How-To Library

Informatica

Strategies for Incremental Updates on Hive in Big Data Management 10.2.1

Abstract

This article describes alternative solutions to the Update Strategy transformation for updating Hive tables to support incremental loads. These solutions include updating Hive tables using the Update Strategy transformation, Update Strategy transformation with the MERGE statement, partition merge solution, and key-value stores.

Supported Versions

- Informatica Big Data Management 10.2.1

Table of Contents

Overview

Many organizations want to create data lakes and enterprise data warehouses on Hadoop clusters to perform near real-time analytics based on business requirements. Building data lakes on a Hadoop cluster requires a one-time initial load from legacy warehouse systems and frequent incremental loads. In most cases, Hive is the preferred analytic store.

Although Hive versions 0.13 and later support transactions, they pose challenges with incremental loads, such as limited ACID compliance and requirements for ORC file formats and bucketed tables.

This article describes various strategies for updating Hive tables to support incremental loads and ensuring that targets are in sync with source systems.

Informatica Big Data Management supports the following methods to perform incremental updates:

- Update Strategy transformation
- Update Strategy transformation using MERGE statement
- Updates using the partition merge solution
- Updates using key-value stores

Solution Recommendations

To optimize updating Hive tables to support incremental loads:

Update Strategy transformation

Use the Update Strategy transformation on all Hadoop distributions except Cloudera and Amazon EMR. On Cloudera and Amazon EMR distributions, when an Update Strategy transformation is not a practical option, you can use the partition merge solution or key-value stores to perform updates.

Update Strategy transformation using MERGE statement

Use the Update Strategy transformation with the MERGE statement on all Hadoop distributions except Cloudera and Amazon EMR. On Cloudera and Amazon EMR distributions, you can use the partition merge solution or key-value stores to perform updates. The Blaze and Hive engines do not support Update Strategy transformation using the MERGE statement.

Key-value stores

Use key-value stores only if the cluster is already configured to use the database services and the data volume is less than 200 GB. Key-value stores such as HBase are NoSQL databases. These databases require Apache Phoenix or Hive tables with HBase storage handlers to provide relational (/SQL) database views. HBase is also CPU, memory, and I/O intensive.

Partition merge

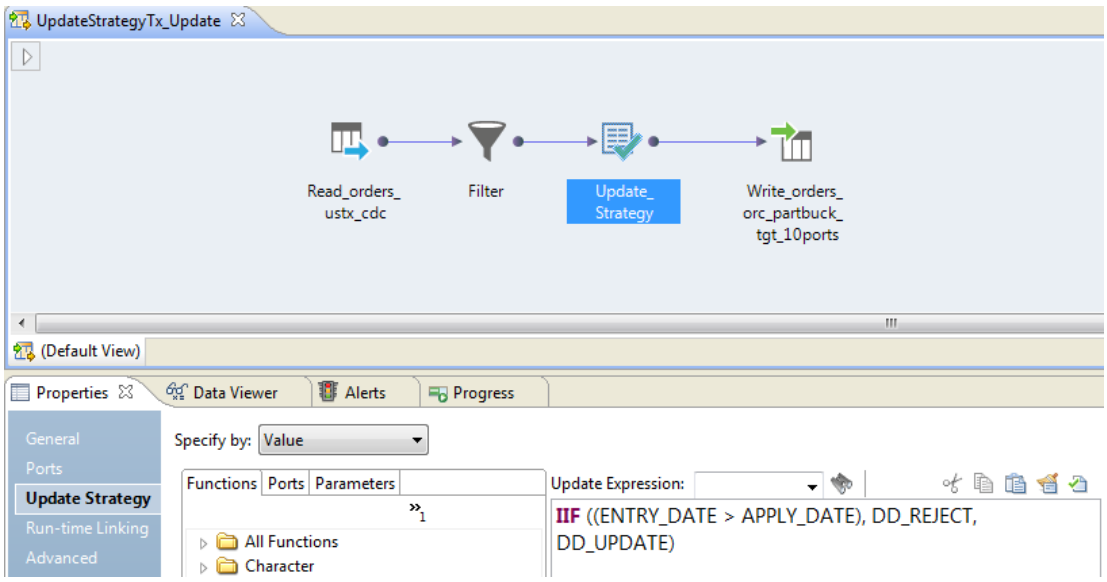
Use the partition merge solution if you understand the underlying data distribution. You must choose the correct partitioning strategy so that the data is uniformly distributed across partitions with approximately 5-10 GB of data per partition. The partition merge solution scales well for large datasets.

Approach 1. Update Strategy Transformation

You can use an Update Strategy transformation to update Hive ACID tables. You can define expressions in an Update Strategy transformation with IIF or DECODE functions to set rules for updating rows. For example, the following IIF function detects and marks a row for reject if the entry date is after the apply date. Otherwise, the function marks the row for update:

```
IIF ((ENTRY_DATE > APPLY_DATE), DD_REJECT, DD_UPDATE)
```

The following image shows the Update Strategy transformation with an IIF function:



When using the Update Strategy transformation, the following restrictions apply to Cloudera and Amazon EMR distributions:

- Cloudera CDH. Discourages using ORC file format, which is a prerequisite for Hive transactions.
- Amazon EMR. Due to the limitation in HIVE-17221, the Update Strategy transformation on transaction enabled partitioned Hive tables fails.

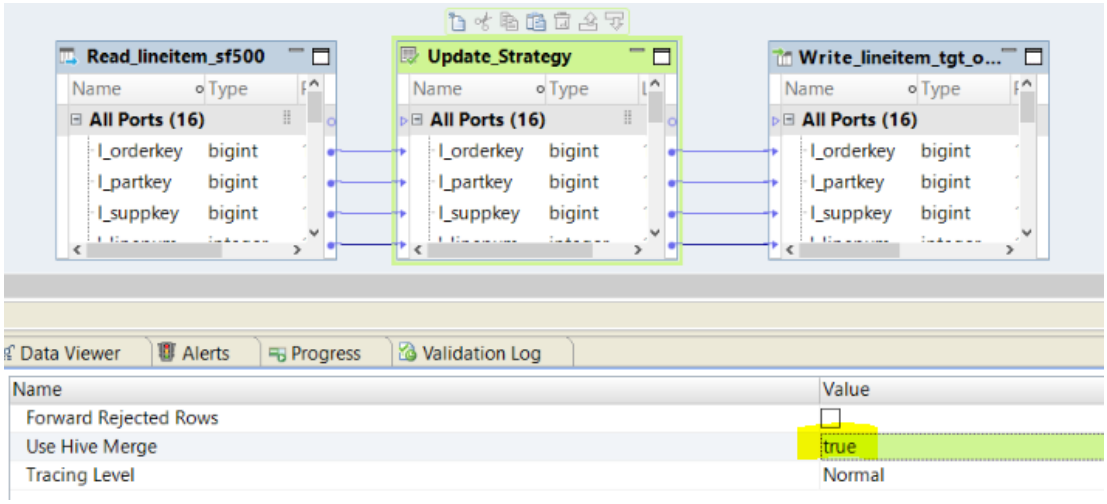
Therefore, you must use an alternative strategy to update Hive tables without enabling Hive transactions.

For more information about the Update Strategy transformation, see the *Informatica 10.2.1 Big Data Management User Guide*.

Approach 2. Update Strategy Transformation Using MERGE Statement

The Update strategy transformation supports MERGE statements for ACID enabled tables. The MERGE statement performs insert, update, and delete operations. The MERGE statement can generate efficient HiveQL and provide overall improved performance.

The MERGE statement is controlled by the Use Hive Merge property in the Update strategy transformation, as shown in the following image:



The Use Hive Merge property can be configured to use the MERGE statement or use the Traditional Update Strategy transformation approach.

If you enable the Hive Merge property, with technical restrictions present, Approach 1 is used by default.

The following restrictions apply to the MERGE statement:

- The Blaze and Hive engines do not support Update Strategy transformation using the MERGE statement.
- ORC file format is a prerequisite for Hive transactions and should not be used with Cloudera CDH. Use an alternative strategy to update Hive tables without enabling Hive transactions.
- Due to the Amazon EMR limitation in HIVE-17221, the Update Strategy transformation fails on transaction enabled partitioned Hive tables.

Approach 3. Partition Merge

The partition merge solution detects, stages, and updates only the impacted partitions based on the incremental records from upstream systems.

To develop a mapping using the partition merge solution, you use the following types of input sources:

Incremental table

Holds the incremental change records from any upstream system that indicate whether a row is marked for insert, update, or delete. Incremental tables must have an identifier in the form of an expression or a column that indicates whether the incoming row is marked for insert, update, or delete.

Historic base table

A target table that initially holds all records from the source system. After the initial processing cycle, it maintains a copy of the most up-to-date synchronized record set from the source.

Temporary insert base table

A temporary target table that holds all insert records. The insert records are eventually loaded into the historic base table.

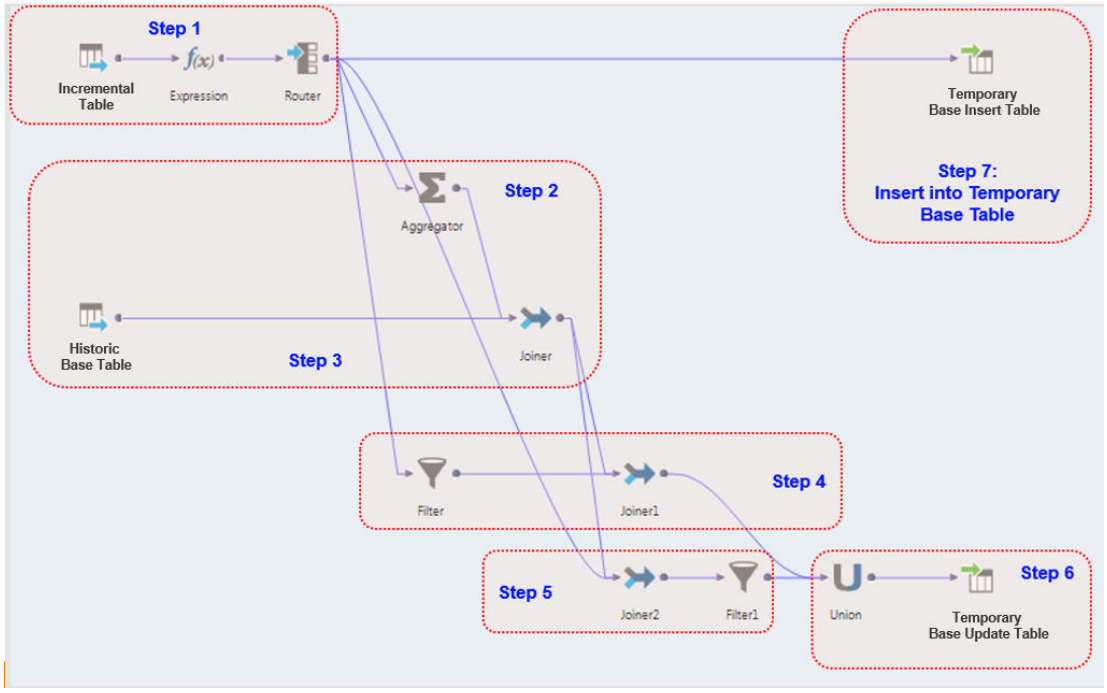
Temporary update base table

A temporary target table that holds all update records after the rows marked for delete have been removed. The update records are eventually loaded into the historic base table.

Create a Partition Merge

To implement the partition merge solution, you create three mappings. The first mapping has two branches. One branch writes inserts to the temporary base insert table. The other branch processes the update and delete operations separately, and merges with a Union transformation before writing to the temporary base update table. The second two mappings are pass-through mappings that insert the temporary tables into the historic base table.

The following image is a graphical representation of the partition merge solution:



Perform the following steps to implement the partition merge solution:

1. Define a Router transformation to separate insert operations from update and delete operations. Create one pipeline to process the inserts, and one pipeline to process the updates and deletes.
2. Define an Aggregator transformation to group the rows marked for update and delete based on the partitioned column and detect the impacted partitions.

3. Define a Joiner transformation to join the Aggregator output with the historic base table on the partitioned column and fetch rows only for those impacted partitions.
4. Define a Filter transformation to filter out rows marked for update. Join them with the impacted partitions to get all updatable records.
5. Use a detail outer join and filter NULL keys to perform a minus operation on the impacted partitions. The minus operation eliminates rows marked for delete and rows from the historic base table that were updated.
6. Define a Union transformation to merge the resulting rows from Step 4 and Step 5 and insert them into the temporary base update table. Enable the **Truncate Hive Target Partition** option on the temporary update base table.
7. Add the insert rows into the temporary update base table.

   Note: Use a workflow to ensure that Step 6 runs before Step 7. Hadoop mappings do not support target load order constraints.
8. Create a pass-through mapping to load the temporary insert base table into the historic base table.
9. Create a pass-through mapping to load the temporary update base table into the historic base table.

Tune the Partition Merge Solution for Optimal Performance

Use the following optimization techniques to get the best performance results based on your requirements:

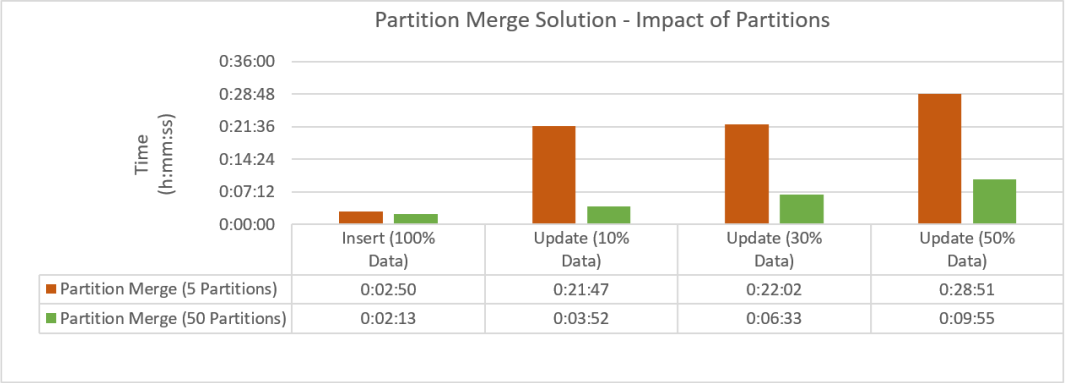
- Do not require Hive transactions to be enabled, and ensure that ACID transactions are disabled. Inserts into transaction enabled Hive tables are approximately three to five times slower than inserts into transaction disabled Hive tables.
- Configure partitions so that each partition has approximately 5-10 GB of data. The partition merge solution performs updates by identifying the impacted partitions. These updates are efficient when the data is well distributed across many partitions.

Case Study: Partition Merge (On-Premise Test Setup)

It is important to distribute data across multiple partitions. Based on Informatica internal testing, updates to Hive tables with 50 partitions perform three to five times faster than updates to tables with five partitions.

Cloudera Hadoop Distribution (CDH5U11)

Chipset: Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz
CPU: 2 x 12 Cores
RAM: 128 GB
OS: RedHat Linux 7.3
Number of Data Nodes: 14
Dataset: TPC-H (Orders Table, Scale Factor SF1000, ~170 GB of data)



	Insert (100% Data)	Update (10% Data)	Update (30% Data)	Update (50% Data)
Partition Merge (5 Partitions)	0:02:50	0:21:47	0:22:02	0:28:51
Partition Merge (50 Partitions)	0:02:13	0:03:52	0:06:33	0:09:55

Approach 4. Key-Value Stores

Inserts into key-value stores are treated as upserts, so they are a convenient choice for handling updates.

This article uses HBase as the primary key-value store and restricts the discussion around it. HBase is a non-relational (NoSQL) database that runs on top of HDFS and performs random reads and writes against large datasets.

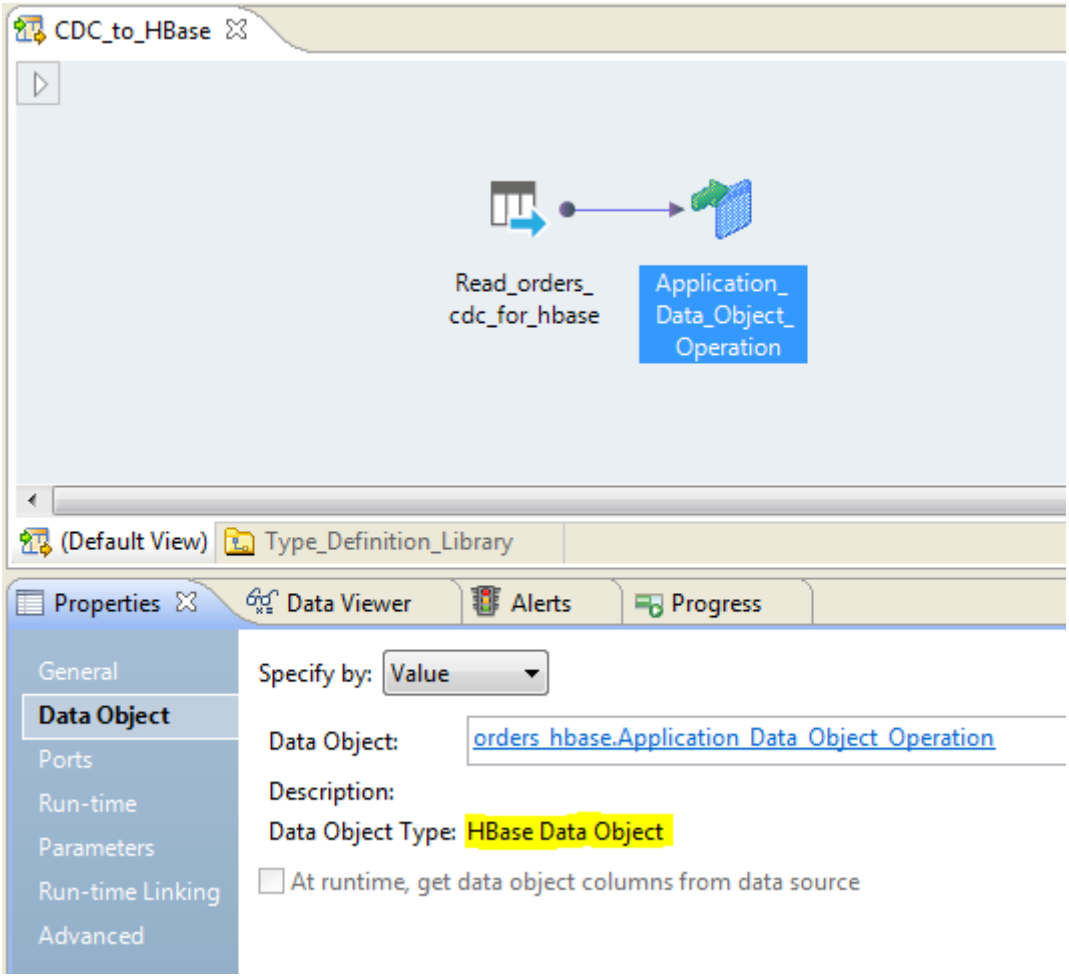
Unlike MapReduce which is mostly I/O bound, HBase is both CPU and memory intensive with sporadic, large, and sequential input and output operations. This advantage comes at a cost in the form of additional system resource requirements on the cluster nodes.

HBase handles deletes by marking rows as "deleted" and removing the data during compaction.

HBase tables are split into chunks of rows called "regions" which are distributed across the cluster and managed by the RegionServer process. By default, HBase allocates only one region per table. Thus, during initial load, all requests go to a single region server regardless of the number of available region servers on the cluster.

Informatica Big Data Management provides native HBase connectors to write to HBase. Create a pass-through mapping with incremental records as the source and an HBase table as the target.

The following image shows an HBase source data object and data object operation:



HBase is a key-value store. For relational (/SQL) database views, either use Apache Phoenix or use HBase storage handlers to create a Hive table.

The following example shows a Hive table created with HBase storage handlers:

```
create table orders(
        rowkey STRING,
        orderKey STRING,
        custKey STRING,
        orderStatus STRING,
        totalPrice DOUBLE
)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties
("hbase.columns.mapping" = "cf: orderKey, cf:custKey,
cf: orderStatus, cf: totalPrice)
tblproperties ("hbase.table.name" = "orders_hbase");
```

Tune HBase Tables for Optimal Performance

You can pre-split HBase tables to ensure even distribution throughout the cluster. With pre-splitting, you can create a table with many regions by supplying the split points at the table creation time. Because pre-splitting ensures that the initial load is more evenly distributed throughout the cluster, you should always consider using it if you know your key distribution.

You can use predefined algorithms like HexStringSplit and UniformSplit, or implement a custom SplitAlgorithm.

For example, the following command pre-splits HBase tables using HexStringSplit:

```
hbase org.apache.hadoop.hbase.util.RegionSplitter orders_hbase HexStringSplit -c 32 -f cf
```

Where:

- *-c* is number of splits.
- *-f* is the column family name.

When you use HexStringSplit, define **rowkey** as **Strings**.

Determining the number of splits depends on several factors, including the type of data mapped to the rowkey, the data volume, and the amount of resources available for region servers across the cluster.

Consider allocating approximately 20 splits for every gigabyte of data.

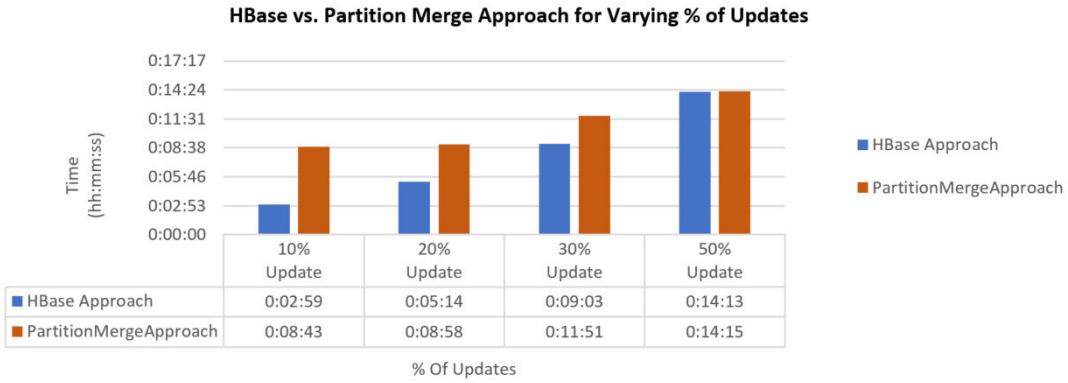
Case Study: Partition Merge vs. Key-Value Stores (EMR Test Setup)

The following case study uses HBase as the key-value store.

EMR Hadoop Distribution (version 5.4)

EMR Instance Type:
m3.2xlarge (16 vCPU & 30 GB)

Number of Core Nodes: 4
Dataset: TPC-H (Orders Table, Scale Factor SF100, ~17 GB of data.



Strategy Comparison

The following table compares the different strategies:

Strategy	Advantages	Disadvantages
Update Strategy Transformation	- It is simple and easy to implement.	- Cloudera discourages using ORC format and Hive ACID properties. Both are prerequisites for the Update Strategy transformation. - On Amazon EMR 5.4, Update Strategy is not supported for partitioned, transaction enabled Hive tables due to a Hive bug.
Update Strategy Transformation Using MERGE Statement	- The MERGE statement can generate efficient HiveQL and provide overall improved performance.	- The Blaze and Hive engines do not support Update Strategy transformation using the MERGE statement. - With Cloudera CDH, it is not recommended to use ORC file format. ORC file format is a prerequisite for Hive transactions. You must use an alternative strategy to update Hive tables without enabling Hive transactions. - Due to the Amazon EMR limitation in HIVE-17221, the Update Strategy transformation on transaction enabled partitioned Hive tables fails.
Partition Merge Solution	- Works on any Hadoop distribution. - No need for ACID transactions on Hive. - Hive tables do not need to be bucketed. - Hive tables can use any Hive supported file format. - With the correct partitioning strategy, this approach scales well for extremely large datasets.	- Does not support updating the entire Hive partition (i.e. updating all records from within a partition). - You must understand the underlying data distribution to decide on the correct partitioning strategy.
Key-Value Stores	- No need for ACID transactions on Hive. - HBase implements inserts as upserts, which is convenient for handling incremental updates.	- Informatica Big Data Management does not handle delete operations. - Delete operations must be forked as a separate standalone query. - HBase stores are not efficient for large scans. - HBase is CPU, memory, and I/O intensive and will compete for additional system resources on the cluster. - Key-value stores are not intended for large data volumes greater than 500 GB.

Authors

Neha Velhankar

Alison Peek