
`INSERT OVERWRITE TABLE PTAB2 PARTITION (PART=1) SELECT ID, COL_1, COL_2, CURRENT_TIMESTAMP FROM TAB1 WHERE ID=1;`
`SELECT * FROM PTAB2;`

Site 2 - [ambari-qa@site2-master-0 ~]$ - hive>
View the contents of the table. Ensure that the contents shown match the contents shown in the previous step.
`USE DB1;` `SELECT * FROM PTAB2;`
Site 3 - [ambari-qa@site3-master-0 ~]$ - hive>
View the contents of the table. Ensure that the contents shown match the contents shown in the previous step.
`USE DB1;`

```
SELECT * FROM PTAB2;
```

Site 2 - [ambari-qa@site2-master-0 ~]$ - hive>

Load into a single partition of a partitioned table.

```
USE DB1;
INSERT OVERWRITE TABLE PTAB2 PARTITION (PART=2) SELECT ID, COL_1, COL_2, CURRENT_TIMESTAMP FROM TAB1 WHERE ID=2;
SELECT * FROM PTAB2;
```

Site 1 - [ambari-qa@site1-master-0 ~]$ - hive>

View the contents of the table. Ensure that the contents shown match the contents shown in the previous step.

```
USE DB1;
SELECT * FROM PTAB2;
```

Site 3 - [ambari-qa@site3-master-0 ~]$ - hive>

View the contents of the table. Ensure that the contents shown match the contents shown in the previous step.

```
USE DB1;
SELECT * FROM PTAB2;
```

Site 3 - [ambari-qa@site3-master-0 ~]$ - hive>

Load into a single partition of a partitioned table.

```
USE DB1;
INSERT OVERWRITE TABLE PTAB2 PARTITION (PART=3) SELECT ID, COL_1, COL_2, CURRENT_TIMESTAMP FROM TAB1 WHERE ID=3;
SELECT * FROM PTAB2;
```

Site 1 - [ambari-qa@site1-master-0 ~]$ - hive>

View the contents of the table. Ensure that the contents shown match the contents shown in the previous step.

```
USE DB1;
SELECT * FROM PTAB2;
```

Site 2 - [ambari-qa@site2-master-0 ~]$ - hive>

View the contents of the table. Ensure that the contents shown match the contents shown in the previous step.

```
USE DB1;
SELECT * FROM PTAB2;
```

Sites 1, 2 and 3 - hive>

In the previous steps, only one query was executed at a time. In this step, execute the same insert queries simultaneously from each site. Then view the contents of the table from each site. Ensure that the contents shown match the contents shown in the previous step.

Site 1 - [ambari-qa@site1-master-0 ~]$ - hive>

Insert records into the transactional table from a select query. Then view the table contents. Note that the column *col_2* will have the timestamp at which it was created or updated.

```
USE DB1;
INSERT INTO ACIDPTAB1 PARTITION (PART=1) SELECT ID, COL_1, COL_2, CURRENT_TIMESTAMP FROM TAB1 WHERE ID IN
(1,2);
SELECT * FROM ACIDPTAB1;
```

Execute a query that will run as a Tez job.

```
SELECT COUNT(ID) FROM ACIDPTAB1;
```

Site 2 - [ambari-qa@site2-master-0 ~]$ - hive>

View the table contents. Insert records into the transactional table from a select query. Then view the table contents again.

```
USE DB1;
SELECT * FROM ACIDPTAB1;
INSERT INTO ACIDPTAB1 PARTITION (PART=2) SELECT ID, COL_1, COL_2, CURRENT_TIMESTAMP FROM TAB1 WHERE ID=2;
SELECT * FROM ACIDPTAB1;
```

Execute a query that will run as a Tez job.

```
SELECT COUNT(ID) FROM ACIDPTAB1;
```

Site 3 - [ambari-qa@site3-master-0 ~]$ - hive>

View the table contents. Insert records into the transactional table from a select query. Then view the table contents again.

```
USE DB1;
SELECT * FROM ACIDPTAB1;
INSERT INTO ACIDPTAB1 PARTITION (PART=3) SELECT ID, COL_1, COL_2, CURRENT_TIMESTAMP FROM TAB1 WHERE ID>2;
SELECT * FROM ACIDPTAB1;
```

Delete and update records.

```
DELETE FROM ACIDPTAB1 WHERE PART=1 AND ID=1;
UPDATE ACIDPTAB1 SET COL_3=CURRENT_TIMESTAMP WHERE PART=3 AND ID=5;
SELECT * FROM ACIDPTAB1;
```

Site 1 - [ambari-qa@site1-master-0 ~]$ - hive>

View the table contents.

```
SELECT * FROM ACIDPTAB1;
```

Site 2 - [ambari-qa@site2-master-0 ~]$

View the Hive warehouse directory corresponding to the *acidptab1* table.

```
hadoop fs -ls -R $FS/apps/hive/warehouse/db1.db/acidptab1
```

Site 2 - [ambari-qa@site2-master-0 ~]$ - hive>

Initiate a minor table compaction and then monitor it.

```
USE DB1;
ALTER TABLE ACIDPTAB1 PARTITION (PART=2) COMPACT 'MINOR';
SHOW COMPACTIONS;
```

Site 1 - [ambari-qa@site1-master-0 ~]$ - hive>

Monitor the status of table compaction from another site.

```
SHOW COMPACTIONS;
```

Initiate major compactions of the other partitions.

```
ALTER TABLE ACIDPTAB1 PARTITION (PART=1) COMPACT 'MAJOR';
ALTER TABLE ACIDPTAB1 PARTITION (PART=3) COMPACT 'MAJOR';
SHOW COMPACTIONS;
```

Site 1 - [ambari-qa@site2-master-0 ~]$

View the Hive warehouse directory corresponding to the *acidptab1*. Notice that the number of files and directories has been reduced as a result of the compactions.

```
hadoop fs -ls -R $FS/apps/hive/warehouse/db1.db/acidptab1
```

Sites 1, 2, and 3

Run the insert queries concurrently from 2 sites into the same partition. The commands below will run each query 100 times from each site. Any errors will cause the execution to stop. The locks can be monitored during the execution.

Site 1:

```
rm insert1.sql
for i in {1..100} ; do echo "select $i, current_timestamp; insert into acidptab1 partition (part=10) select
id, col_1, col_2, current_timestamp from tab1 where id=1;" >> insert1.sql ; done
hive --database db1 -v ON_ERROR_STOP=1 -f insert1.sql ; date -u
```

Site 2:

```
rm insert2.sql
for i in {1..100} ; do echo "select $i, current_timestamp; insert into acidptab1 partition (part=10) select
id, col_1, col_2, current_timestamp from tab1 where id=2;" >> insert2.sql ; done
hive --database db1 -v ON_ERROR_STOP=1 -f insert2.sql ; date -u
```

Site 3:

```
watch 'hive -e "show locks; show transactions; show compactions;"'
```

View the partition contents.

```
SELECT * FROM ACIDPTAB1 WHERE PART=10;
```

Run the queries below concurrently.

Site 1:
`UPDATE ACIDPTAB1 SET COL_3=CURRENT_TIMESTAMP, COL_2=1 WHERE PART=10 AND ID=1;`
Site 2:
`UPDATE ACIDPTAB1 SET COL_3=CURRENT_TIMESTAMP, COL_2=2 WHERE PART=10 AND ID=1;`
Site 3:
`SHOW LOCKS; SHOW TRANSACTIONS;`

View the partition contents.
`SELECT * FROM ACIDPTAB1 WHERE PART=10;`

STEADY STATE

This will be the baseline when all systems are online and functioning normally. Perform all tests in the Standard Test Suite. All tests should succeed in this state.

TEMPORARY WAN LINK FAILURE

For these tests, disconnect the network link that connects site 3 to the other two sites. If using a WAN emulator, this can be simulated by setting 100% packet loss between site 1 and site 3.

With the WAN link disconnected, perform all tests in the Standard Test Suite. It is expected that the command to list the root directory of the bucket "hadoop fs -ls $FS/" may fail for the first 15 minutes after the WAN link is disconnected. Such a failure will result in the error message "ViPRFS internal error (ERROR)". Repeat this command until it succeeds on sites 1 and 2. Note that site 3 will continue to encounter this error. Continue with the other tests in the Standard Test Suite. All read-only tests are expected to succeed on sites 1 and 2. Tests that attempt to write to the ECS bucket are expected to fail.

Reconnect the WAN link. Perform all tests in the Standard Test Suite. All tests should succeed.

TEMPORARY SITE POWER FAILURE

For these tests, power off all ECS nodes, Hadoop nodes, and the Hive Metastore database node at site 3.

Although ECS has a graceful shutdown process documented in the ECS Administrator's Guide, a more realistic a way of simulating a site-wide power failure is to immediately cause a power loss. Of course, this is not recommended in a production environment. This can be performed quickly on all ECS nodes using the command below. First, create a file named bmc.txt that contains IP addresses or FQDNs of the BMC NICs associated with each ECS node.

```
date -u ; cat bmc.txt | xargs -i -n 1 ipmitool -I lanplus -H {} -U root -P passwd power off
```

With the power off at site 3, perform all tests in the Standard Test Suite. Like the WAN link failure case, it is expected that the command to list the root directory of the bucket "hadoop fs -ls $FS/" may fail for the first 15 minutes after the WAN link is disconnected. Such a failure will result in the error message "ViPRFS internal error (ERROR)". Repeat this command until it succeeds on sites 1 and 2. Continue with the other tests in the Standard Test Suite. All read-only tests are expected to succeed on sites 1 and 2. Tests that attempt to write to the ECS bucket are expected to fail.

Restore power to site 3. The following command can power on the ECS nodes.

```
date -u ; cat bmc.txt | xargs -i -n 1 ipmitool -I lanplus -H {} -U root -P passwd power on
```

Once all systems have booted up completely, perform all tests in the Standard Test Suite. All tests should succeed.

TEMPORARY NODE POWER FAILURE

For these tests, power off ECS node 1 in site 3 to simulate a failure of a single node. Perform all tests in the Standard Test Suite. All tests should succeed in this state. However, I/O to ECS may be delayed or produce errors during the first few minutes after the node failure. Also, warnings such as "WARN vipr.ClientConnection: Failed to connect to '10.246.22.151'" may be seen but will not impact the I/O result.

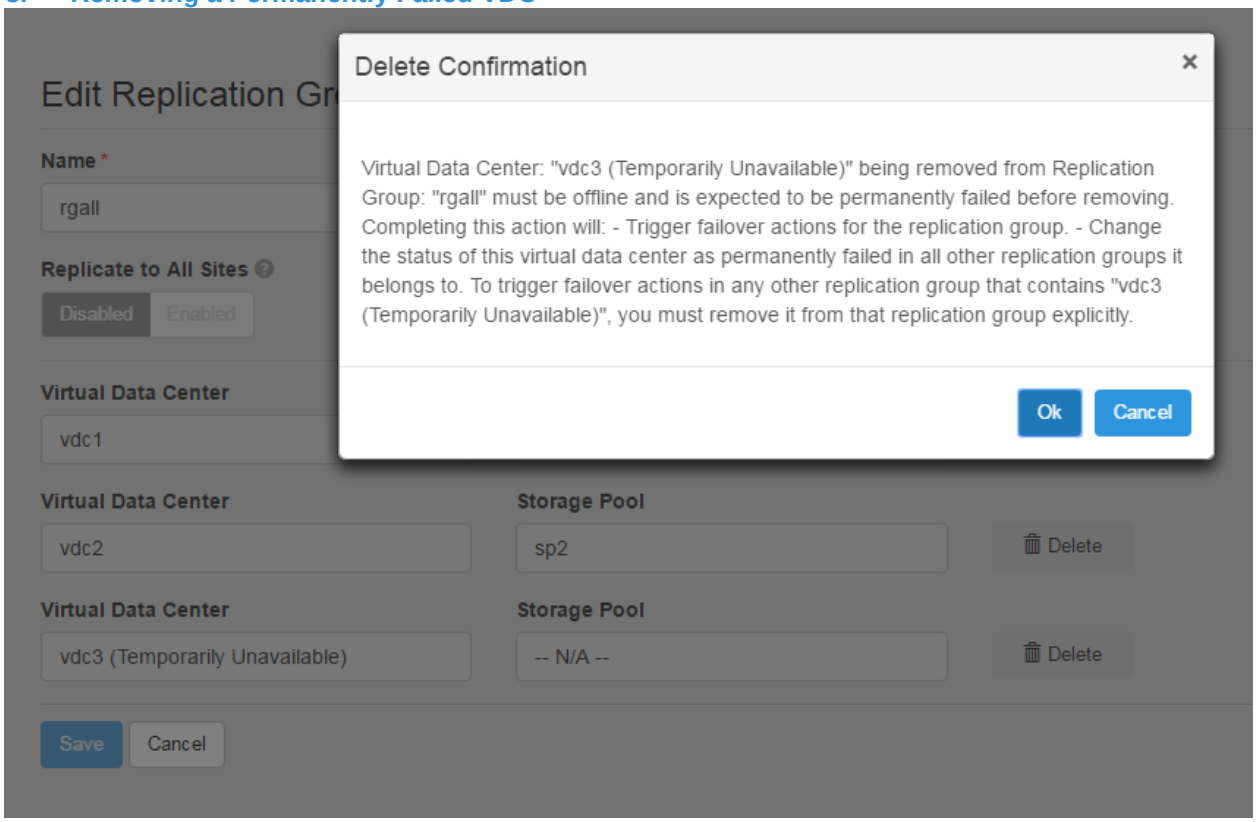
Restore power to ECS node 1 in site 3. Perform all tests in the Standard Test Suite. All tests should succeed in this state.

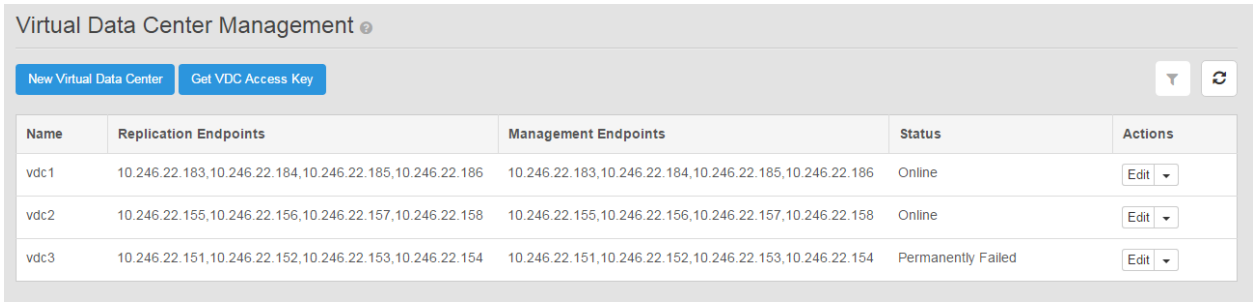
PERMANENT SITE FAILURE

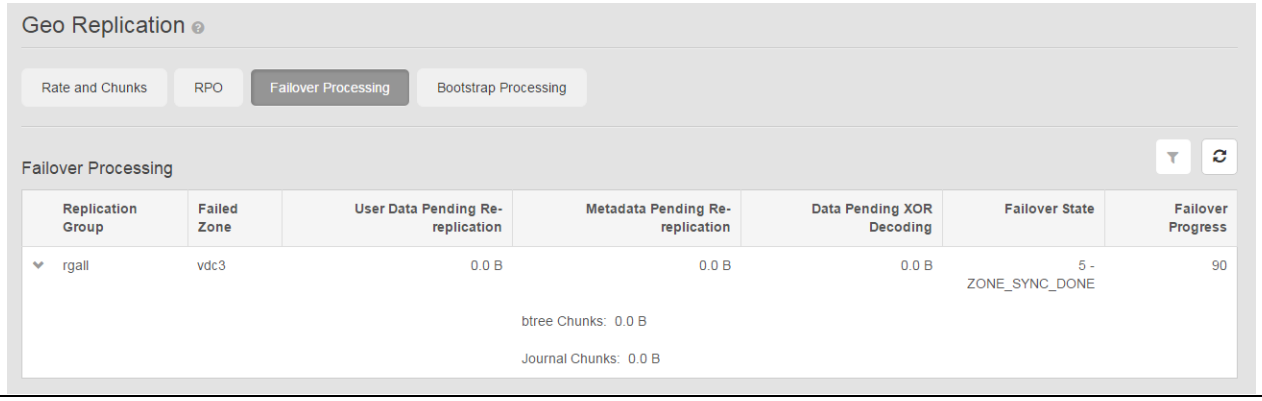
For these tests, power off all ECS nodes, Hadoop nodes, and the Hive Metastore database node at site 3, just like for the Temporary Site Power Failure tests. This time however, a permanent failure will be simulated by keeping site 3 offline.

When TSO detects that site 3 is offline, TSO (temporary site outage) will be triggered which will provide read-only access to the replicated buckets. If the failure is permanent, for instance in the case that a natural disaster physically destroyed the data center, we'll want to force the bucket to be read-write, even though there is a potential that some data recently updated from site 3 did not get replicated to sites 1 and 2. This is done in the ECS administrative UI by removing the failed VDC from the replication group. Refer to the ECS Administrator's Guide for details.

Figure 8. Removing a Permanently Failed VDC







These screens show the process of removing a permanently failed VDC from a replication group.

After the VDC has been removed from the replication group, perform all tests in the Standard Test Suite. All tests should succeed.

To bring back site 3 (e.g. to simulate installing new equipment in site 3), first power on all systems. Perform a new installation of ECS software on the ECS nodes. This will delete all data on the ECS nodes at this site. Of course, the data in the bucket will remain safe in

sites 1 and 2. Then add the ECS cluster in site 3 to the ECS federation in sites 1 and 2. Finally, add the new VDC to the appropriate replication groups to begin re-replication of the buckets.