

## Performance Tuning and Sizing Guidelines for Informatica Big Data Management® 10.2.1

## Abstract

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

## Supported Versions

- Big Data Management® 10.2.1

## Table of Contents

Overview. . . . .	3
Big Data Management Deployment Types. . . . .	3
Deployment Criteria. . . . .	3
Deployment Type Comparison. . . . .	4
Sizing Recommendations. . . . .	5
Hadoop Cluster Hardware Recommendations. . . . .	5
Informatica Domain Sizing . . . . .	6
Big Data Streaming Sizing and Tuning Recommendations. . . . .	7
Tune the Hardware and the Hadoop Cluster. . . . .	8
Tune the Informatica Domain and Application Services. . . . .	8
Analyst Service. . . . .	9
Content Management Service. . . . .	9
Data Integration Service. . . . .	9
Mass Ingestion Service. . . . .	11
Model Repository Service. . . . .	12
Search Service. . . . .	12
Tune the Blaze Engine. . . . .	13
Mapping Optimization. . . . .	13
Transformation Optimization. . . . .	15
Filter Optimization for Hive Sources. . . . .	17
Tune the Spark Engine. . . . .	19
Spark Configuration. . . . .	19
Transformation Optimization. . . . .	20
Troubleshooting Spark Job Failures. . . . .	23
Tune the Sqoop Parameters. . . . .	25
Sqoop Command Line Arguments. . . . .	26
Sqoop Tuning Guidelines. . . . .	26
Tune the TDCH for Sqoop Parameters. . . . .	27
TDCH for Sqoop Tuning Tips. . . . .	28

TDCH for Sqoop Import and Export Guidelines. . . . .	28
Tune Ephemeral Clusters on Amazon EMR. . . . .	28
Amazon EMR Auto-Scaling Rules. . . . .	29
Tune the Oracle Database. . . . .	29
Case Studies. . . . .	29
Case Study: Model Repository Service Upgrade. . . . .	29
Case Study: Application Deployment. . . . .	30
Case Study: Data Integration Service Application Load and Start-Up. . . . .	32
Case Study: Data Integration Service Concurrency. . . . .	33
Case Study: Java String Port Conversion Overhead. . . . .	33
Case Study: Traditional Update Strategy versus Hive MERGE. . . . .	34
Case Study: Sqoop TDCH Export and Import. . . . .	35
Case Study: Sqoop Oracle Import (Reader). . . . .	36
Case Study: Sqoop Oracle Export (Writer). . . . .	37
Case Study: Amazon EMR Auto-Scaling. . . . .	38
Documentation Reference. . . . .	40

## Overview

Tuning Big Data Management for better performance includes tuning the Hadoop environment and the Informatica domain environment.

You can tune Big Data Management in the following areas:

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

## Big Data Management 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 Big Data Management into the following types:

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

## Deployment Criteria

The following criteria determine the Big Data Management deployment type:

### Number of active users

Number of users working on the Model repository at design time, using the Analyst tool, or running Big Data Management 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 Big Data Management 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	20 -500	500 - 1000	> 1000
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 (for batch processing use cases)	10 GB	100 GB	500 GB	1 TB +
Messages processed per second (for streaming use cases with Informatica Big Data 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 Big Data Management, you can use `infacmd autotune autotune` to automatically tune certain properties in your environment.

For more information, see the "Tuning for Big Data Processing" chapter in the *Informatica Big Data Management Administrator Guide*.

## Sizing Recommendations

Based on your Big Data Management 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 <sup>1</sup>	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)
<sup>1</sup> 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: <code>\${yarn.nodemanager.local-dirs}/usercache/\${user}/appcache/application_\${appid}</code> . You can find the work directories of individual containers, <code>container_\${contid}</code> , 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](#).

## Informatica Domain Sizing

### Deployment Types

The following table contains guidelines for choosing the deployment type for the domain:

Deployment Type	Use Case	Typical configuration
Sandbox	Proof of concepts or as a sandbox environment with minimal users	16 cores, 32 GB RAM, and about 50 GB disk space
Basic	Low volume processing, low levels of concurrency	24 cores, 64 GB RAM, and about 100 GB disk space

Deployment Type	Use Case	Typical configuration
Standard	High volume processing, low levels of concurrency	Multi-node setups configured with 64 GB RAM, more than 100 GB disk space per node, and 48 cores across nodes
Advanced	High volume processing, high levels of concurrency	Multi-node setups configured with 128 GB RAM, more than 100 GB disk space per node, and 96 cores across nodes

### Minimum Requirements

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

Deployment Type	Total CPU Cores <sup>1</sup>	Total System Memory	Disk Space Per Node <sup>2</sup>
Sandbox	16	32 GB	50 GB
Basic	24	32 GB	100 GB +
Standard	36	64 GB	100 GB +
Advanced	96	128 GB	100 GB +
<sup>1</sup> CPU cores are physical cores.			
<sup>2</sup> 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.

## Big Data Streaming Sizing and Tuning Recommendations

Use Informatica Big Data 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.

For more sizing and tuning recommendations, refer to the following Informatica How-To Library article:

[Sizing Guidelines and Performance Tuning for Big Data Streaming 10.2.1](#)

## 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

For more information, refer to the following Informatica How-to Library article:

[Tuning the Hardware and Hadoop Clusters for Informatica Big Data Products](#)

## 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	16,000	26,000	32,000	64,000
Max user processes	1024	3000	8000	12000

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	640 MB	2 GB	2 - 4 GB	4 - 6 GB
Execution pool size for the Hadoop environment	10	500	1000	2000+
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.

## Best Practices for Handling High Concurrent Workloads

Consider the following best practices to handle high concurrent workloads on the Data Integration Service:

**Configure the Data Integration Service on a node with at least 12 CPU cores that are dedicated to the service.**

To run 2,000 concurrent mappings of medium complexity that contain 10 - 15 transformations, the Data Integration Service requires around 8 - 12 CPU cores and 8 GB of JVM heap memory.

**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.

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 properties:

Property	Value
enable_client_gateway	true
num_max_threads*	1000
memory	4096m

\* Change the value based on the maximum concurrency requirements. Note that the execution pool size for the Hadoop environment must be at least as large as the maximum number of threads. For example, if you configure 1000 maximum threads, set the execution pool size to at least 1000.

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.

For more information, see the "Tuning for Profiling Performance," "Tuning Profile Warehouse," and "Profile Warehouse Guidelines for Column Profiling" sections in the following Informatica How-to Library article:

[Tuning Enterprise Data Catalog Performance](#)

## Mass Ingestion Service

Mass ingestion is a big data 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

Run no more than 500 concurrent Sqoop jobs on a Hadoop cluster. To limit the number of concurrent Sqoop jobs, set the following property on the Data Integration Service:

Property	Value
Maximum Hadoop Batch Execution Pool Size	500

### 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 Model Repository Service and the heap size required to store the monitoring configuration in the Model repository.

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	1 GB	2 GB	4 GB
Max heap size for monitoring	1 GB	1 GB	2 GB	4 GB

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

### Additional Guidelines

Consider the following additional guidelines for the Model Repository Service:

- Create a separate Model Repository Service for 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. Prior to 10.2.1, when Aggregator and Rank transformations contained many string ports with precision as large as 4000, mapping execution was slow and eventually failed due to inadequate disk space.

In 10.2.1, 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:

Name	Type	Precision
<b>All Ports (5)</b>		
o_custkey	decimal	100
o_orderstatus	integer	10
o_totalprice	decimal	10
o_orderprio...	string	100
o_clerk	string	4000

Name	Type	Length/Precision
<b>All Ports (5)</b>		
o_custkey	decimal	100
o_orderstatus	string	10
o_totalprice	decimal	10
o_orderprio...	string	15
o_clerk	string	100

#### 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:

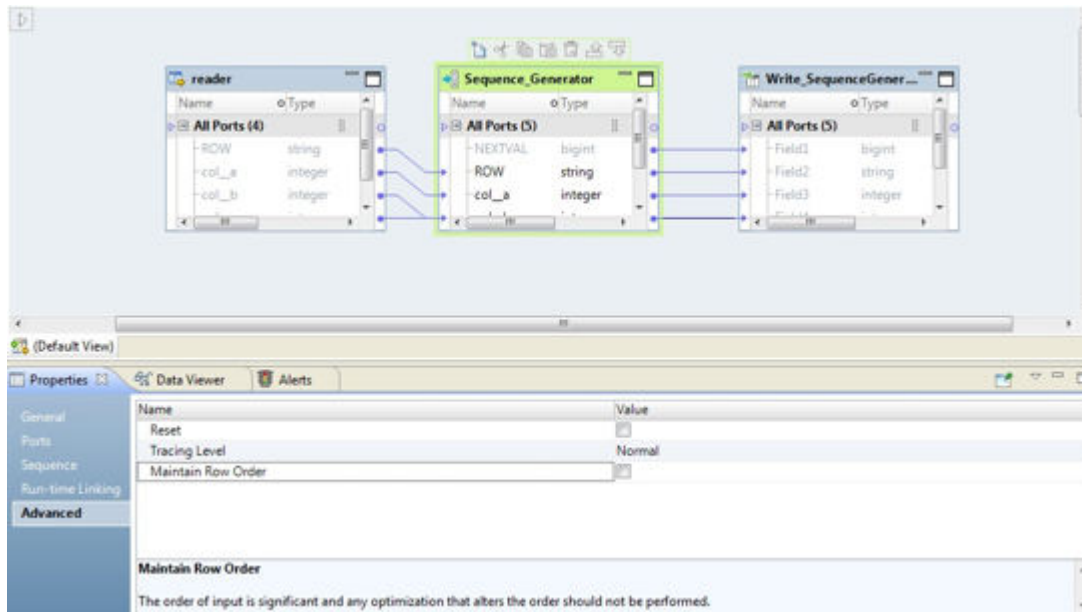
- <Transformation type> Data Cache Size
- <Transformation type> Index Cache Size

## 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.

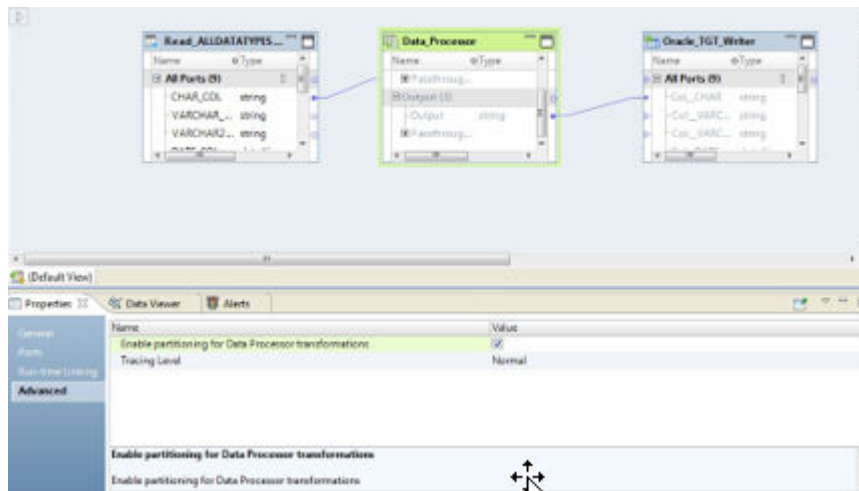
### Data Processor Transformation

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:
  - In the Developer tool, open the mapping and select the Data Processor transformation.
  - 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

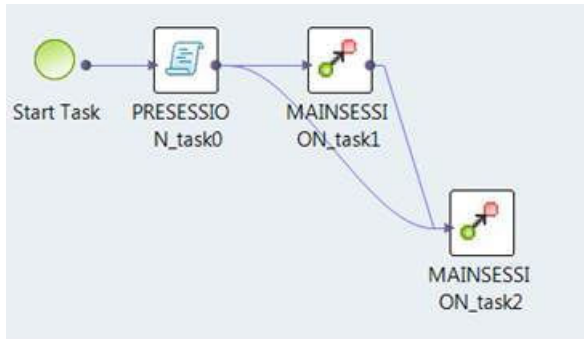
## Filter Optimization for Hive Sources

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:

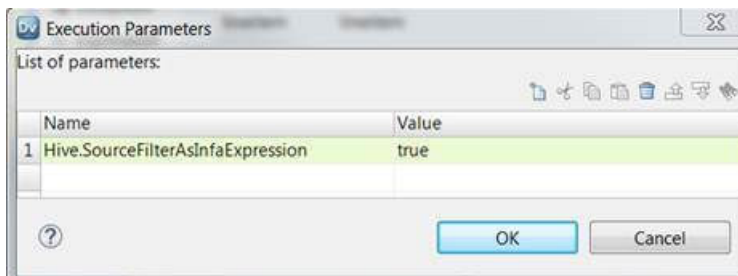


To avoid the performance overhead, set the following custom flag as an advanced property of the mapping:

```
Hive.SourceFilterAsInfraExpression = 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.
- In the Hadoop connection to run mappings on the Spark engine, set up the Spark History Server, Spark HDFS staging directory, and Spark event log directory.

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

For more information about configuring the Hadoop connection, refer to the *Informatica Big Data Management User Guide*.

### Spark Configuration

The properties for the Spark engine are tuned in the Hadoop connection.

The values are tuned for advanced deployment types. 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 <sup>1</sup> (8 * per GB data at shuffle)	800	800	4000	8000
spark.driver.memory <sup>2</sup>	1 GB	2 GB	4 GB	4 GB + (default)
spark.driver.maxResultSize	1 GB	1 GB	2 GB	4 GB + (default)

#### <sup>1</sup> Infaspark.shuffle.max.partitions

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

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

#### <sup>2</sup> 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).

## 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 33](#)

### 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.

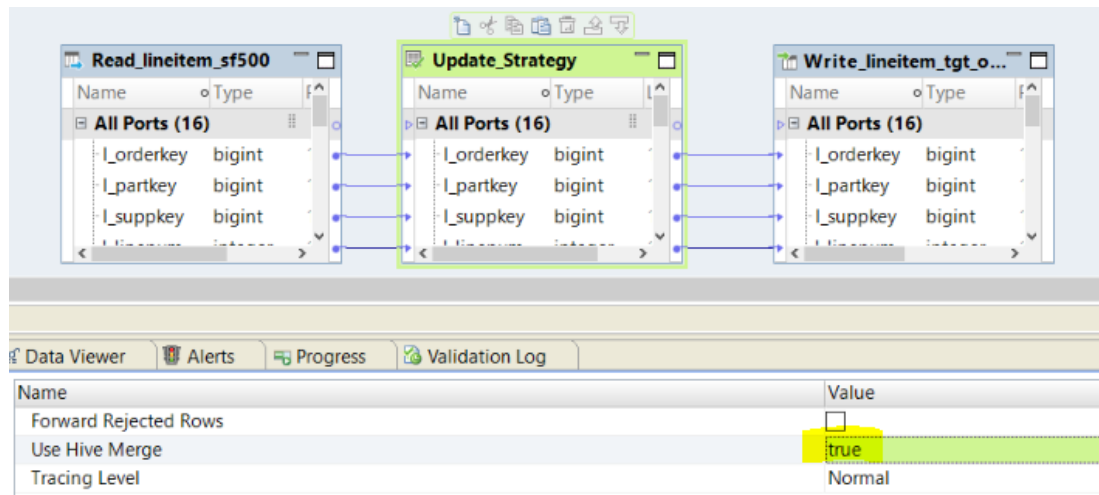
### Lookup Transformation

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.

## 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 23](#).

## 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:

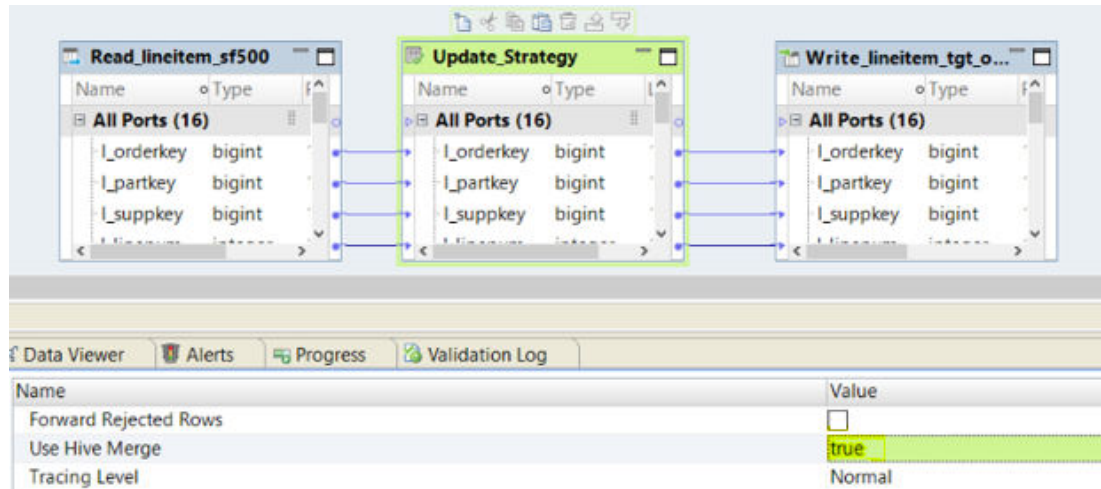
- Cloudera CDH does not endorse ORC file formats and ACID-enabled tables.
- Updates are not compatible with partitioned tables on Amazon EMR due to a Hive issue.

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 34.](#)

### Hive MERGE Run-time Failures

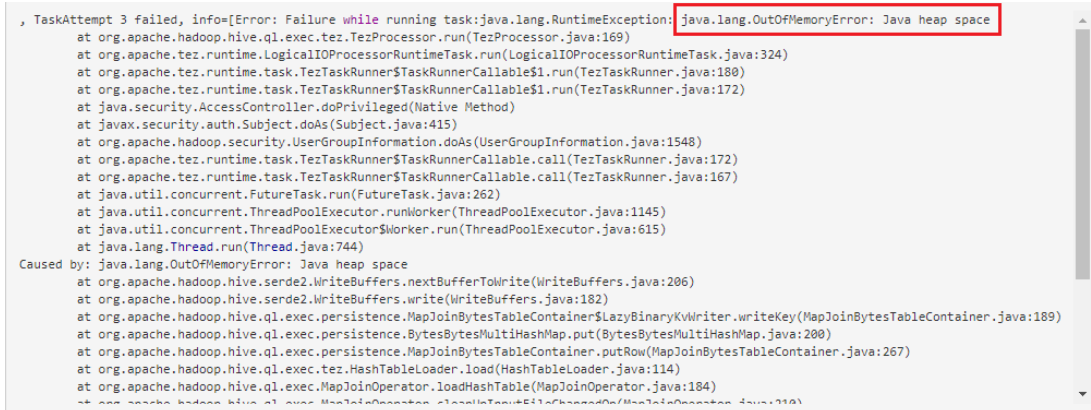
When you use Hive MERGE statements to perform update strategy tasks, the mapping might fail at run time.

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:



```
, TaskAttempt 3 failed, info=[Error: Failure while running task:java.lang.RuntimeException: java.lang.OutOfMemoryError: Java heap space
    at org.apache.hadoop.hive.q1.exec.tez.TezProcessor.run(TezProcessor.java:169)
    at org.apache.tez.runtime.LogicalIOProcessorRuntimeTask.run(LogicalIOProcessorRuntimeTask.java:324)
    at org.apache.tez.runtime.task.TezTaskRunner$TaskRunnerCallable$1.run(TezTaskRunner.java:180)
    at org.apache.tez.runtime.task.TezTaskRunner$TaskRunnerCallable$1.run(TezTaskRunner.java:172)
    at java.security.AccessController.doPrivileged(Native Method)
    at javax.security.auth.Subject.doAs(Subject.java:415)
    at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1548)
    at org.apache.tez.runtime.task.TezTaskRunner$TaskRunnerCallable.call(TezTaskRunner.java:172)
    at org.apache.tez.runtime.task.TezTaskRunner$TaskRunnerCallable.call(TezTaskRunner.java:167)
    at java.util.concurrent.FutureTask.run(FutureTask.java:262)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
    at java.lang.Thread.run(Thread.java:744)
Caused by: java.lang.OutOfMemoryError: Java heap space
    at org.apache.hadoop.hive.serde2.WriteBuffers.nextBufferToWrite(WriteBuffers.java:206)
    at org.apache.hadoop.hive.serde2.WriteBuffers.write(WriteBuffers.java:182)
    at org.apache.hadoop.hive.q1.exec.persistence.MapJoinBytesTableContainer$LazyBinaryKvWriter.writeKey(MapJoinBytesTableContainer.java:189)
    at org.apache.hadoop.hive.q1.exec.persistence.BytesBytesMultiHashMap.put(BytesBytesMultiHashMap.java:200)
    at org.apache.hadoop.hive.q1.exec.persistence.MapJoinBytesTableContainer.putRow(MapJoinBytesTableContainer.java:267)
    at org.apache.hadoop.hive.q1.exec.tez.HashTableLoader.load(HashTableLoader.java:114)
    at org.apache.hadoop.hive.q1.exec.MapJoinOperator.loadHashTable(MapJoinOperator.java:184)
    at org.apache.hadoop.hive.q1.exec.MapJoinOperator.closeTask(HashMapOperator.java:110)
```

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 10. 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, refer to the following Informatica How-To Library article:

[Strategies for Incremental Updates on Hive in Big Data Management 10.2.1](#)

## 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)
```

## 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. <b>Note:</b> 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-mapper	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 Big Data Management deployment types:

Parameter	Sandbox or Basic Deployment	Standard Deployment	Advanced Deployment (Default)
fetch-size	1000	10000	20000
num-mapper	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:

Name	Value
Tracing Level	Normal
<b>Target</b>	
Load type	Normal
Update override	
Delete	<input type="checkbox"/>
Insert	<input checked="" type="checkbox"/>
Create or replace table at runtime	<input type="checkbox"/>
DDL query	
Truncate target table	<input type="checkbox"/>
Update strategy	Update as Update
PreSQL	
PostSQL	
Disable Sqoop Connector	<input type="checkbox"/>
Additional Sqoop Export Arguments	-m 29 -- --output-method internal.fastload --batch-size 10000
Maintain row order	<input type="checkbox"/>

## 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:  
`Teratacop1` or `teratacop1.domain.com`  
`Teratacop2` or `teratacop2.domain.com`  
`Teratacop3` or `teratacop3.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 Ephemeral Clusters on Amazon EMR

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 the Informatica server runs.

- 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 as necessary.
- Use Amazon S3 buckets to archive all Hadoop and application logs for future analysis.

## Amazon EMR Auto-Scaling Rules

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.

## 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.

## Case Studies

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

### Case Study: Model Repository Service Upgrade

Model Repository Service upgrade time depends on the number of objects in the repository. Larger backup file size means more time to upgrade. A minimum of 4GB heap memory is required for Model Repository Service during the upgrade process.

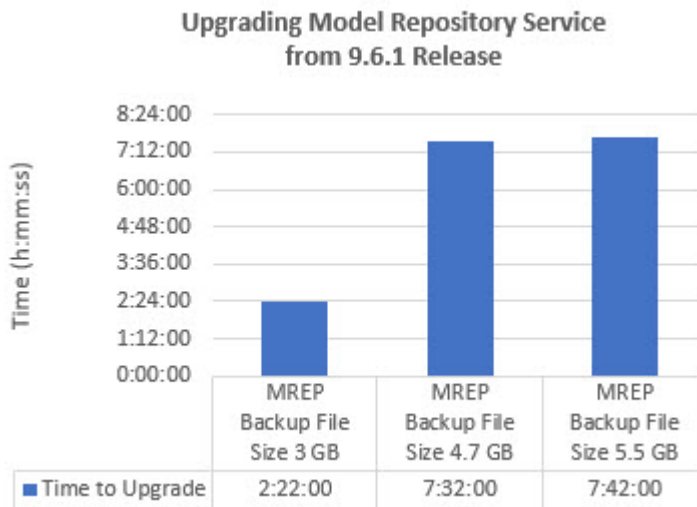
#### Environment

Chipset	Intel® Xeon® CPU E5-4650 0 @ 2.70GHz
Cores	32 cores

Memory	128 GB
Operating system	Red Hat Enterprise Linux 6.5

### Performance Chart

The following chart shows the time taken to upgrade Model Repository Services from Big Data Management 9.6.1 to 10.2.1:



### Case Study: Application Deployment

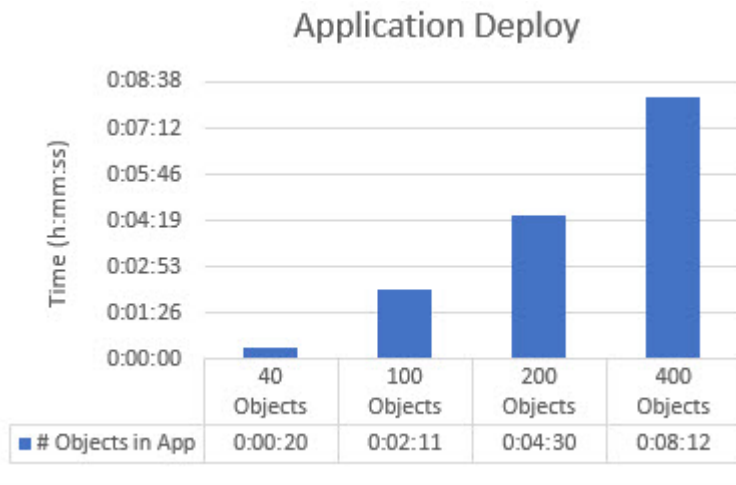
Application deployment requires communication between the Data Integration Service and the associated Model Repository Service. The Data Integration Service fetches the object and writes to the database schema of the Model Repository Service.

#### Environment

Chipset	Intel® Xeon® CPU E5-4650 0 @ 2.70GHz
Cores	32 cores
Memory	128 GB
Operating system	Red Hat Enterprise Linux 6.5

## 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. Thus, an application must not be designed with too many objects within.

**Recommendation:** ~ 50 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  $\geq$  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.

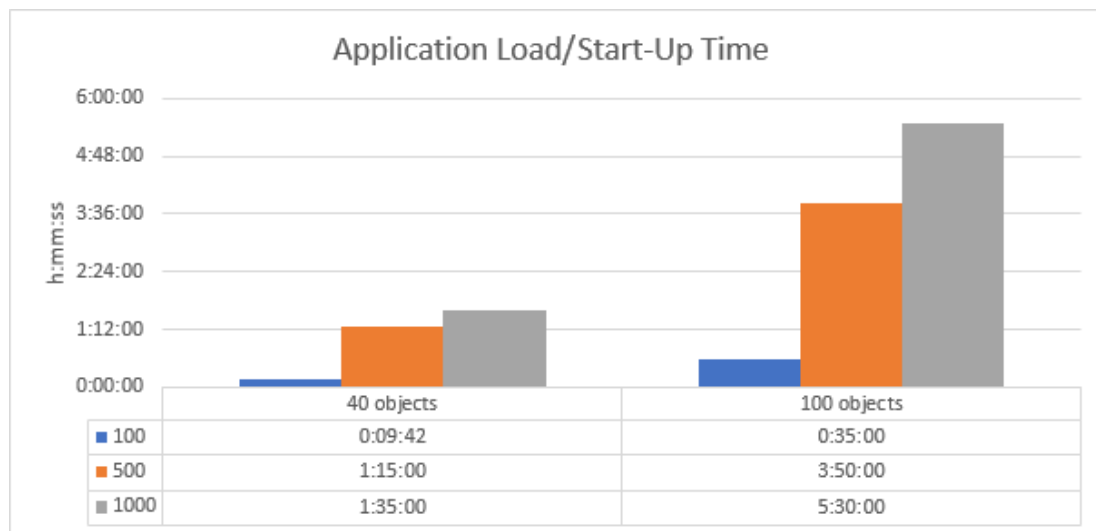
The database used by the Model Repository Service is deployed on the same machine as the domain node.

### Environment

Chipset	Intel® Xeon® CPU E5-2680 v3 @ 2.50GHz
Cores	12 cores
Memory	132 GB
Operating system	Red Hat Enterprise Linux 7.2

### 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

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.

When you run a deployed application, the application must be loaded and initialized. If the Data Integration Service is still loading or initializing a deployed application, mapping execution requests against the application will fail. Note that design-time mappings are not affected. You can continue to perform design-time operations such as creating new mappings or running data preview.



## Case Study: Data Integration Service Concurrency

A single Data Integration Service in Informatica Big Data management can handle hundreds and thousands of concurrent pushdown mappings. The following case study shows that the Data Integration Service with 8 GB heap memory can dispatch up to **2000** concurrent mappings of TPC-DS benchmark queries of medium complexity.

### Environment

Chipset	Intel® Xeon® Processor X5675 @ 3.06 GHz
Cores	2 x 6 cores
Memory	32 GB
Operating system	Red Hat Enterprise Linux 6.7
Hadoop distribution	Hortonworks HDP 2.6
Hadoop cluster	25 nodes

### 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

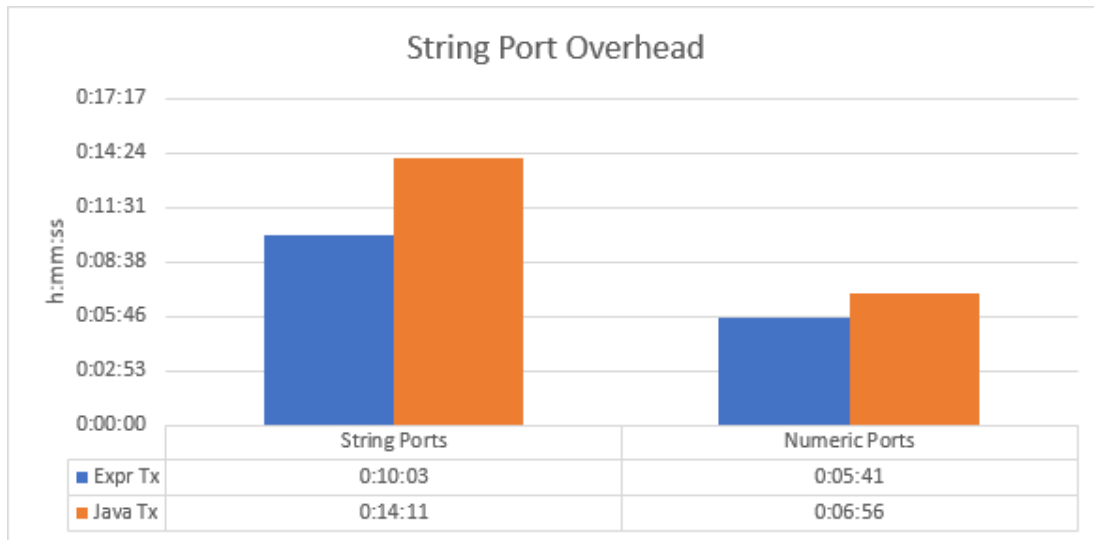
The study compares one operation on 25 string ports to one operation on 25 numeric ports. The operation is performed in the Java transformation. For additional comparison, the same operation is performed in the Expression transformation.

### 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: Traditional Update Strategy versus Hive MERGE

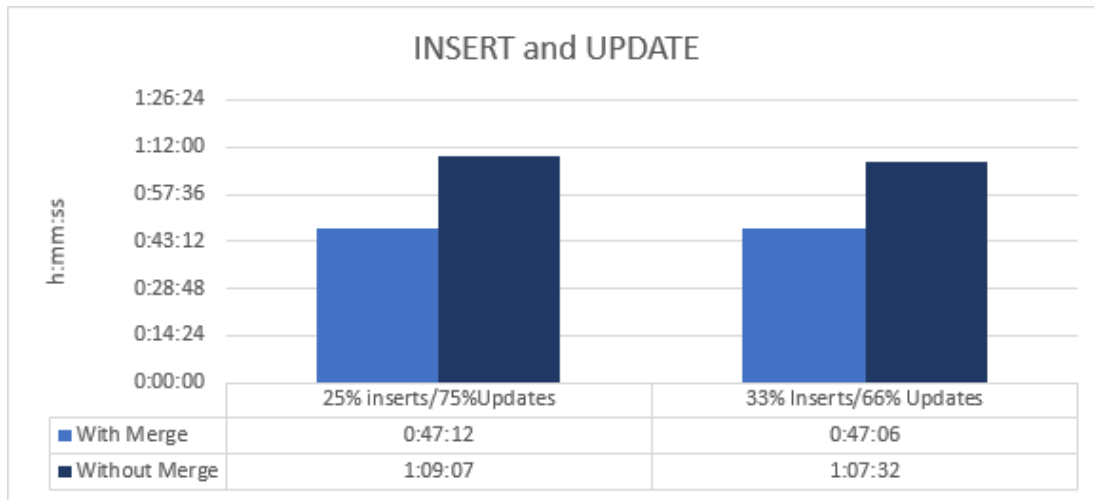
The following case study compares the amount of time that an update strategy task requires to complete depending on the number of INSERT and UPDATE statements in the task and whether the task implements Hive MERGE.

#### 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: Sqoop TDCH Export and Import

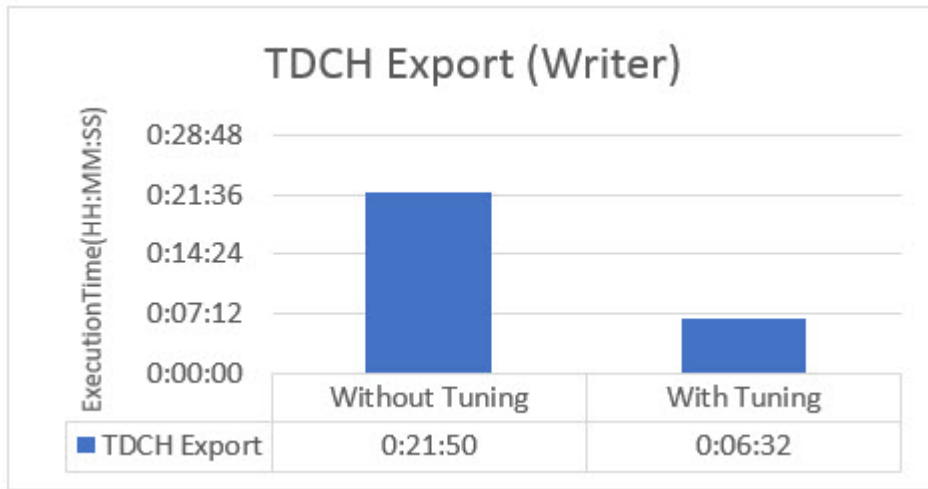
The following case study uses a simple pass-through mapping that reads data from Teradata and writes to HDFS or Hive using TDCH for Sqoop. It also reads data from HDFS or Hive and writes to Teradata.

## Environment

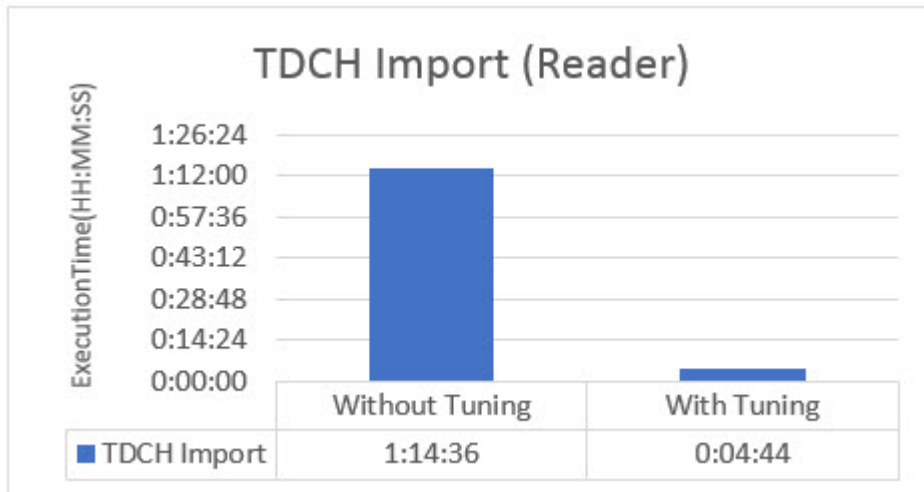
Chipset	Intel® Xeon® Processor X5675 @ 3.2 GHz
Cores	2 x 6 cores
Memory	256 GB
Operating system	Red Hat Enterprise Linux 7.0
Hadoop distribution	Cloudera Enterprise 5.11.1
Hadoop cluster	7 nodes
Data set	TPC-H.Lineitem SF-10, ~7.5 GB. 16 Col, 600 Million Rows, Row Size- ~405 Bytes

## 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)

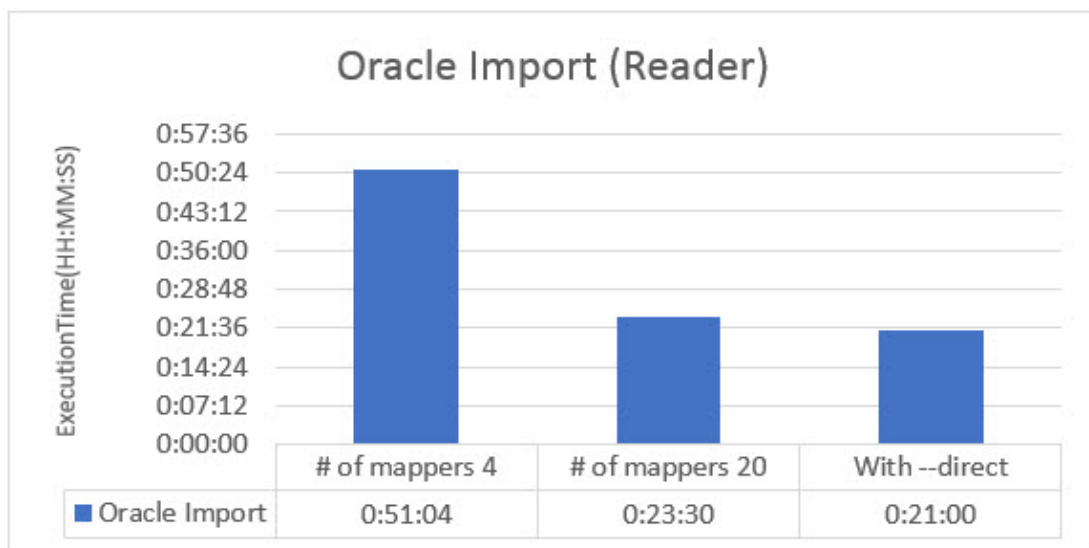
The following case study uses a simple pass-through mapping that reads data from Oracle source and writes to HDFS using Sqoop.

## 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.

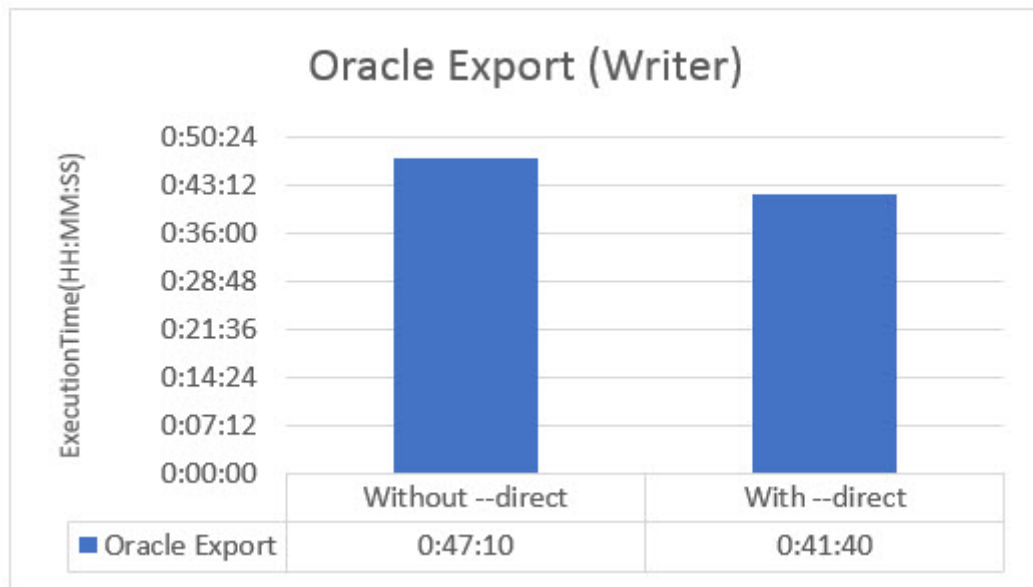
## Environment

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:



### Conclusions

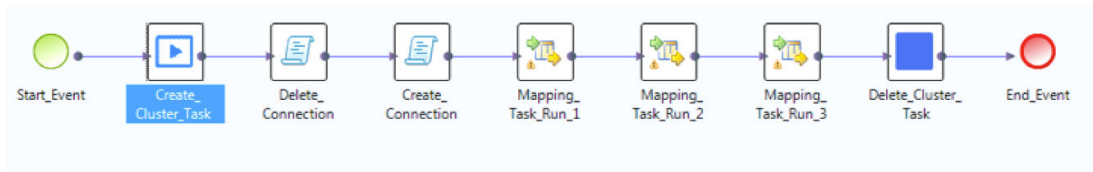
Use mappers in a way that the sources and targets can handle the request in parallel. Also, make sure that the cluster has adequate resources to process data in parallel. Passing a large number of mappers does not improve the performance linearly if the cluster resources are inadequate.

### 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

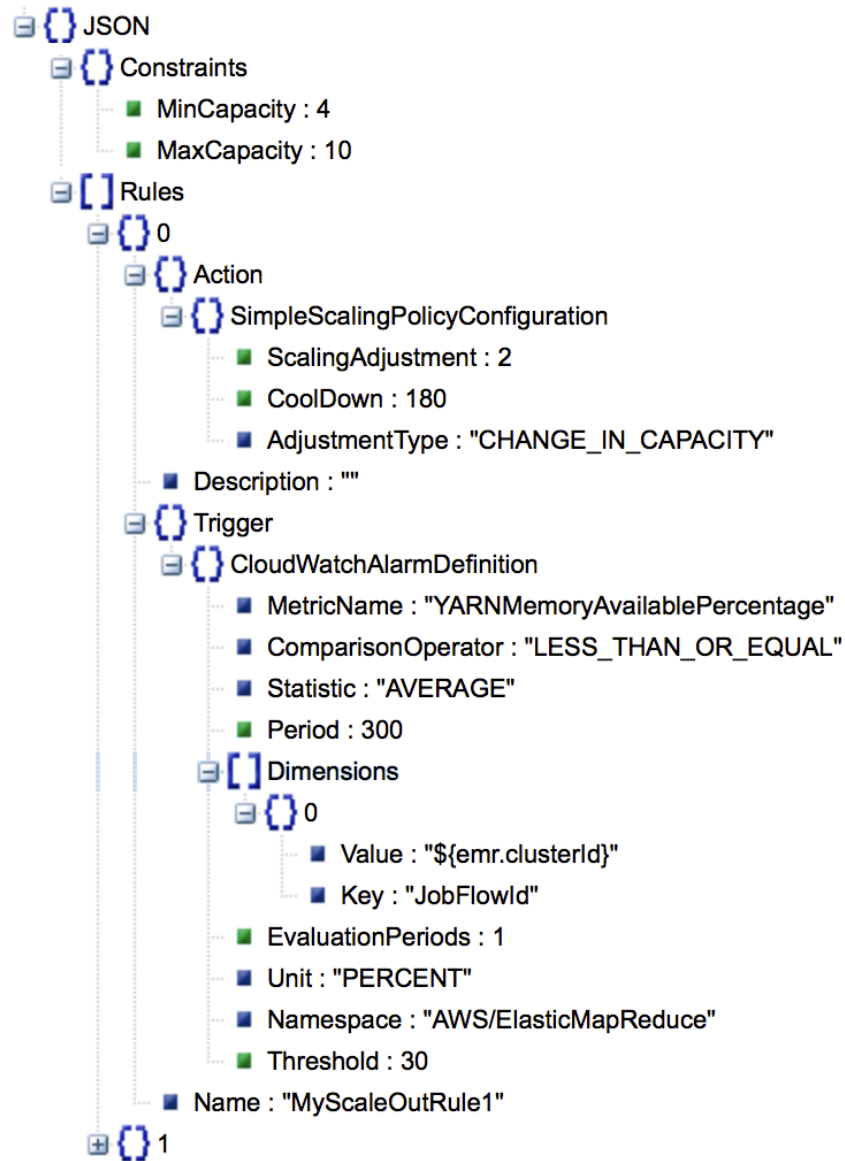
The study uses the following cluster workflow:



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:

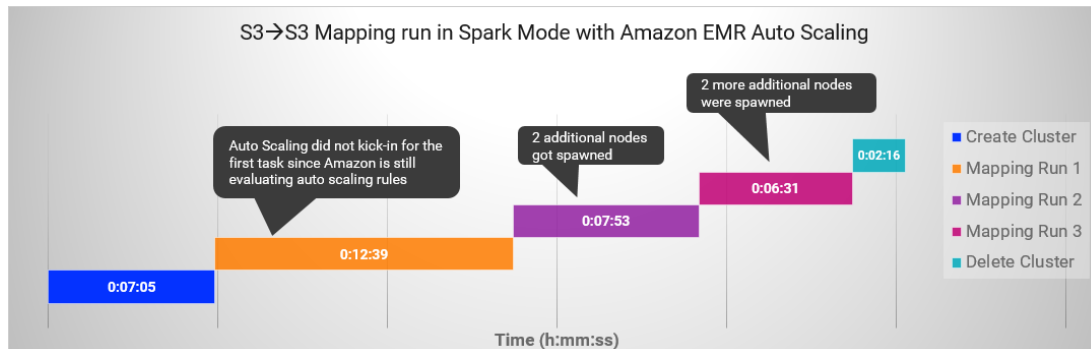


## Environment

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:



As you can see in the performance chart, 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, it is recommended to implement auto-scaling rules only when processing large volumes of data, such as Spark applications that run mapping jobs over multiple iterations.

## Documentation Reference

The following table lists performance-related How-To Library articles for Informatica big data products:

Article	Description
<b>Informatica big data products</b>	
Tuning the Hardware and Hadoop Clusters for Informatica Big Data Management <a href="#">[10.2 &amp; 10.2.1]</a>	Provides tuning recommendations for the hardware and the Hadoop cluster for better performance of Informatica big data products.
<b>Big Data Management</b>	
Performance Tuning and Sizing Guidelines for Big Data Management <a href="#">[10.2]</a> <a href="#">[10.2.1]</a>	Provides sizing recommendations for the Hadoop cluster and the Informatica domain, tuning recommendations for various Big Data Management components, best practices to design efficient mappings, troubleshooting tips, and case studies.
Tuning the Hive Engine for Big Data Management <a href="#">[10.2 &amp; 10.2.1]</a>	Provides tuning recommendations to run mappings on the Hive engine, best practices to design efficient mappings, and case studies.



Article	Description
Strategies for Incremental Updates on Hive in Big Data Management <a href="#">[10.2]</a> <a href="#">[10.2.1]</a>	Describes alternative solutions to the Update Strategy transformation for updating Hive tables to support incremental loads.
Big Data Management 10.2.1 on Microsoft Azure: Architecture and Best Practices <a href="#">[10.2.1]</a>	Provides sizing recommendations for the Hadoop cluster and the Informatica domain in a cloud or hybrid deployment of Big Data Management on the Microsoft Azure cloud platform.
<b>Enterprise Data Catalog</b>	
Tuning Enterprise Data Catalog Performance <a href="#">[10.2.1]</a>	Provides recommendations for tuning metadata ingestion, profiling, databases, and search.
<b>Enterprise Data Lake</b>	
Sizing and Tuning Guidelines for Enterprise Data Lake <a href="#">[10.2]</a> <a href="#">[10.2.1]</a>	Provides sizing recommendations and tuning guidelines for ingesting data, previewing data assets, adding data assets to projects, managing projects, publishing projects, searching for data assets, exporting data assets, and profiling data.
<b>Big Data Streaming</b>	
Sizing Guidelines and Performance Tuning for Big Data Streaming <a href="#">[10.2]</a> <a href="#">[10.2.1]</a>	Provides sizing recommendations and tuning guidelines for Informatica Big Data Streaming.

## Authors

Vishal Kamath

Shruti Buch

Neha Rajendra Velhankar

Mohammed Morshed

Vrushank J Doshi

Indra Sivakumar

Margarita Pelyushenko