How-To Library

Informatica

Tuning and Sizing Guidelines for Data Engineering Integration (10.4.x)

Abstract

You can tune Informatica Data Engineering Integration for better performance. This article provides sizing recommendations for the Hadoop cluster and the Informatica domain, tuning recommendations for various Data Engineering Integration components, best practices to design efficient mappings, and troubleshooting tips. This article is intended for Data Engineering Integration users, such as Hadoop administrators, Informatica administrators, and Informatica developers.

Supported Versions

- Data Engineering Integration 10.4 x

Table of Contents

Overview

Tuning Data Engineering Integration for better performance includes tuning the Hadoop environment and the Informatica domain environment.

You can tune Data Engineering Integration in the following areas:

- Hardware
- Hadoop cluster parameters
- Domain parameters and application services in the domain
- Data Engineering engines

Deployment Types

Sizing and tuning recommendations vary based on the deployment type. Based on certain deployment factors in the domain and Hadoop environments, Informatica categorizes Data Engineering Integration into the following types:

- Sandbox deployment
- Basic deployment
- Standard deployment
- Advanced deployment

Deployment Criteria

The following criteria determine the Data Engineering Integration deployment type:

Number of active users

Number of users working on the Model repository at design time, using the Analyst tool, or running data engineering jobs in the native or Hadoop run-time environment at any given point of time.

Number of concurrent pushdown mappings

Total number of mappings running on the Blaze, Spark, or Hive engines that are concurrently submitted to the Data Integration Service.

Number of objects in the Model repository

Total number of design-time and run-time objects in the Model repository. For example, data objects, mappings, workflows, and applications.

Number of deployed applications

Total number of applications deployed across all the Data Integration Services in the Informatica domain.

Number of objects per application

Total number of objects of all types that are deployed as part of a single application.

Total operational data volume

Total volume of data being processed in the Hadoop environment at any given point of time.

Total number of data nodes

Total number of data nodes in the Hadoop cluster.

yarn.nodemanager.resource.cpu-vcores

A property in the yarn-site.xml on the Hadoop cluster that specifies the number of virtual cores for containers.

yarn.nodemanager.resource.memory-mb

A property in the yarn-site.xml on the Hadoop cluster that specifies the maximum physical memory available for containers.

Deployment Type Comparison

Compare Data Engineering Integration deployment types based on the standard values for each deployment factor.

Domain Environment

The following table contains guidelines for deployment factors in the domain environment:

Deployment Factor	Sandbox Deployment	Basic Deployment	Standard Deployment	Advanced Deployment
Number of active users	1	5	10	50
Number of concurrent pushdown mappings	< 10	10 – 1000	1000 – 2000	2000 – 5000
Number of objects in the Model repository	< 1000	< 5000	< 20,000	20,000 +
Number of deployed applications	< 10	< 25	< 100	< 500
Number of objects per application	< 10	10 - 50	50 -100	50 -100

Deployment Factor	Sandbox Deployment	Basic Deployment	Standard Deployment	Advanced Deployment
Total operational data volume on the compute cluster (for batch processing use cases)	10 GB	100 GB	500 GB	1 TB +
Messages processed per second (for streaming use cases with Informatica® Data Engineering Streaming)	100,000	500,000	1 Million	10 Million

Hadoop Environment

The following table contains guidelines for deployment factors in the Hadoop environment:

Deployment Factor	Sandbox Deployment	Basic Deployment	Standard Deployment	Advanced Deployment
Total number of data nodes	3	5 - 10	10 - 50	50 +
yarn.nodemanager.resource.cpu-vcores	12	24	24	36
yarn.nodemanager.resource.memory-mb	12288 MB	24576 MB	49152 MB	98304 MB

Based on the deployment type that you use to categorize Data Engineering Integration, you can use `infacmd autotune autotune` to automatically tune certain properties in your environment.

For more information, see the *Data Engineering Administrator Guide*.

Sizing Recommendations

Based on your deployment type, use the sizing guidelines for the Hadoop and domain environments.

Hadoop Cluster Hardware Recommendations

The following table lists the minimum and optimal hardware requirements for the Hadoop cluster:

Hardware	Sandbox Deployment	Basic or Standard Deployment	Advanced Deployment
CPU speed	2 - 2.5 GHz	2 - 2.5 GHz	2.5 - 3.5 GHz
Logical or virtual CPU cores	16	24 - 32	48
Total system memory	16 GB	64 GB	128 GB
Local disk space for yarn.nodemanager.local-dirs[1]	256 GB	500 GB	2.4 TB
DFS block size	128 MB	256 MB	256 MB

Hardware	Sandbox Deployment	Basic or Standard Deployment	Advanced Deployment
HDFS replication factor	3	3	3
Disk capacity	32 GB	256 GB - 1 TB	1.2 TB
Total number of disks for HDFS	2	8	12
Total HDFS capacity per node	64 GB	2 - 8 TB	At least 14 TB
Number of nodes	2 +	4 - 10+	12 +
Total HDFS capacity on the cluster	128 GB	8 - 80 TB	144 TB
Actual HDFS capacity (with replication)	43 GB	2.66 TB	57.6 TB
/tmp mount point	20 GB	20 GB	30 GB
Installation disk space requirement	12 GB	12 GB	12 GB
Network bandwidth (Ethernet card)	1 Gbps	2 Gbps (bonded channel)	10 Gbps (Ethernet card)

[1] A property in the yarn-site.xml that contains a list of directories to store localized files. You can find the localized file directory in: `${yarn.nodemanager.local-dirs}/usercache/${user}/appcache/application_${appid}`. You can find the work directories of individual containers, `container_${contid}`, as the subdirectories of the localized file directory.

MapR Cluster Recommendation

When you run mappings on the Blaze, Spark, or Hive engine, local cache files are generated under the directory specified in the yarn.nodemanager.local-dirs property in the yarn-site.xml. However, the directory might not contain sufficient disk capacity on a MapR cluster.

To make sure that the directory has sufficient disk capacity, perform the following steps:

1. Create a volume on HDFS.
2. Mount the volume through NFS.
3. Configure the NFS mount location in yarn.nodemanager.local-dirs.

For more information, refer to the MapR documentation.

Amazon EMR Sizing Guidelines

The following table lists the requirements for an Amazon EMR cluster:

Deployment Environment	Sandbox	Basic/Standard	Advanced
Storage type	HDD optimized	HDD optimized	HDD optimized
Number of EBS volumes per node	2	2-4	6-8
EBS volume size for HDFS	100 GB	100-250 GB	250-500 GB
Total HDFS capacity per node	200 GB	200-1000 GB	1.5-4.5 TB

Deployment Environment	Sandbox	Basic/Standard	Advanced
Replication factor	2	2	3
YARN VCores per node	14	14-30	36
YARN memory per node	28 GB	54 GB	144 GB
Total operational data volume	10 GB	100-500 GB	1 TB +
Recommended minimum number of nodes	2	5-10	10 +
Recommended instance types, Informatica version 10.4.0	m5.4xlarge, c4.4xlarge	m5.4xlarge, c4.8xlarge	m5.10xlarge
Recommended instance types, Informatica version 10.4.1	m5.4xlarge, c5.4xlarge	m5.4xlarge, c5.8xlarge	m5.10xlarge

HDInsight Sizing Guidelines

The following table lists the requirements for an HDInsight cluster:

Deployment Environment	Sandbox	Basic/Standard	Advanced
Storage type	Storage v2 (general purpose v2), ADLS Gen2	Storage v2 (general purpose v2), ADLS Gen2	Storage v2 (general purpose v2), ADLS Gen2
Total operational data volume	10 GB	100-500 GB	1 TB +
Recommended instance types	D4 v2, D5 v2	D4 v2, D5 v2	D4 v2, D5 v2
Recommended minimum number of nodes	2	5-10	10 +

Azure Databricks Cluster Sizing Guidelines

The following table lists the requirements for a Databricks cluster on the Azure platform:

Deployment Environment	Sandbox	Basic/Standard	Advanced
Storage type	Storage v2 (general purpose v2), ADLS Gen2	Storage v2 (general purpose v2), ADLS Gen2	Storage v2 (general purpose v2), ADLS Gen2
Total operational data volume	10 GB	100-500 GB	1 TB +
Recommended instance types	Standard_D8s_v4, Standard_DS13_v2	Standard_DS4_v2, Standard_DS5_v2, Standard_DS13_v2	Standard_DS4_v2, Standard_DS5_v2, Standard_DS13_v2
Recommended minimum number of nodes	2	5-10	10 +

Informatica Domain Sizing

The following table lists the minimum hardware requirements for the server on which the Informatica domain runs:

Deployment Type	Use Case	Total Virtual Cores	Ram Per Node	Disk Space Per Node[1]
Sandbox	Proof of concept, or a sandbox environment with minimal users	16	32 GB	50 GB
Basic	Low-volume processing, low concurrency	24	32 GB	100 GB +
Standard	High-volume processing, low concurrency	48	64 GB	100 GB +
Advanced	High-volume processing, high concurrency	96	64 GB	100 GB +

[1] Disk space requirement is for Informatica services. Additional disk capacity is required to process data in the native run-time environment.

Note: Informatica services are designed to scale. You can begin with a basic deployment type. Over time, you can promote your domain to a standard or advanced deployment type to increase computational resources.

Data Engineering Streaming Sizing and Tuning Recommendations

Use Informatica® Data Engineering Streaming mappings to collect streaming data, build the business logic for the data, and push the logic to a Spark engine for processing. The Spark engine uses Spark Streaming to process data. Streaming mapping includes streaming sources such as Kafka or JMS. The Spark engine reads the data, divides the data into micro batches, and publishes it.

Streaming mappings run continuously. When you create and run a streaming mapping, a Spark application is created on the Hadoop cluster which runs forever unless killed or cancelled through the Data Integration Service. Because a batch is triggered for every micro batch interval that is configured for the mapping, consider the following recommendations:

- The processing time for each batch must remain the same over the entire duration.
- The batch processing time of every batch must be less than batch interval.

Tune the Hardware and the Hadoop Cluster

Tune the following hardware parameters for better performance:

- CPU frequency
- NIC card ring buffer size

Tune the following Hadoop cluster parameters for better performance:

- Hard disk
- Transparent huge page
- HDFS block size
- HDFS access timeout
- YARN settings for parallel jobs

Tune the Informatica Domain and Application Services

Informatica service processes can use a large number of files and Informatica services can use a large number of user processes. To account for all the files and user processes, you can change the file descriptor and max user processes settings.

To change the settings, edit the infaservice.sh file at the following location:

<INFA_HOME>/tomcat/bin/

For example:

/export/home/<INFA_HOME>/tomcat/bin/infaservice.sh

```
# Add on extra jar files to CLASSPATH
CLASSPATH="${INFA_JAVA_CMD_CLASSPATH}":"${CATALINA_HOME}"/bin/bootstrap.jar:"${CATALINA_HOME}"/b

_DEBUG=""
#_DEBUG=-Xrunjdwp:transport=dt_socket,server=y,address=9000,suspend=n

INFA_JAVA_OPTS="-Xmx512m ${INFA_JAVA_OPTS} -XX:GCTimeRatio=9 -XX:MaxMetaspaceSize=256m"
  export INFA_JAVA_OPTS
```

The following table lists the optimal values for the domain parameters:

Domain Parameters	Sandbox Deployment	Basic Deployment	Standard Deployment	Advanced Deployment
Heap requirement	512 MB (Default)	512 MB (Default)	2 GB	4 GB
File descriptors	1024 (Default)	26000	32000	64000
Max user processes	1024 (Default)	8000	16000	32000

You can tune the following application services for performance:

- Analyst Service
- Content Management Service
- Data Integration Service
- Mass Ingestion Service
- Model Repository Service
- Search Service

Analyst Service

You can tune the maximum heap size for the Analyst Service.

The following table lists the guidelines to tune the heap sizes:

Parameter	Sandbox Deployment	Basic Deployment	Standard Deployment	Advanced Deployment
Max heap size	768 MB	1 GB	2 GB	4 GB

Note: The number of active concurrent users working on the Analyst tool and the number of objects processed per user determine the heap size.

Content Management Service

You can tune the maximum heap size for the Content Management Service.

The following table lists the guidelines to tune the heap sizes:

Parameter	Sandbox Deployment	Basic Deployment	Standard Deployment	Advanced Deployment
Max heap size	1 GB	2 GB	4 GB	4 GB

Note: The number and size of the reference tables determine the heap size.

Data Integration Service

You can configure the maximum heap size and the batch execution pool sizes for the Hadoop and native environments.

The following table lists the recommended values for the Data Integration Service parameters:

Parameter	Sandbox Deployment	Basic Deployment	Standard Deployment	Advanced Deployment
Max heap size	1 GB (default)	6 GB	8 GB	16 GB
Execution pool size for the Hadoop environment	100	1000	2000	5000
Execution pool size for the native environment	10	10	15	30

Note: The number of concurrent pushdown jobs submitted to the Data Integration Service determine the heap size and the execution pool size.

If you implement CI/CD, use a maximum heap size of 8 GB to scale CI/CD operations for up to 1,000 objects.

Best Practices for Handling Highly Concurrent Workloads

If you tend to run highly concurrent workloads (over ~5000 concurrent jobs), consider the following best practices:

- Create a separate Model repository for persisting monitoring statistics.
- If the workloads include Sqoop sources, set the custom property ExecutionContextOptions.SqoopPoolSize on the Data Integration Service.

Custom Property	Description
ExecutionContextOptions.SqoopPoolSize	Number of concurrent Sqoop jobs. If you set the ExecutionContextOptions.SqoopPoolSize to a value of -1, the concurrency for Sqoop is determined by the value set for the Maximum Hadoop Batch Pool Size property. If you want to restrict the number of Sqoop jobs that run in parallel, you can set ExecutionContextOptions.SqoopPoolSize to a value between 0 and 100. The value you specify controls the concurrency of Sqoop jobs. Default is 100. Recommended value is -1.

For more information about this property, see the "Sqoop Concurency" section and KB article 570014.

- Because an increase in the number of concurrent jobs increases monitoring activity, you can also set the following custom property on the Data Integration Service Process custom properties to increase the buffer size:

Custom Property	Description	Default	Recommendation
MonitoringOptions.StatsBufferSize	Size of the monitoring statistics buffer, measured in number of jobs	10000	25000

Use the infacmd gateway service to run highly concurrent pushdown mappings.

The gateway service is a single-client Java process that intercepts requests from the infacmd client. The gateway service submits the requests as mappings or workflows to the Data Integration Service using threads to limit system resource consumption.

Configure the infacmd gateway service on a remote machine other than the Data Integration Service host machine. Configure the following properties:

Parameter	Sandbox Deployment	Basic Deployment	Standard Deployment	Advanced Deployment
CPU cores	1	1	2	4
Memory	512 MB	1 GB	2 GB	4 GB

To enable and configure the gateway service, edit the properties file in the following location:

```
<Informatica_installation_directory>/isp/bin/plugins/ms/clientgateway/
msgatewayconfig.properties
```

In the properties file, configure the following property:

Property	Value
enable_client_gateway	true

For details on configuring the gateway service, refer to the following article:

Gateway Service to Submit Mappings and Workflows to the Data Integration Service

Additional Guidelines

Consider the following additional guidelines for the Data Integration Service:

- Application deployment requires communication between the Data Integration Service and the associated Model Repository Service. To fetch objects and to write to the database schema of the Model Repository Service, tune the database cursors as follows:

  ```
  Number of database cursors >= Number of objects in the application
  ```

- To run jobs in the native environment and to preview data, the Data Integration Service requires at least one physical core for each job execution.
- If the Data Integration Service is enabled to use multiple partitions for native jobs, the Data Integration Service node resource requirements increase based on the parallelism. If the number of jobs in the native environment are typically high, you must allocate additional resources for other jobs.

Profiling Parameters

To optimize performance, perform profiling using the Blaze engine. Tuning profiling performance involves configuring the Data Integration Service parameters, the profile database warehouse properties, and the advanced profiling properties.

Mass Ingestion Service

Mass ingestion is a data engineering solution that you can use to replicate or ingest data from different relational sources to a data lake or a Hadoop cluster.

To improve job performance, consider the best practices in the following areas:

Data Integration Service deployment

Create a Data Integration Service that is dedicated to mass ingestion jobs. Deploy mass ingestion specifications to the dedicated service.

Sqoop concurrency

Sqoop pool size determines the number of deployed Sqoop jobs that you can run concurrently in the Hadoop environment.

By default, Sqoop pool size is set to 100. You can disable the Hadoop batch execution pool by setting the value of the Maximum Hadoop Batch Execution Pool Size property to -1. This forces the Data Integration Service to control Sqoop concurrency by treating each Sqoop job equally.

The following table describes the Maximum Hadoop Batch Execution Pool Size property:

Property	Description	Reccomended Value
Maximum Hadoop Batch Execution Pool Size	Maximum number of deployed jobs that can run concurrently in the Hadoop environment. The Data Integration Service moves Hadoop mapping jobs from the queue to the Hadoop job pool when enough resources are available. Default: 100.	-1

See the "Best Practices for Highly Concurrent Workloads" section of the "Data Integration Service" on page 10 topic.

Relational database concurrency

To allow concurrent Sqoop jobs to establish connections to the database, ensure that the database supports concurrent database connections.

Sqoop performance

Mass ingestion uses Sqoop jobs to ingest data from relational tables to Hive or HDFS targets on the Hadoop cluster. By default, Sqoop jobs spawn 4 tasks. Each task establishes one connection to the relational database.

To reconfigure the number of tasks in a Sqoop job, configure the following Sqoop argument in the JDBC connection:

Argument	Description
-num-mappers -m	Number of mappers (tasks) to run concurrently. Default is 4 when Sqoop jobs run on the Spark engine.

Note: When you reconfigure the JDBC connection, the changes affect all Sqoop jobs that use the connection. For example, if you reconfigure the JDBC connection to increase the number of tasks to 10, then 50 concurrent Sqoop jobs spawn 500 tasks and require 500 database connections.

Polling time

The polling time determines how often the Mass Ingestion tool updates the status of ingestion jobs.

By default, the polling time is 30 seconds. You can decrease the polling time to increase the number of requests and refresh the ingestion job status more frequently.

To configure the polling time, set the following custom property on the Mass Ingestion Service:

```
POLLING_TIME=<time in seconds>
```

Mass ingestion specifications

When you create a mass ingestion specification, ingest no more than 2,000 relational tables in the specification.

Model Repository Service

You can tune the maximum heap size for the separate Model Repository Service instances that you use for design and for monitoring.

The following table lists the guidelines to tune heap sizes:

Parameter	Sandbox Deployment	Basic Deployment	Standard Deployment	Advanced Deployment
Max heap size -- Design Model repository	1 GB	1 GB	2 GB	4 GB
Max heap size -- Monitoring Model repository	1 GB	1 GB	2 GB	4 GB

Note: Heap size requirements on the Model Repository Service are primarily driven by the number of simultaneous save, fetch, and delete operations and the number of concurrent mapping executions. Model Repository Service actions, such as application deploy, redeploy, import, and export, also affect the heap size requirements and the efficiency of design-time operations.

If you implement CI/CD, use a maximum heap size of 8 GB on the design Model repository to scale CI/CD operations for up to 1,000 objects.

Additional Guidelines

Consider the following additional guidelines for the Model Repository Service:

- Create a separate Model Repository Service for persisting monitoring statistics.
- Schedule a periodic purge of monitoring statistics and retain only the required statistics.

- When you upgrade the Model Repository Service, the minimum JVM heap requirement is 4 GB.

When you create a Model Repository Service for monitoring, you must size your database to account for the growth of monitoring statistics. Consider the rate of growth in monitoring statistics when you decide statistics retention policies and schedule purge policies for the Model Repository Service. For instance, the monitoring statistics for 1,000 mappings require around 200 MB. Retaining the statistics for 30 days with a daily workload of 1,000 mappings requires around 6 GB of tablespace on the database.

Search Service

You can tune the maximum heap size for the Search Service.

The following table lists the guidelines to tune the heap sizes:

Parameter	Sandbox Deployment	Basic Deployment	Standard Deployment	Advanced Deployment
Max heap size	768 MB	1 GB	2 GB	4 GB

Note: The size of the Model Repository Service and the profiling warehouse and the volume of business glossary data being processed determine the heap size.

The following table lists additional Search Service properties that you can tune:

Parameter	Description and Recommendation
Index Location	Directory that contains the search index files. To save index files, you allocate disk space on one of the nodes in the domain. Make sure that the disk location has enough space to write index files. Recommendation: Disk space must be approximately twice the size of the schema.
Extraction Interval	Interval in seconds at which the Search Service updates the search index. The Search Service checks for changes in the metadata before reindexing. Default extraction interval is 60 seconds. Recommendation: The extraction interval must be greater than or equal to the time taken to complete indexing the first time.

Tune the Blaze Engine

When you develop mappings in the Developer tool to run on the Blaze engine, consider the following tuning recommendations and performance best practices.

Mapping Optimization

Consider the following best practices when you develop mappings to run on the Blaze engine:

Tune port precision values.

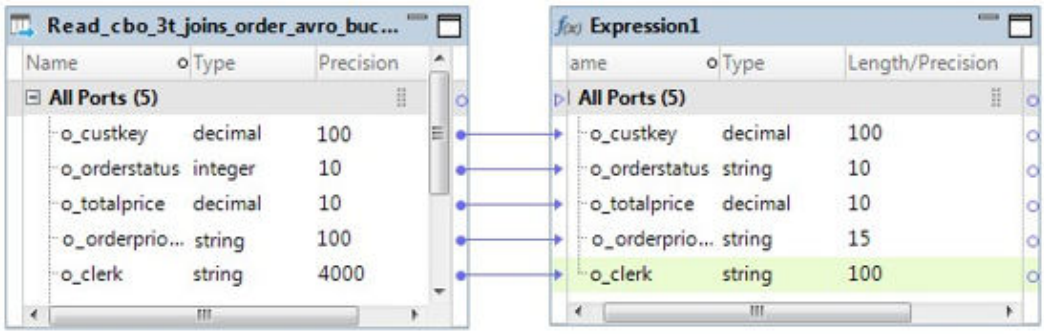
When you import Hive data objects, the precision of string ports is set to 4000 by default. Aggregator and Rank transformations are automatically converted to sorted Aggregator and sorted Rank transformations. The sorted Aggregator and Rank transformations have a variable length cache which reduces the size of generated cache. The variable length cache eliminates the need to manually tune precision for string ports and reduces the chance of mapping failures due to inadequate disk space.

Performance best practice: String ports with large precision require a larger buffer memory for data processing, which reduces the number of rows that are processed per block. When performance is critical, tune string ports to avoid large precision.

Avoid unnecessary data type conversions.

Mismatched port data types and mismatched port precisions increase the computational overhead. Ensure that the port precisions and data types are consistent across sources, transformations, and targets within a mapping.

The following image shows a mismatch between the data type and the precision values in some of the ports in the Read transformation and the Expression transformation:



Optimize transformation cache.

When processing cache-based transformations, such as Aggregator, Joiner, Sorter, and Lookup transformations, the Data Integration Service stores conditional values and output values in the transformation cache.

If the memory allocated to the cache-based transformation is not sufficient, the cache data is spilled or written to disk. To avoid data spill to disk, set explicit cache sizes for each Lookup, Joiner, Sorter, and Aggregator transformation in a mapping.

In the Developer tool, select a transformation and set numeric values in bytes for the following run-time properties in the **Advanced** tab:

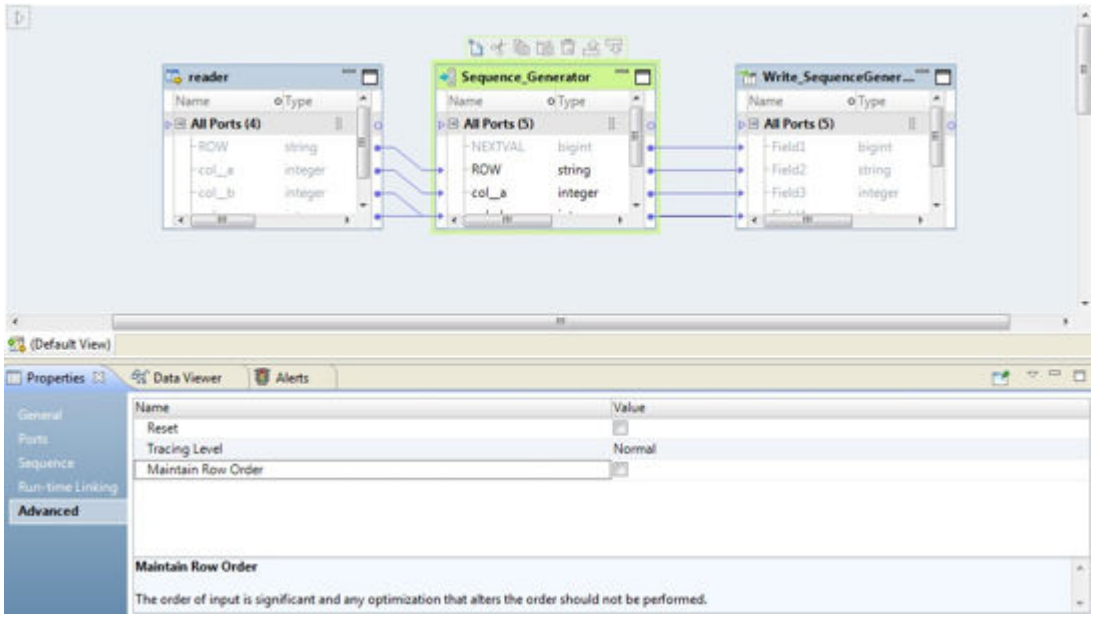
- <Transformation type> Data Cache Size
- <Transformation type> Index Cache Size

Transformation Optimization

You can optimize transformations to enable the Blaze engine to process transformations in a mapping efficiently.

Sequence Generator Transformation

To **Maintain Row Order**, the Sequence Generator transformation generates the following types of global and unique sequences. The following image shows where you can configure the **Maintain Row Order** property:



The Sequence Generator transformation generates the following types of sequence numbers:

Global sequence

When you configure the transformation to maintain row order, the mapping runs in a single partition to generate a global unique running sequence. You must run the job in a single partition with only one mapper or reducer. When the Blaze engine performs distributed processing it cannot generate numbers in sequential order.

Configure the transformation to maintain row order only if your business needs require a global unique running sequence.

Unique sequence

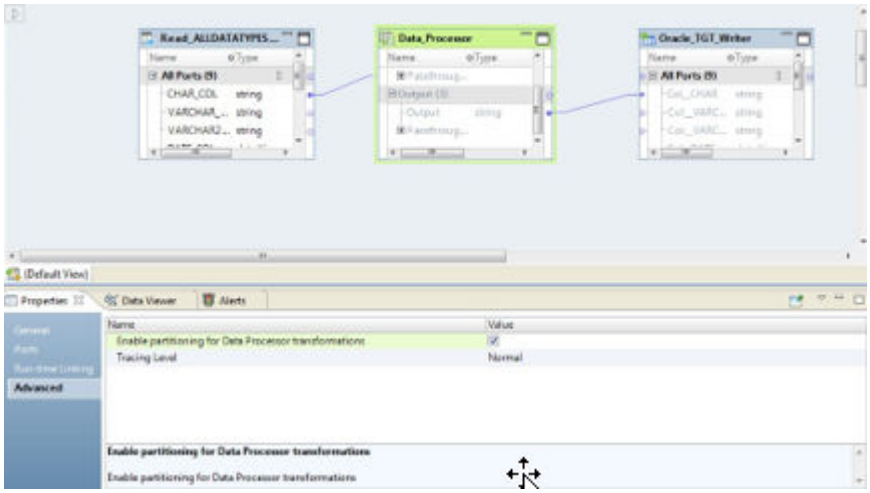
When you do not configure the transformation to maintain row order, the mapping runs in multiple partitions to efficiently generate unique sequence numbers.

Data Processor Transformation

Consider the following best practices for mappings that contain a Data Processor transformation:

- For mappings that run on the Blaze engine, it is required to enable partitioning for Data Processor transformations. To enable partitioning, perform the following steps:
  1. In the Developer tool, open the mapping and select the Data Processor transformation.
  2. In the **Properties** view, click the **Advanced** tab.

3. Select **Enable partitioning for Data Processor transformations**.
   The following image shows the Advanced tab of a Data Processor transformation:

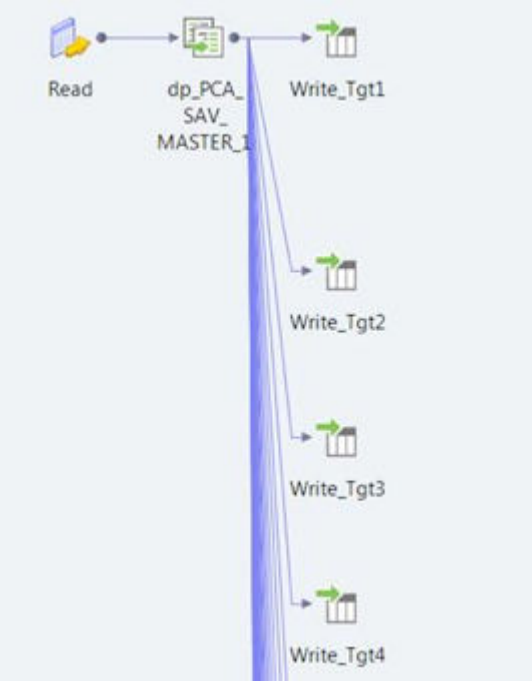


- When a mapping with a Data Processor transformation meets all of the following conditions, the Blaze engine processes the entire mapping in a single tasklet:

  - The mapping source file is of a non-splittable input format.
  - The transformation contains multiple output groups.

  The Data Processor transformation might output a higher data volume than the source. For such scenarios, configure the Blaze engine to first stage the data generated by the transformation at each output group.

  The following image shows a mapping with a Data Processor transformation with multiple output groups:



To stage data at every output group, set the following mapping run-time property in the Developer tool:

Parameter	Value
Blaze.StageOutputGroupDataForInstances	The name of the Data Processor transformation instance.

When the Blaze engine is configured to first stage the data, it performs the following tasks:

- Re-partitions the data.
- Processes the staged data.
- Creates the correct number of tasklets based on the staged data volume.

Aggregator Transformation

Disable map-side aggregation if a unique key is used as the group by key in an Aggregator transformation.

In the Blaze engine, map-side aggregation is analogous to the aggregation done in the map phase in a MapReduce job that runs on the Hive engine. Source data is aggregated based on the group-by port set in an Aggregator transformation. The aggregated data then moves to the data shuffle stage for the second level of aggregation.

If you specify a unique key as the group by port, disable the map-side aggregation in a mapping that runs on the Blaze engine.

In the Developer tool, set the following run-time property in the Run-time tab to disable map-side aggregation for the mapping:

Parameter	Value
GridExecutor.EnableMapSideAgg	False

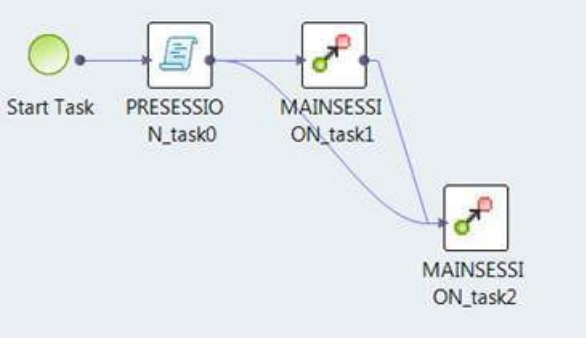
Filter Optimization for Hive Sources

To optimize mappings that read from a partitioned Hive source, you typically add filter conditions on a relational data object to remove rows at the source. The filter limits the data that flows through the mapping pipeline which gives a performance benefit. However, when the Blaze engine reads from a Hive source with a filter condition, the engine interprets the filters as SQL overrides, translates into multiple grid tasks, and adds a performance overhead.

The Blaze engine translates the filters into the following grid tasks:

- A grid task to create a HiveServer2 job for the filter override to stage the intermediate data.
- A grid task to operate against the staged data and to apply the downstream mapping logic.

For example, the following image shows the Blaze engine execution plan with two grid tasks for a passthrough mapping with a filter:

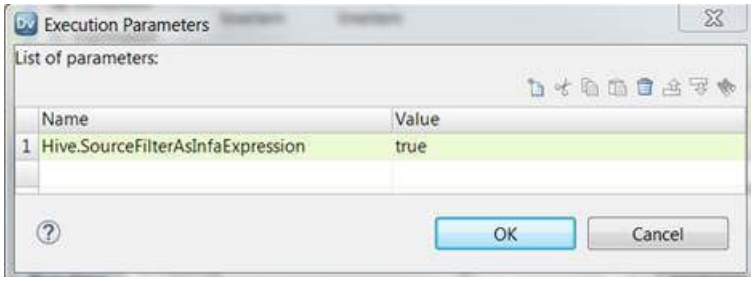


To avoid the performance overhead, set the following custom flag as an advanced property of the mapping:

```
Hive.SourceFilterAsInfaExpression = true
```

If multiple mappings require the flag, add the custom flag at the Data Integration Service level.

The following image shows the custom flag that you added as an execution parameter for the mapping:



The following image shows the Blaze engine execution plan with a single grid task after you set the custom flag:

Note: Use a valid Informatica expression as a filter condition. The filter condition must not refer to the table name. For example, instead of `LineItem.l_tax > $dataasofdate`, use `l_tax > $dataasofdate`.

The following image shows the **Query** view of the relational data object where you define the filter condition:

Tune the Spark Engine

When you develop mappings in the Developer tool to run on the Spark engine, consider the following prerequisites, tuning recommendations, and performance best practices.

Meet the following prerequisites:

- On the Hadoop cluster, configure the Spark History Server.
- On the Hadoop cluster, enable the Spark Shuffle Service.
- To run mappings on the Spark engine, configure the Hadoop connection with the location of the Spark HDFS staging directory and the Spark event log directory.
  Use the same directory as the event log directory from which the Spark History Server is reading. The Spark Event Log Directory is the base directory that logs Spark events. Within this base directory, Spark creates a subdirectory for each application, and logs the events specific to the application in this directory.

For more information about configuring Spark History Server and Spark Shuffle Service, refer to the Hadoop distribution documentation or the Apache Spark documentation.

For more information about configuring the Hadoop connection, refer to the *Data Engineering Integration User Guide*.

Spark Configuration

Configure properties for the Spark engine in the Hadoop connection.

You can use the following properties to tune job execution when you run jobs with the Spark engine:

spark.executor.memory

Amount of memory to use per executor process. Specify a value with a size unit suffix "k", "m", "g" or "t". For example, 512k or 1g. Default: 1 GB.

spark.executor.cores

The number of cores to use on each executor. Default: 1.

Infaspark.shuffle.max.partitions

Sets the number of shuffle partitions to the maximum number of partitions seen across all input sources. Default: 10000.

Recommended value: Allocate approximately 8 dynamic shuffle partitions for each gigabyte of shuffle data. For example, for 400 GB of shuffle data, set this value to 3200.

For columnar formats like ORC for Hortonworks or Parquet for Cloudera, you might set this property to a lower value.

If the data being shuffled in mid-stream is less than ~250 GB, you can reduce the value of infaspark.shuffle.max.partitions to 1000 for increased performance.

spark.driver.memory

Sets the driver process memory to a default value of 4 GB. The driver requires more memory based on the number of data sources and data nodes.

Recommended value: Allocate at least 256 MB for every data source participating in map join. For example, if a mapping has eight data sources, set the driver memory to at least 2 GB (8 x 256).

spark.driver.maxResultSize

Limit of total size of serialized results of all partitions for each Spark action in bytes. Should be at least 1M, or 0 for unlimited. Jobs will be aborted if the total size is above this limit. Default: 1 GB.

The following table lists the tuning recommendations for sandbox, basic, standard, and advanced deployment types:

Property	Sandbox Deployment	Basic Deployment	Standard Deployment	Advanced Deployment (Default)
spark.executor.memory	2 GB	4 GB	6 GB	6 GB
spark.executor.cores	2	2	2	2
Infaspark.shuffle.max.partitions (8 * per GB data at shuffle)	800	800	4000	10000
spark.driver.memory	1 GB	2 GB	4 GB	4 GB + (default)
spark.driver.maxResultSize	1 GB	1 GB	2 GB	4 GB + (default)

Transformation Optimization

You can optimize transformations to enable the Spark engine to process transformations in a mapping efficiently.

Java Transformation

Consider the following elements when you optimize the Java transformation to run on the Spark engine:

Complex data types

When you process complex data types in the Java transformation, the Spark executor memory requirements increase depending on the complexity and the level of nesting in the hierarchical data.

With higher complexity and nesting, jobs that are executed on the Spark engine might not progress or they might eventually fail with an `Out of Memory` error. If these issues occur, increase the Spark executor memory or run only one task per executor.

To increase the Spark executor memory, configure the run-time property `spark.executor.memory` for the Hadoop execution environment. To run one task per executor, configure the run-time property `spark.executor.cores` for the Hadoop execution environment and set the value equal to 1.

String ports

When you process string ports in the Java transformation, there is an overhead cost due to UTF conversions. The overhead cost is a factor of the number of string ports that are processed in the Java transformation.

Performance best practice: Propagate only the ports that are necessary to the Java transformation.

For more details, refer to the following case study:

"Case Study: Java String Port Conversion Overhead" on page 37

Joiner Transformation

You can optimize Joiner transformations to enable the Spark engine to efficiently perform a full outer join.

To increase memory for a full outer join and to determine shuffle partitions, perform the following two-step tuning process:

1. Ensure every executor core has at least 3 GB of memory.
   For example, set spark.executor.memory=6 GB and spark.executor.cores=2.

2. Set spark.sql.shuffle.partitions = <master splits> + <detailed partitions>.
   The spark.sql.shuffle.partitions property determines the number of partitions to use when shuffling data for joins or aggregations.

   For example, with a DFS block size of 256 MB, 100 GB of master data will have 400 splits and 200 GB of details will have 800 partitions.

Lookup Transformation

When a mapping runs on the Spark engine, the Spark engine translates unconnected Lookup transformations to join operations each time that you reference the transformation using a lookup expression. When there are more than five unconnected lookup expressions in a single mapping, the performance might degrade considerably depending on the size of the lookup source.

If the mapping logic requires multiple unconnected lookup expressions, consider running the mapping on the Blaze engine. The Blaze engine is optimized to generate the lookup cache one time and reuse the cache each time that an unconnected lookup expression references the same lookup source.

Router Transformation

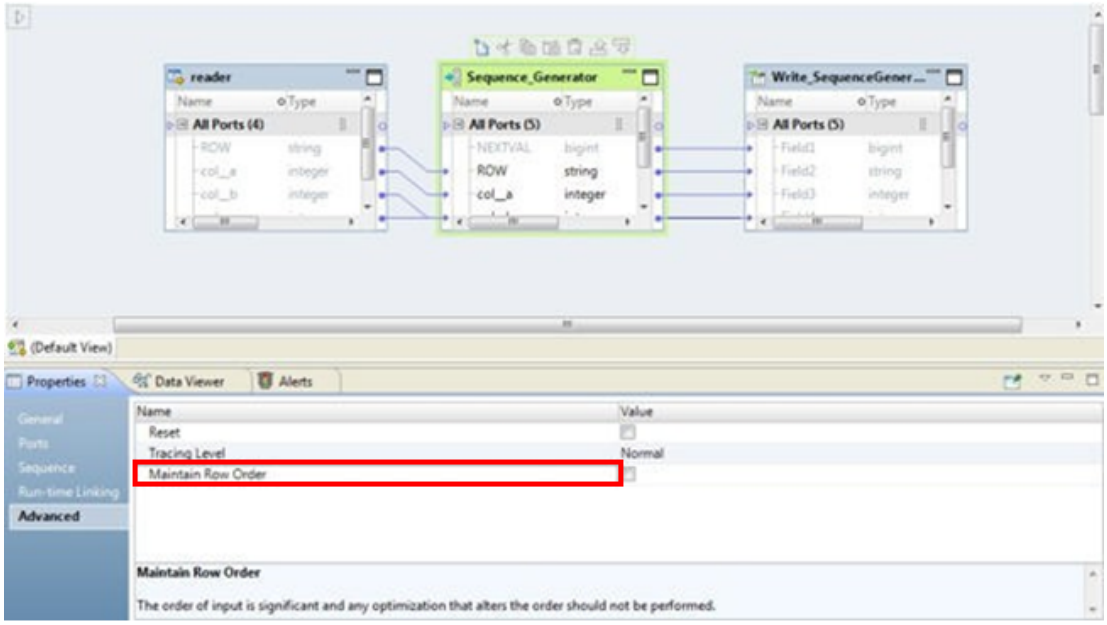
Using more than one output group in a Router transformation might cause poor performance. To avoid performance impact, configure the following properties:

Property and Value	Description
infaspark.sql.forcePersist=true	Persists data in the memory and disk to avoid repeated reads for each output group.
spark.rdd.compress=true	Enables compression to store data more efficiently.

You can set these properties either in the Spark advanced properties for the Hadoop connection or at the mapping level. If you set the properties in the Spark configuration, the values apply to all mappings that you run on the Spark engine.

Sequence Generator Transformation

To **Maintain Row Order**, the Sequence Generator transformation generates the following types of global and unique sequences. The following image shows where you can configure the **Maintain Row Order** property:



The Sequence Generator transformation generates the following types of sequence numbers:

Global sequence

When you configure the transformation to maintain row order, the mapping runs in a single partition to generate a global unique running sequence. You must run the job in a single partition with only one mapper or reducer. When the Spark engine performs distributed processing it cannot generate numbers in sequential order.

Configure the transformation to maintain row order only if your business needs require a global unique running sequence.

Unique sequence

When you do not configure the transformation to maintain row order, the mapping runs in multiple partitions to efficiently generate unique sequence numbers.

For information about troubleshooting the Sequence Generator transformation, see "Troubleshooting Spark Job Failures" on page 26.

Update Strategy Transformation

To conduct near real-time analytics in data lakes and enterprise data warehouses, it is necessary to perform frequent incremental loads. In most cases, Hive is the preferred analytic store.

To perform incremental loads on Hive, you must use transactions that are ACID-compliant, support bucketed tables, and support ORC file formats.

The following issues depict some constraints to performing incremental loads:

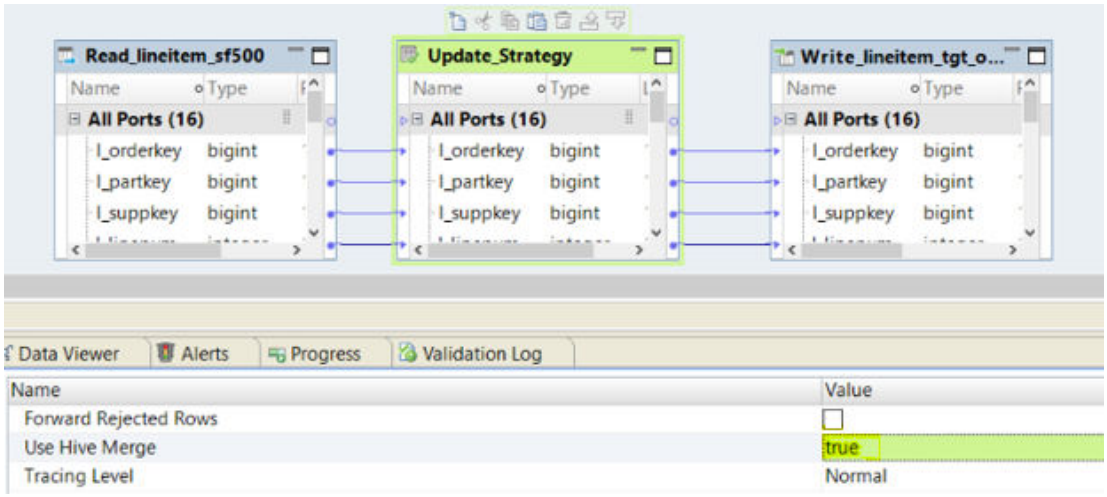
- Cloudera CDH does not endorce ORC file formats and ACID-enabled tables.
- Updates are not compatible with partitioned tables on Amazon EMR due to a Hive issue.

To accommodate for these constraints, you can implement alternate strategies to perform incremental loads. One strategy is to perform an ACID merge. You can perform an ACID merge through Hive MERGE statements in an Update Strategy transformation.

When you use MERGE statements in mappings that run on the Spark engine, the statements perform ACID transactions on Hive tables. You can use the statements to efficiently perform record-level INSERT, UPDATE, and DELETE operations.

To use Hive MERGE statements, select the option in the advanced properties of the Update Strategy transformation. You can toggle the option to use a MERGE statement or a traditional update strategy approach. The traditional update strategy approach does not employ ACID transactions.

The following image shows the Hive MERGE option in the advanced properties of an Update Strategy transformation:



When you enable Hive MERGE in an Update Strategy transformation, the following changes in performance occur:

- ~ 25% decrease in performance for INSERT statements.
- ~ 40% increase in performance for UPDATE statements.
- Comparable performance for DELETE statements.

Based on this analysis, Hive MERGE results in better performance if the update strategy task involves a large number of UPDATE statements. When there is a combination of INSERT, UPDATE, and DELETE statements, Hive MERGE is expected to produce better results than the traditional update strategy approach.

For more information on the performance analysis, see ""Case Study: Traditional Update Strategy versus Hive MERGE" on page 39"

Hive MERGE Run-time Failures

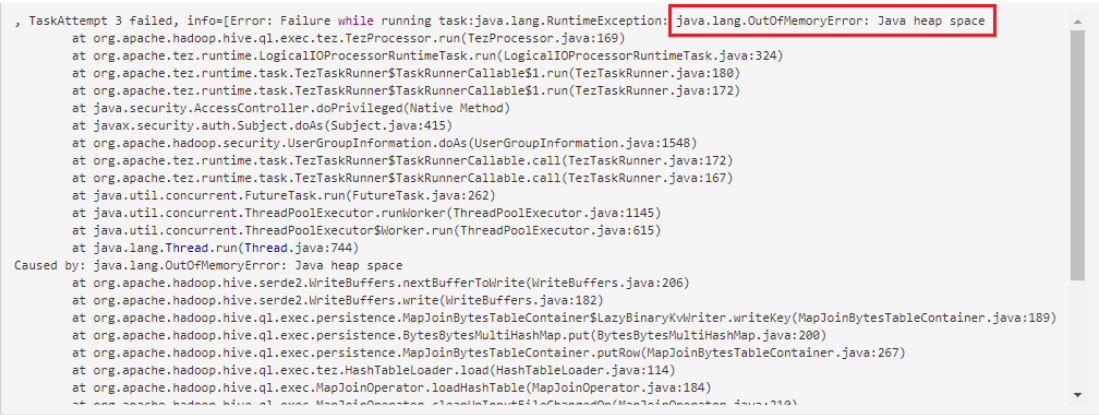
When you use Hive MERGE statements to perform update strategy tasks, the mapping might fail at run time.

The MERGE statement internally performs a map join during transactional operations on Hive tables. Due to this behavior, it is possible to encounter a run-time failure for mapping tasks that contain an Update Strategy transformation and a partitioned bucketed Hive table.

Task failures occur due to the following error:

```
java.lang.OutOfMemoryError: Java heap space
```

The following image shows a log file that contains the error:



If you encounter the error, you can use one of the following workarounds:

Option 1. Disable map join.

To disable map join, set `hive.auto.convert.join=false` for the Environment SQL in the Hive connection properties.

Option 2. Reduce the map join threshold.

If the map join threshold has been modified, reduce the threshold. The default value is 10MB. To reduce the value, set `hive.auto.convert.join.noconditionaltask.size=10000000`, or to a lower value in bytes, for the Environment SQL in the Hive connection properties.

The issue is tracked in HIVE-8044.

Alternate Strategies to Perform Incremental Loads

In addition to Hive MERGE, you can use the following alternate strategies to perform incremental loads:

Use a traditional update strategy approach.

Define expressions in an Update Strategy transformation with IIF or DECODE functions to set rules for updating rows.

Because the approach has restrictions on Cloudera CDH and Amazon EMR distributions, you might want to use a different strategy.

Perform updates using the partition merge solution.

Detect, stage, and update only the impacted partitions on the incremental records from upstream systems.

Perform updates using key-value stores.

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

For detailed information on these alternate approaches, see the Informatica® How-To Library article "Strategies for Incremental Updates on Hive in Big Data Management 10.2.1" on https://docs.informatica.com/.

Update Strategy Transformations Using Relational Table Targets

To tune the performance of an Update Strategy transformation with JDBC-connected relational table targets, use the following properties:

Spark.JdbcNumPartitions

The number of partitions to write to the relational database.

Each Spark task opens a connection to the database. This might result in too many connections. Use this property to limit the number of open connections to the database.

To calculate the value of Spark.JdbcNumPartitions, use the formula `Min(X,Y)` where:

X = (Data volume in GB) * 4

and

Y = (Max connections allowed on database)

For example, where Data volume = 2 GB, and Max connections allowed on database = 10, the formula is:

```
Min(8,10)
```

The value of Spark.JdbcNumPartitions should be 8.

Spark.JdbcBatchSize

Number of rows to be sent to the database target as a batch. Must be an integer. Default: 10000.

When the number of columns/fields is large and results in a large row size, consider reducing the value of this property to ~5000-8000.

Troubleshooting Spark Job Failures

This section provides information on troubleshooting common error messages and limitations that you might encounter when you enable dynamic resource allocation on the Spark engine. These errors might occur when you process a large volume of data, such as 10 TB or more, or when a job has a large shuffle volume.

Could not find CoarseGrainedScheduler.

When you stop a process, you might lose one or more executors with the following error:

```
cluster.YarnScheduler: Lost executor 8 on myhost1.com: remote Rpc client disassociated
```

One of the most common reasons for executor failure is insufficient memory. When an executor consumes more memory than the maximum limit, YARN causes the executor to fail. By default, Spark does not set an upper limit for the number of executors if dynamic allocation is enabled. (SPARK-14228)

Configure the following advanced properties for Spark in the Hadoop connection:

Property	Description
spark.dynamicAllocation.maxExecutors	Set a limit for the number of executors. Determine the value based on available cores and memory per node.
spark.executor.memory	Increase the amount of memory per executor process. The default value is 6 GB.

Total size of serialized results is bigger than spark.driver.maxResultSize.

The spark.driver.maxResultSize property determines the limit for total size of serialized results across all partitions for each Spark action, such as the collect action. Spark driver issues a collect() for the whole broadcast data set. The spark default of 1 GB is overridden and increased to 4 GB. This value should suffice most use cases. If the spark driver fails with the following error message, consider increasing this value:

```
Total size of serialized results is bigger than spark.driver.maxResultSize
```

Configure the following advanced property for Spark in the Hadoop connection:

Property	Description
spark.driver.maxResultSize	Set the result size to a size equal to or greater than the driver memory, or 0 for unlimited.

java.util.concurrent.TimeoutException; Futures timed out after [300 seconds].

The default broadcast timeout limit is set to 300 seconds. Increase the SQL broadcast timeout limit.

Configure the following advanced property for Spark in the Hadoop connection:

Property	Description
spark.sql.broadcastTimeout	Set the timeout limit to at least 600 seconds.

A job fails due to Spark speculative execution of tasks.

With spark speculation, the Spark engine relaunches one or more tasks that are running slowly in a stage. To successfully run the job, disable spark speculation.

Configure the following advanced property for Spark in the Hadoop connection:

Property	Description
spark.speculation	Set the value to false.

The Spark driver process hangs.

The Spark driver process might hang due to multiple reasons. The Spark driver process dump and the YARN application logs might not reveal any information to isolate the cause.

The following Informatica Knowledge Base article describes a step-by-step process that you can use to troubleshoot mappings that fail due to the Spark driver process:

HOW TO: Troubleshoot a mapping that fails on the Spark engine when the Spark driver process hangs

ShuffleMapStage 12 (rdd at InfaSprk1.scala:48) has failed the maximum allowable number of times: 4.

The Spark shuffle service fails because the garbage collector exceeded the overhead limit. This forces the Node Manager to shut down, which eventually causes the Spark job to fail.

To resolve this issue, perform the following steps:

1. Open the YARN node manager.
2. In the NodeManager Java heap size property, increase the maximum heap size in MB.

For further debugging, check the Node Manager logs:

```
 java.lang.OutOfMemoryError : GC overhead limit exceeded
 2016-12-0 7 19:38:29,934 FATAL
 yarn.YarnUncaughtExceptionHandler
(YarnUncaughtExceptionHandler.java:uncaughtException 51))-
Thread Thread[IPCServer handler 0on 8040,5,main] threw
an error. Shutting down now...
```

NoRouteToHostException shown in the YARN application master log with the Sequence Generator transformation.

Spark tasks communicate with the Data Integration Service through the Data Integration Service HTTP port to get the sequence range. Ensure that the Data Integration Service is accessible through the HTTP port from all Hadoop Cluster nodes. Spark tasks, including the Sequence Generator transformation, will fail if the HTTP port is not accessible.

The NoRouteToHostException in the YARN application master log indicates that the Data Integration Service HTTP port is not accessible from the Hadoop cluster nodes. The following example shows the NoRouteToHostException in the YARN application master log:

```
18/06/29 14:26:31
WARN TaskSetManager: Lost task 0.0 in stage 0.0 (TID 0,
<DIS_Host_Name>, executor 1):
java.net.NoRouteToHostException: No route to host (Host
unreachable) at
java.net.PlainSocketImpl.socketConnect(Native
Method)
```

MapR Issue: Slowness Due to File Copy from tmp to Target Directory

Consider the following troubleshooting tips for the MapR distribution.

Copying a file from the tmp directory to the target

While a job is in progress, Spark creates an intermediate target directory to which all processing tasks write data. Each task creates its own temporary file in this intermediate folder.

Source splits are determined based on the dfs.blockSize. For example, a 256 MB block size results in 4 splits per GB of data. The number of files written to the tmp directory depends either on the source partitions, in the case of non-shuffle mappings, or on the number of shuffle partitions. Due to a limitation on MapR, whenever there are a large number of partitions, the sequential copying of files from the temporary directory to the target might result in slow performance. The Yarn logs contain a warning message for each copied file.

For example, you might see a warning like the following message:

```
maprfs:///user/hive/warehouse/tpch_text_1000.db/lineitem_tgt/.hive-
staging_hive_2018-12-20_13-13-45_870_7817067372900421111-1/-ext-10000/part-00953-4b1fe48f-
a41b-4525-ae7a-662d08cb5963-c000 to maprfs:/user/hive/warehouse/tpch_text_1000.db/lineitem_tgt/
part-00953-4b1fe48f-a41b-4525-ae7a-662d08cb5963-c000 because HDFS encryption zones are different
```

Amazon EMR Distribution Tips

Consider the following performance tuning and troubleshooting tips for the Amazon EMR distribution.

Tuning Amazon EMR Ephemeral Clusters

When you deploy ephemeral Amazon EMR clusters, consider the following best practices to improve workflow performance:

- Deploy the cluster in the same region and availability zone as the EC2 instance where you deployed the Informatica domain.
- Deploy the cluster with a small number of core EC2 instances. The default number of core EC2 instances is 2. Scale up or scale down depending on your requirements.
- Use Amazon S3 buckets to archive all Hadoop and application logs for future analysis.

Best Practices for Amazon EMR Auto-Scaling

You can enable auto-scaling to dynamically adjust to performance thresholds on the Spark engine. Consider the following best practices for auto-scaling rules:

Implement auto-scaling rules for Spark applications that run for at least 10 minutes.

By default, the AWS CloudWatch auto-scaling evaluation period is 5 minutes. Another 5 to 7 minutes are required to add additional nodes to the cluster. If Spark applications do not run for at least 10 minutes, it is more cost-efficient to disable auto-scaling on the ephemeral cluster.

Always specify MinCapacity and MaxCapacity

MinCapacity and MaxCapacity determine the minimum and maximum number of cluster nodes that auto-scaling rules can create. You can set MinCapacity to the default number of core nodes that are created on the ephemeral cluster.

REST API Mapping Service Tips

Consider the following configuration and performance tuning tips when you use the REST API to run deployed mappings.

Using the REST API from a remote client

When you make a mapping service request from a remote client, ensure that any parameter files required to run the mapping are present on the Data Integration Service node.

Persisting monitoring statistics

When you configure monitoring on the domain, ensure that you use a dedicated Model Repository Service that is only used to persist monitoring statistics.

Running concurrent jobs

When you run concurrent jobs, use the Administrator tool to configure the following properties for the Data Integration Service:

- In the **Properties** tab under **Execution Options**, set *Maximum Hadoop Batch Execution Pool Size* to a value larger than the number of concurrent jobs.
- In the **Processes** tab under **HTTP Configuration Properties**, set the following properties:

Property	Value
Maximum Concurrent Requests	400
Maximum Backlog Requests	400

Tune the Sqoop Parameters

Use Sqoop to process data between relational databases and HDFS through MapReduce programs. You can use Sqoop to import and export data. When you use Sqoop, you do not need to install the relational database client and software on any node in the Hadoop cluster.

Sqoop Command Line Arguments

You can tune certain parameters to optimize performance of Sqoop readers and writers. Add the parameters in the JDBC connection or to Sqoop mappings.

The following table lists the parameters that you can tune:

Parameter	Applies To	Description
batch	Writer	Specifies that you can group the related SQL statements into a batch when you export data.
direct	Reader and writer	Specifies the direct import fast path when you import data from a relational source. **Note:** Applies to Oracle and TDCH connectors.
Dsqoop.export.records.per.statement	Writer	Specifies to insert multiple rows with a single statement.
Enable primary key	Reader	Enables the primary key constraint on the source table to optimize performance when reading data from a source.
fetch-size	Reader	Specifies the number of entries that Sqoop can import at a time.
num-mappers	Reader and writer	Specifies the number of map tasks that can run in parallel.
compress or z	Reader	Enables compression.
Dmapreduce.map.java.opts	Reader	Specifies the Java options per statement if Java runs out of memory.
Dmapred.child.java.opts	Reader	Specifies the Java options per mapper if Java runs out of memory.

Sqoop Tuning Guidelines

The following table lists the tuning recommendations for Sqoop parameters based on the data volume of the deployment types:

Parameter	Sandbox or Basic Deployment	Standard Deployment	Advanced Deployment (Default)
fetch-size	1000	10000	20000
num-mappers	4	10	20
Dsqoop.export.records.per.statement	1000	10000	10000
Dmapreduce.map.java.opts	1 GB	2 GB	4 GB
Dmapred.child.java.opts	1 GB	2 GB	4 GB

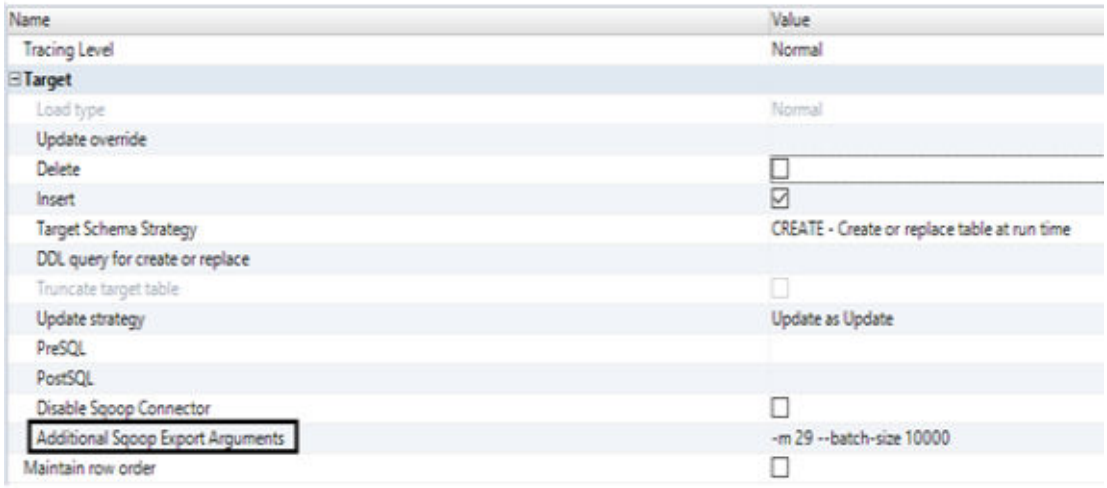
Tune the TDCH for Sqoop Parameters

When you read data from or write data to Teradata, you can use Teradata Connector for Hadoop (TDCH) specialized connectors for Sqoop. When you run Sqoop mappings on the Blaze and Spark engines, you can configure the Cloudera Connector Powered by Teradata and Hortonworks Connector for Teradata.

The following table lists the parameters that you can tune:

Parameter	Applies To	Description
batch.insert	Writer	This Teradata target plugin associates an SQL JDBC session with each mapper in the TDCH job when loading a target table in Teradata.
internal.fastexport	Reader	This Teradata source plugin associates a FastExport JDBC session with each mapper in the TDCH job to retrieve data from the source table in Teradata.
internal.fastload	Writer	This Teradata target plugin associates a FastLoad JDBC session with each mapper in the TDCH job when loading a target table in Teradata.
split.by.amp	Reader	The connector creates one mapper per available Teradata AMP, and each mapper subsequently retrieves data from each AMP. As a result, no staging table is required.
split.by.hash	Reader	This input method is similar to the split.by.partition method. Instead of directly operating on value ranges of one column, this method operates on the hash of the column. Use this method to extract data in situations where split.by.value and split.by.partition are not appropriate.
split.by.partition	Reader	This method is preferred to extract a large amount of data from the Teradata system. Behavior of this method depends on whether the source table is partitioned or not.
split.by.value	Reader	This method creates input splits as ranges on the split by column, which is typically the table's primary key. Each split is subsequently processed by a single mapper to transfer the data using SELECT queries.

The following image shows the additional Sqoop arguments that you specify at the mapping level:



Tuning Guidelines

Consider the following guidelines when you tune internal.fastload, internal.fastexport, and batch.insert methods:

- internal.fastLoad. Informatica has a restriction on number of sessions that can be opened at a time to write to Teradata. Use the following formula to determine the number of sessions with an upper limit of 100 sessions per job:

  ```
  If number of AMPs <= 20, then use 1 per AMP.

  If number of AMPs > 20, then use (20 + (Number of AMPs / 20))
  ```

- internal.fastexport. Uses 1 session per AMP with an upper limit of 4 sessions per job. If the number of mappers specified is more than the max number of sessions that can be opened, TDCH restricts the mappers to max sessions.
- batch.insert. If the number of AMPs is high such as 170, 180, or more, then Informatica observed that the performance of the batch.insert method is better.

TDCH for Sqoop Tuning Tips

Consider the following tuning tips for Sqoop TDCH:

- If you use an IP address in the connection, all connections connect to a single server which can be a bottleneck.
- COP stands for Communications Processor. COP Discovery refers to the process of performing multiple DNS lookups to identify all the Teradata Database nodes that the client software could potentially connect to. Using cop address allows connections to round-robin over cop servers to connect each mapper to a different node. For example:
  Teradatacop1 or teradatacop1.domain.com

  Teradatacop2 or teradatacop2.domain.com

  Teradatacop3 or teradatacop3.domain.com
- Laddered Concurrent Connect (LCC) occurs second, and uses the two-dimensional array as the list of possible IP addresses to choose from. The purpose of LCC is to make multiple TCP socket connect attempts in parallel. LCC provides 3-4x faster throughput with parallel connections.

TDCH for Sqoop Import and Export Guidelines

Spark job scales linearly during Sqoop import and export. You can tune Spark jobs based on cluster resources. Configure the following advanced properties for Spark in the Hadoop connection:

spark.executor.instances=<number of executor instances>

The following formula determines the total running containers:

```
Total running containers = (Number of cores) x (Number of executor instances)
```

The Spark engine uses 2 executor instances by default. So, only 4 containers run in parallel. For better performance, fine tune the spark.executor.instances property.

Tune the Oracle Database

To optimize the performance of Oracle databases, perform the following tasks:

- Analyze database statistics to fine tune queries.
- Maintain different physical disks for different tablespaces.
- Determine the expected database growth.

- Use the EXPLAIN PLAN statement to fine tune queries.
- Avoid foreign key constraints.
- Drop indexes before loading data.
- Set open cursors and sessions optimally for mappers to process queries in parallel.

Tune the PostgreSQL Database

Perform configuration tasks to use a PostgreSQL database for the Model repository.

Ensure that the PostgreSQL installation has sufficient disk space for data files. By default, the files are in the directory `<PostgreSQL installation folder>/data`.

Ensure that the following properties are set in the PostgreSQL configuration file:

- Set *max_connections* to 4000.
- Set *shared_buffers* to 16GB.
- Set *max_locks_per_transaction* to 1024.

Case Studies

Refer to the following case studies for a general idea on the performance numbers.

Case Study: Application Deployment

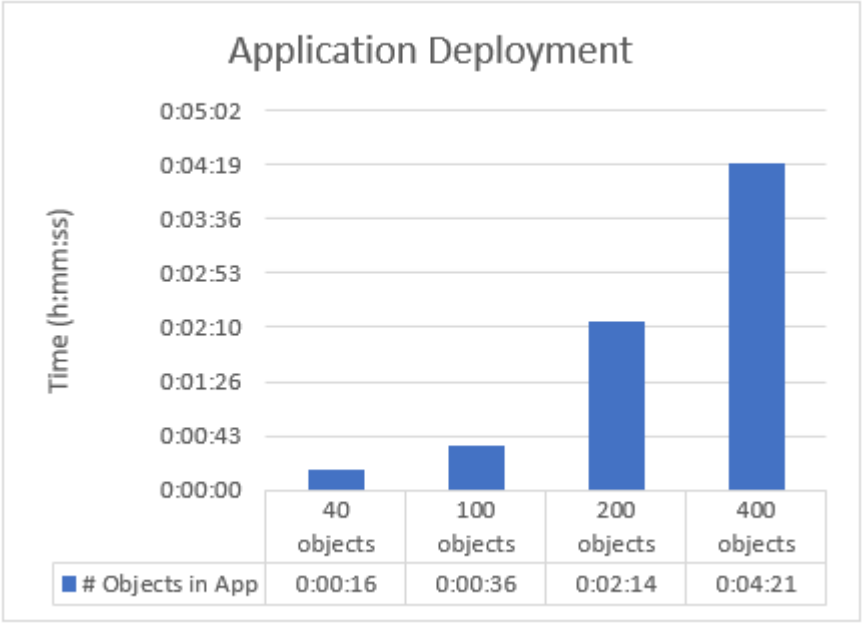
Application deployment requires communication between the Data Integration Service and the associated Model Repository Service. The Data Integration Service fetches the object and writes to the database schema of the Model Repository Service.

Environment

Chipset	Intel® Xeon® CPU E5-2680 v3 @ 2.50GHz
Cores	48 cores
Memory	132 GB
Operating system	Red Hat Enterprise Linux 7.6

Performance Chart

The following chart shows the time taken to deploy applications with a different number of objects:



Conclusions

Based on the case study, consider the following best practices for application deployment when you have a similar configuration:

Number of objects in applications

Having a large number of objects in the same application is not desirable. It increases the deployment time. It also increases the resource usage of the Data Integration Service and the Model Repository Service. Distributing objects in an optimal manner between various applications is key to achieve better performance.

Recommendation: ~ 50 objects per application.

Workaround for incremental application deployment

To deploy the changes made to the application, you stop the application and redeploy it. This process causes downtime. Applications must be designed to minimize the effects on this downtime. If the number of objects in the application are less, the effect of the downtime will be less severe.

Recommendation: The number of objects in an application should not exceed ~ 60 objects per application.

Cursor requirements on the database

The process of application deployment needs to use cursors at the database layer (the schema associated with the Model Repository service). If applications are designed to be too large (1000+ of objects within) or with deep hierarchy of objects, the cursor usage will be greater at the database level.

Recommendation: Required number of cursors >= Number of objects in application.

Case Study: Data Integration Service Application Load and Start-Up

The following case study observes the load and start-up of deployed applications on the Data Integration Service.

Test Setup

The study compares the following types of applications:

- 40-object applications. Each application contains 40 application objects that include 32 mappings and 8 physical data objects.
- 100-object applications. Each application contains 100 application objects that include 50 mappings and 50 physical data objects.

The applications are deployed in sets of 100, 500, or 1,000 applications to the Data Integration Service.

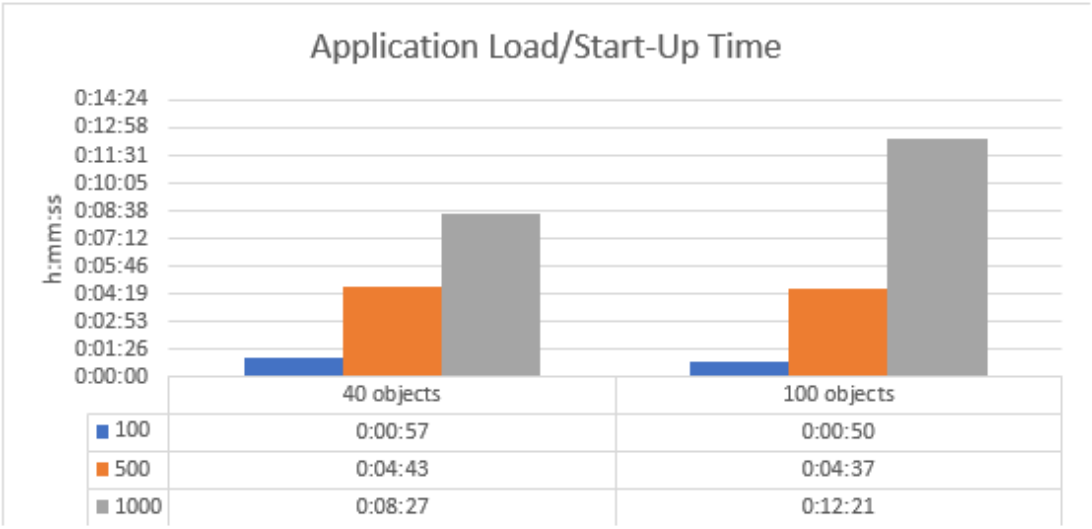
The database used by the Model Repository Service is deployed on the same machine as the domain node.

Environment

Chipset	Intel® Xeon® CPU E5-2680 v3 @ 2.50GHz
Cores	12 cores
Memory	132 GB
Operating system	Red Hat Enterprise Linux server release 7.6

Performance Chart

The following chart shows the time taken to start up the Data Integration Service based on the number of applications deployed to the Data Integration Service and the number of objects in each application:



Conclusions

The total number of objects in an application has little to no effect on application load/startup time. To reduce the number of deployed applications on a Data Integration Service, group related entities to place 40 - 60 application objects that include mappings and workflows in one application.

Case Study: Data Integration Service Concurrency

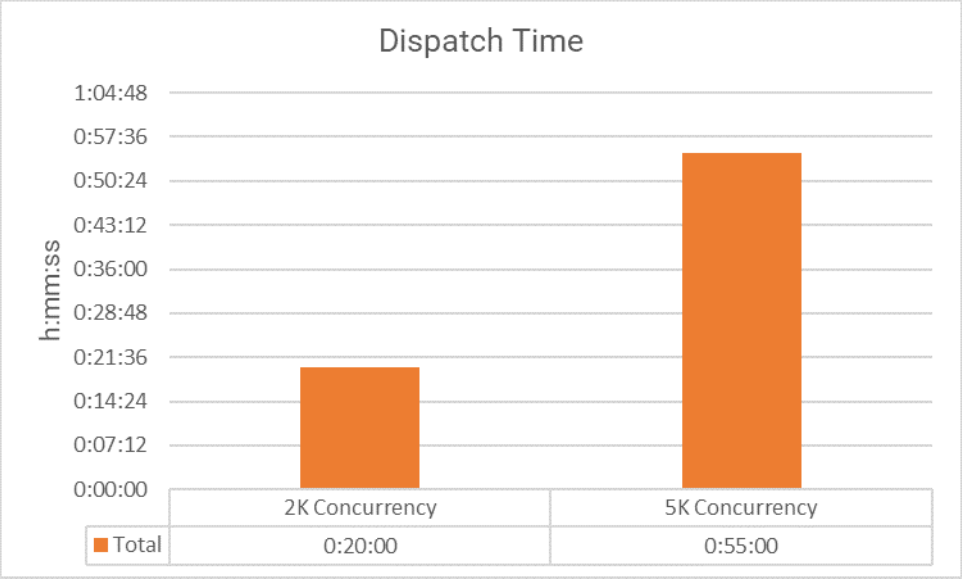
The following case study tested a large number of concurrent mappings running on a single node Data Integration Service. The mappings used TPC-DS benchmark queries of medium complexity with Hive sources and parameterized HDFS targets. During the test, peak CPU utilization was ~20 cores (80%) for less than 5 minutes. The average utilization was ~8 cores.

Environment

Chipset	Intel® Xeon® Processor X5675 @ 3.06 GHz
Cores	4 x 6 cores
Memory	32 GB
Operating system	Red Hat Enterprise Linux 6.1
Hadoop distribution	Cloudera 5.15
Hadoop cluster	25 nodes

Performance Chart

The following performance chart compares the dispatch times for 2K and 5K concurrent jobs. Dispatch time is the time taken by DIS to submit all the mappings to the cluster:



Conclusions

- When you submit concurrent mapping requests, use the infacmd gateway service to optimize performance. For more details, refer to the following Informatica Knowledge Base article:

  Gateway Service to Submit Mappings and Workflows to the Data Integration Service

Case Study: Java String Port Conversion Overhead

The following case study compares performance between string ports and numeric ports and evaluates the UTF conversion overhead for port processing.

Test Setup

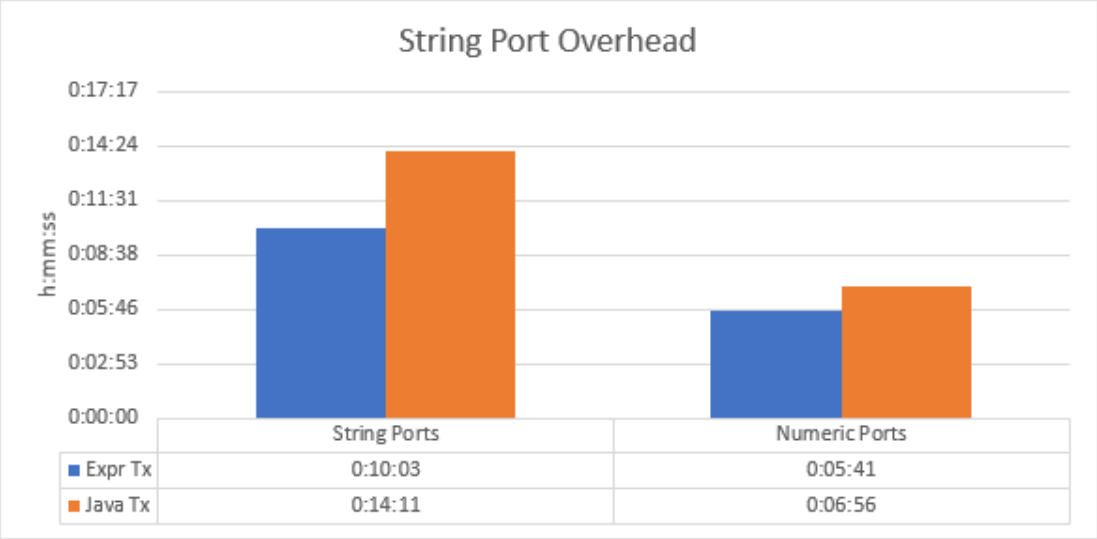
The study compares one operation on 25 string ports to one operation on 25 numeric ports. The operation is performed in the Java transformation. For additional comparison, the same operation is performed in the Expression transformation.

Environment

Chipset	Intel® Xeon® CPU E5-2650 v4 @ 2.20GHz
Cores	12 cores
Memory	256 GB
Operating system	Red Hat Enterprise Linux 7.2
Hadoop cluster	12 nodes
String ports	7 ports
Data volume	~ 750 GB

Performance Chart

The following performance chart compares the time taken for the operation to process the string and numeric ports in the Java transformation and the Expression transformation:



Conclusions

Based on the case study, the Java transformation experiences an overhead cost of around 10% for numeric ports. For string ports, the same overhead cost is around 40%.

Case Study: Data Integration Service Concurrency with Multiple HS2 Load Balancers

The following case study shows the benefits of having multiple Hive Server2 Load Balancers for large concurrent mappings running on a Data Integration Service 4 node grid.

The mappings used TPC-DS benchmark queries of medium complexity with Hive sources and parameterized HDFS targets. During the test, peak CPU utilization was ~20 cores (80%) for less than 5 minutes. The average utilization was ~4 cores.

Environment

	Cloudera Cluster	**Data Integration Service 4 Node Grid**
Chipset	Intel® Xeon® Processor X5675 @ 3.06 GHz	Intel® Xeon® Gold 6132 CPU @ 2.60GHz
Cores	4 x 6 cores	4 x 14 cores
Memory	32 GB	125 GB
Operating System	Red Hat Enterprise Linux 6.1	Red Hat Enterprise Linux 7.5 (Maipo)
Hadoop Distribution	Cloudera 5.15	–
Hadoop Cluster Size	25 nodes	–

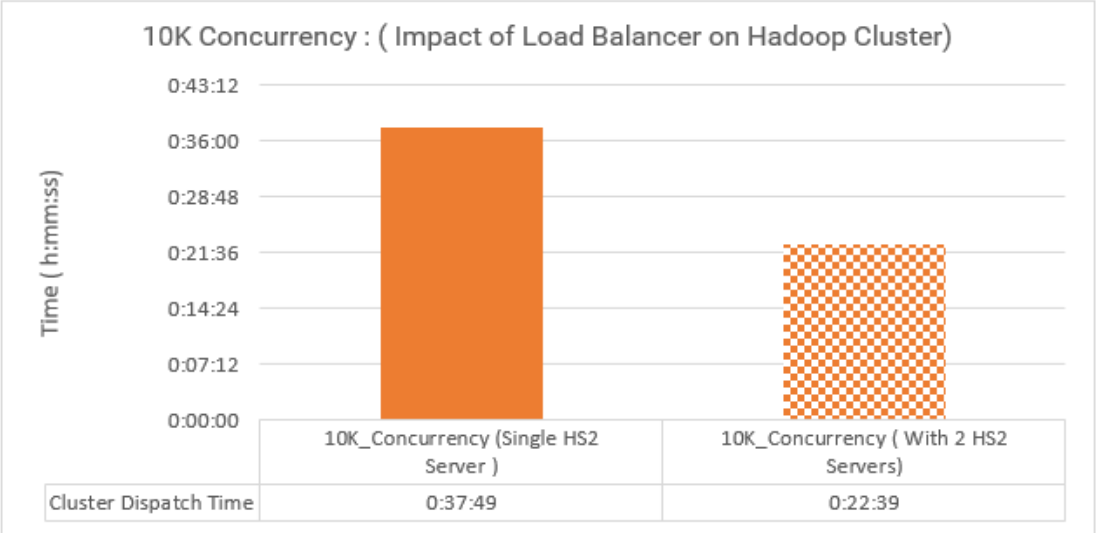
Hive Server 2 Load Balancer Configuration

Test staff configured the HiveServer 2 load balancers using the following steps:

- Install the HA Proxy package or another load balancer recommended by your IT team.
- Configure the HA proxy service to listen on port 10000 and include the HS2 instances.
- Configure the HA Proxy service to start on bootup.
- In Cloudera Manager, include the Load Balancer server address in the HiveServer2 Load Balancer configuration properties.
- Restart the Hive service.

Performance Chart

The following performance chart compares the dispatch times for 10K concurrent jobs on a Hadoop cluster. Cluster Dispatch time is the time taken by the Data Integration Service to submit all mappings to the cluster:



Conclusions

The test found that dispatch time improved ~40% with two HiveServer 2 instances.

Case Study: Traditional Update Strategy versus Hive MERGE

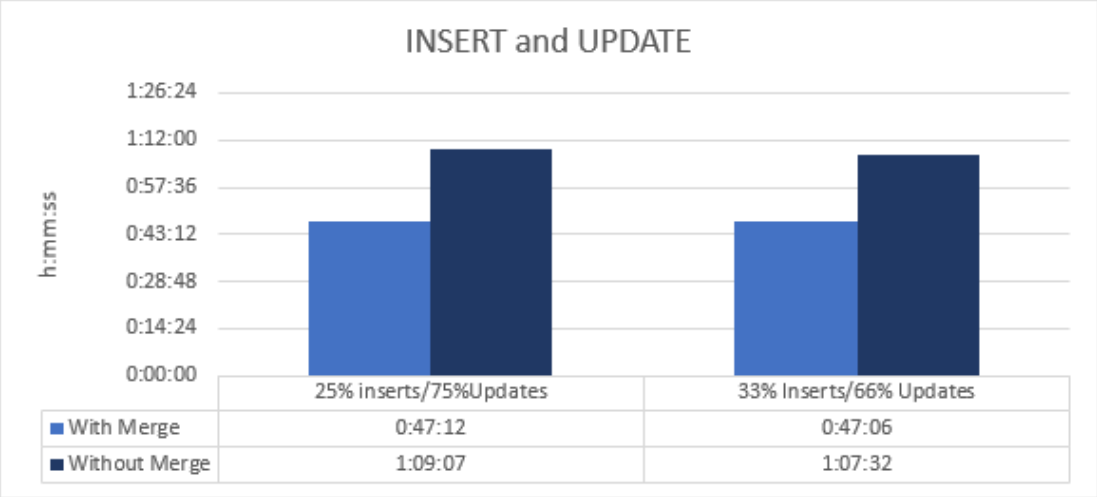
The following case study compares the amount of time that an update strategy task requires to complete depending on the number of INSERT and UPDATE statements in the task and whether the task implements Hive MERGE.

Environment

Chipset	Intel® Xeon® CPU E5-2680 v3 @ 2.50GHz
Cores	12 cores
Memory	128 GB
Operating system	Red Hat Enterprise Linux 7.3
Hadoop cluster	14 nodes
Data volume	~ 75 GB

Performance Chart

The following performance chart shows a comparison of the time taken for an update strategy task to complete when the task contains a combination of INSERT and UPDATE statements:



Conclusions

Based on the case study, a 30% increase in performance is observed in the update strategy task when the task implements Hive MERGE.

When you perform incremental updates and the percentage of UPDATE statements is higher than the percentage of INSERT statements, consider using Hive MERGE.

Case Study: Python Transformation

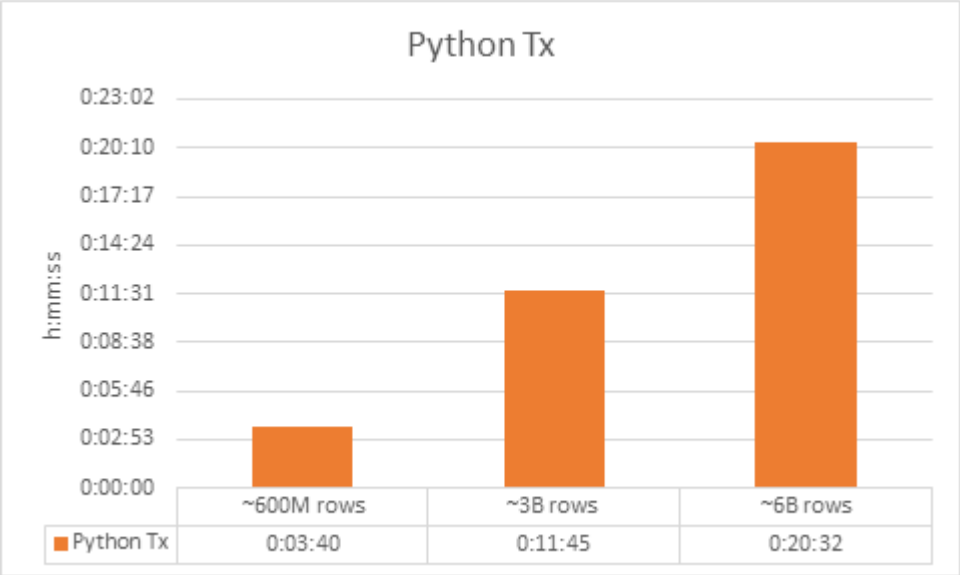
The following case study compares performance of a mapping containing a Python transformation handling data of various sizes.

Environment

Chipset	Intel® Xeon® CPU E5-2680 v3 @ 2.50GHz
Cores	12 cores
Memory	128 GB
Operating system	Red Hat Enterprise Linux 7.3
Hadoop cluster	12 nodes
Data volume	100 GB –– 1 TB

Performance Chart

The following performance chart compares performance of a mapping containing a Python transformation handling data of various sizes:



Case Study: Sqoop TDCH Export and Import

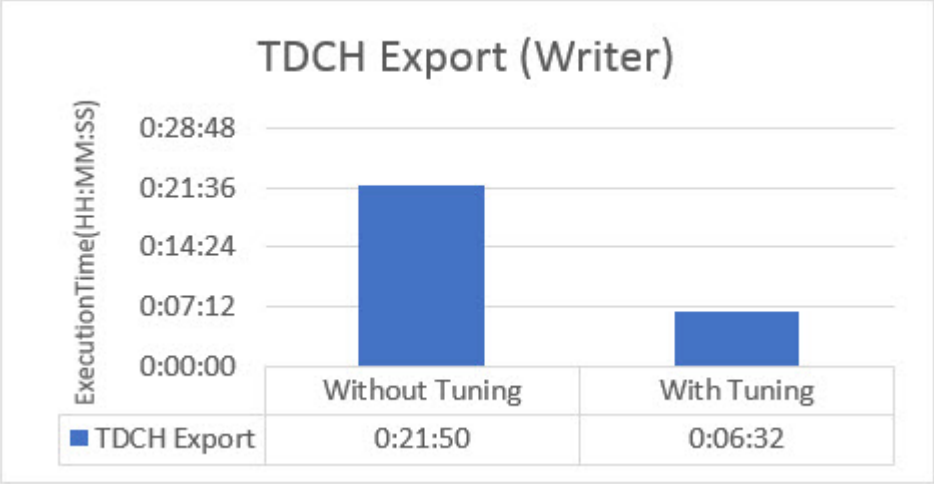
The following case study uses a simple pass-through mapping that reads data from Teradata and writes to HDFS or Hive using TDCH for Sqoop. It also reads data from HDFS or Hive and writes to Teradata.

Environment

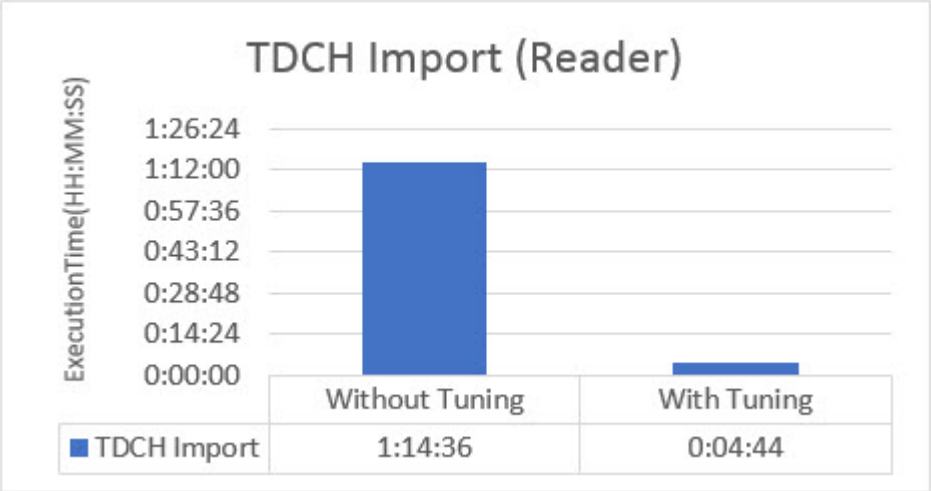
Chipset	Intel® Xeon® Processor X5675 @ 3.2 GHz
Cores	2 x 6 cores
Memory	256 GB
Operating system	Red Hat Enterprise Linux 7.0
Hadoop distribution	Cloudera Enterprise 5.11.1
Hadoop cluster	7 nodes
Data set	TPC-H.Lineitem SF-10, ~7.5 GB. 16 Col, 600 Million Rows, Row Size- ~405 Bytes

Performance Chart

The following chart shows the execution time for TDCH export:



The following chart shows the execution time for TDCH import:



Conclusions

- For the Sqoop writer, the number of mappers increased from the default 4 to 144. With the maximum session restriction for the internal.fastLoad method, the actual sessions created were 25.
- For the Sqoop reader, the number of mappers increased from the default 1 to 144. Default value is 1 because the table has a primary key defined. When the number of mappers increase, set the value of the `spark.executor.instances` property equal to the number of mappers for optimal performance.

Case Study: Sqoop Oracle Import (Reader)

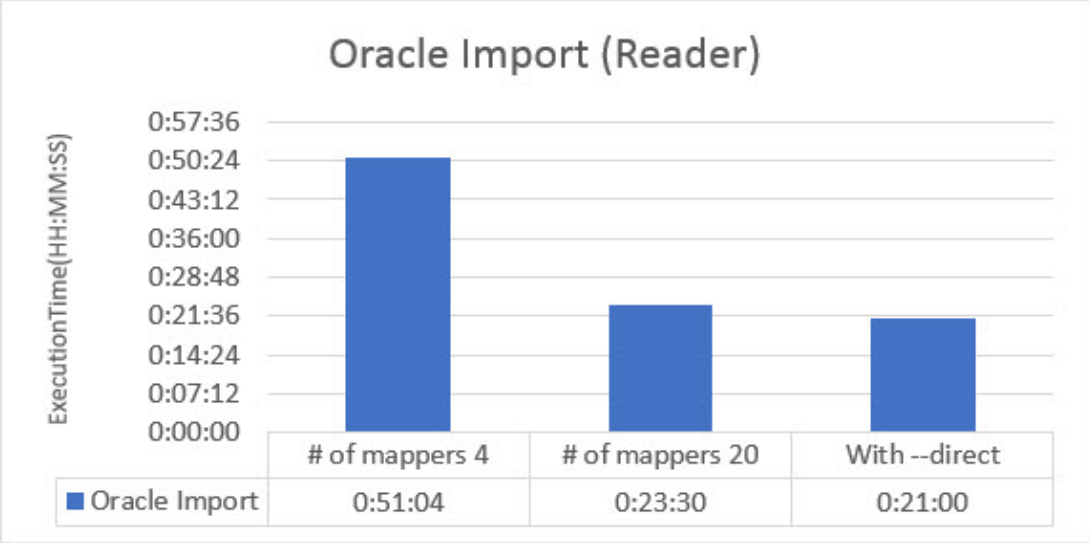
The following case study uses a simple pass-through mapping that reads data from Oracle source and writes to HDFS using Sqoop.

Environment

Chipset	Intel® Xeon® Processor X5675 @ 3.2 GHz
Cores	2 x 6 cores
Memory	256 GB
Operating system	Red Hat Enterprise Linux 7.0
Hadoop distribution	Cloudera Enterprise 5.11.1
Hadoop cluster	7 nodes
Data set	TPC-H.Lineitem SF-100, ~75 GB. 16 Col, 6 Billion Rows, Row Size- ~405 Bytes

Performance Chart

The following chart shows the execution time for Oracle import:



Case Study: Sqoop Oracle Export (Writer)

The following case study uses a simple pass-through mapping that reads data from HDFS and writes to Oracle using Sqoop.

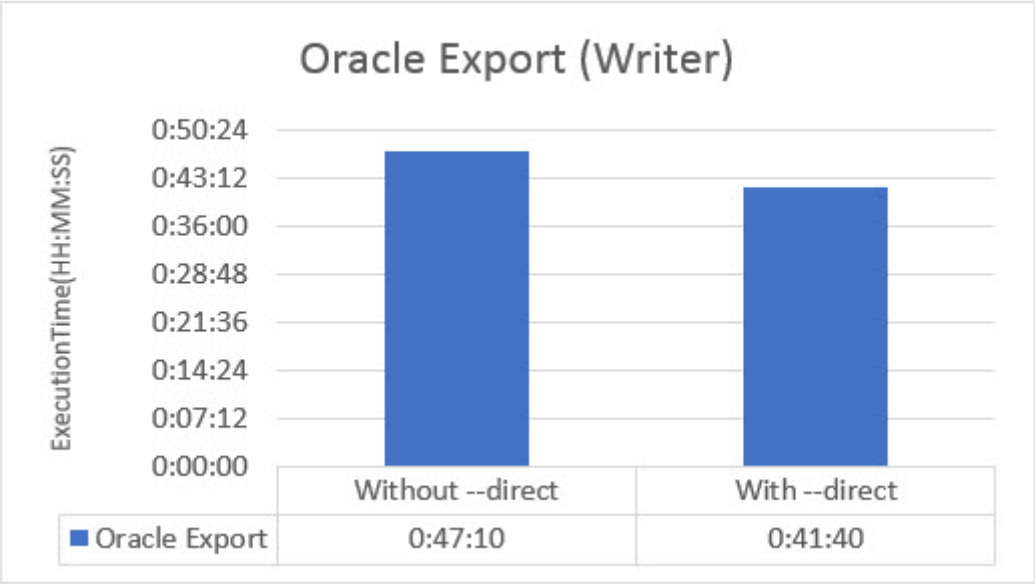
Environment

Chipset	Intel® Xeon® Processor X5675 @ 3.2 GHz
Cores	2 x 6 cores
Memory	256 GB
Operating system	Red Hat Enterprise Linux 7.0

Hadoop distribution	Cloudera Enterprise 5.11.1
Hadoop cluster	7 nodes

Performance Chart

The following chart shows the execution time for Oracle export:



Recommendations

Consider the following recommendations:

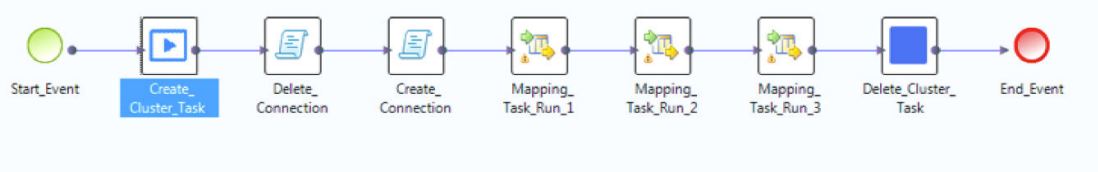
- Design mappings to enable sources and targets to handle requests in parallel.
- Verify that the cluster has adequate resources to process data in parallel. Passing a large number of mappings does not improve the performance linearly if cluster resources are inadequate.

Case Study: Amazon EMR Auto-Scaling

The following case study analyzes the performance of a cluster workflow that is deployed to an ephemeral Amazon EMR cluster.

Test Setup

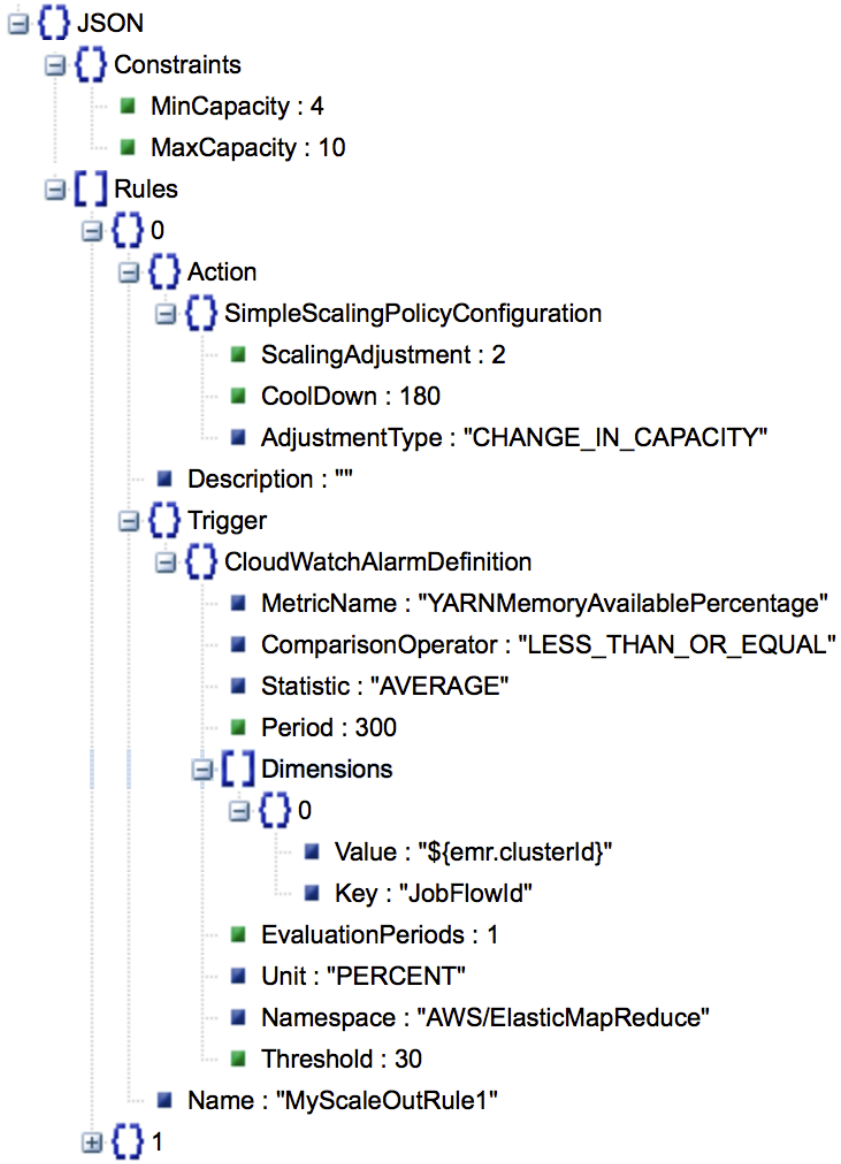
The study uses the following cluster workflow:



The cluster workflow contains three mapping tasks, *Task 1*, *Task 2*, and *Task 3*. The mapping tasks are identical mappings that process the same data volume. The tasks are executed sequentially.

The workflow is submitted to an ephemeral Amazon EMR cluster that employs auto-scaling. The auto-scaling rules have a minimum capacity set to 4 nodes and a maximum capacity set to 10 nodes. The auto-scaling rules add nodes to the cluster in increments of 2.

The following image shows the auto-scaling policy:



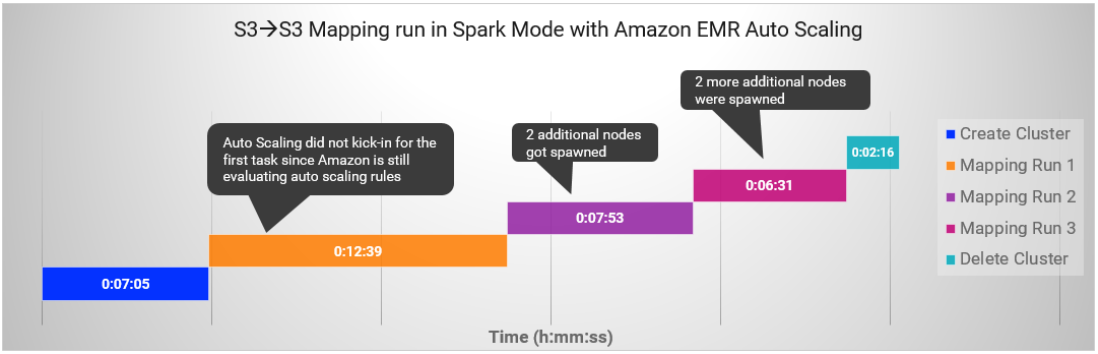
Environment

Chipset	Intel® Xeon® Processor X5675 @ 3.2 GHz
Cores	2 x 6 cores
Memory	256 GB
Operating system	Red Hat Enterprise Linux 7.0

Hadoop distribution	Cloudera Enterprise 5.11.1
Hadoop cluster	7 nodes

Performance Chart

The following performance chart shows a timeline of the workflow execution on the Amazon EMR cluster:



The performance chart shows the following events:

- Creating an ephemeral cluster took 7 minutes.
- The first mapping task ran on 4 data nodes and took around 13 minutes to complete.
- While the first task ran, the auto-scaling policy was prompted to add 2 data nodes to the cluster, but it took around 8 minutes for the additionally commissioned nodes to be available for processing.
- By the time that the additional nodes became available, the execution of the first mapping task was completed.

The second mapping task leveraged the additional nodes and ran on 6 data nodes. The additional computational resources reduced the execution run time from around 13 minutes to around 8 minutes.

Conclusions

Auto-scaling rules for ephemeral clusters on AWS can be defined in multiples of 5 minutes. Once the auto-scaling rules are prompted, it can take between 5 to 10 minutes for additionally commissioned nodes to become available.

Based on these observations, Informatica recommends that you implement auto-scaling rules only when processing large volumes of data, such as Spark applications that run mapping jobs over multiple iterations.

Authors

Angela Rae

Mark Pritchard