How-To Library

Informatica

Tuning the Hive Engine for Big Data Management®

Abstract

You can tune the Hive engine to optimize performance of Big Data Management®. This article provides tuning recommendations for various Big Data Management components, best practices to design efficient mappings, and case studies. 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 - 10.2.1

Table of Contents

Overview

You can tune the Big Data Management components to improve the performance of Hadoop MapReduce jobs. You can configure performance tuning parameters for various components such as the Hadoop cluster, Hive, Data Integration Service, and mapping.

YARN and MapReduce Settings for MapReduce Version 2 (MRv2)

You can tune parameters at the Hadoop cluster level to improve performance. MapReduce version 2 uses YARN which relies on multiple parameters to determine the number of parallel containers.

Based on internal tests, Informatica recommends the following formula to determine the number of containers:

```
Mp
 = Number of Parallel Map Task Containers
 = min of (4 x number of disks, number of logical cores)

Rp
 = Number of Parallel Reduce Task Containers
 = min of (65% of number of logical cores, 3 x number of Disks)
```

After you determine the values for M_p and R_p, you can modify certain parameters to ensure that the Hadoop node can allocate the parallel containers. To modify the parameters in the site.xml files, open the Administrator tool, go to the Connections tab, and select the cluster configuration.

Configure the following properties in the yarn-site.xml:

yarn.nodemanager.resource.memory-mb	The amount of physical memory in MB that can be allocated for containers. Informatica recommends reserving some memory for other processes running in a node.
yarn.nodemanager.resource.cpu-vcores	The number of CPU cores that can be allocated for containers. Informatica recommends setting the value to the number of logical cores available in the node.
yarn.nodemanager.vmem-check-enabled	The virtual memory check is set to false by default. Retain the default value.

Configure the following properties in mapred-site.xml

mapreduce.map.memory.mb	Memory allocated to map task containers.
mapreduce.reduce.memory.mb	Memory allocated to reduce task containers.

Note: The values arrived at using the formula can serve as a starting point and can be tweaked based on simple performance tests.

The following example shows how to configure the MRv2 YARN and MapReduce settings:

```
Number of logical cores = 24
Number of disks = 7
Amount of physical memory available for containers = 64 GB

Mp = min (4 X 7, 24) = min (28, 24) = 24
Rp = min (0.65 X 24, 3 X 7) = min (15.6, 21) = 15.6 ≈ 16

yarn.nodemanager.resource.memory-mb = 65536
```

```
mapreduce.map.memory.mb
= yarn.nodemanager.resource.memory-mb / Mp
= 65536 / 24 = 2730

mapreduce.reduce.memory.mb
= yarn.nodemanager.resource.memory-mb / Rp
= 65536 / 16 = 4096
```

For more information about tuning the hardware and the Hadoop cluster, refer to the following Informatica How-To Library article:

Tuning the Hardware and Hadoop Clusters for Informatica Big Data Products

MapR Hadoop Distribution

In a MapR environment, consider the following performance guideline for MapR volumes:

MapR Volume is a logical unit that allows you to apply policies to a set of files, directories, and sub-volumes. When the Hive warehouse or the Hive scratch directory is configured on a different volume than the HDFS source or targets, there is a significant delay between subsequent Hive queries.

For more information, refer to the following link:

http://answers.mapr.com/questions/3560/failed-with-exception-cannot-rename-across-volumes

Data Integration Service

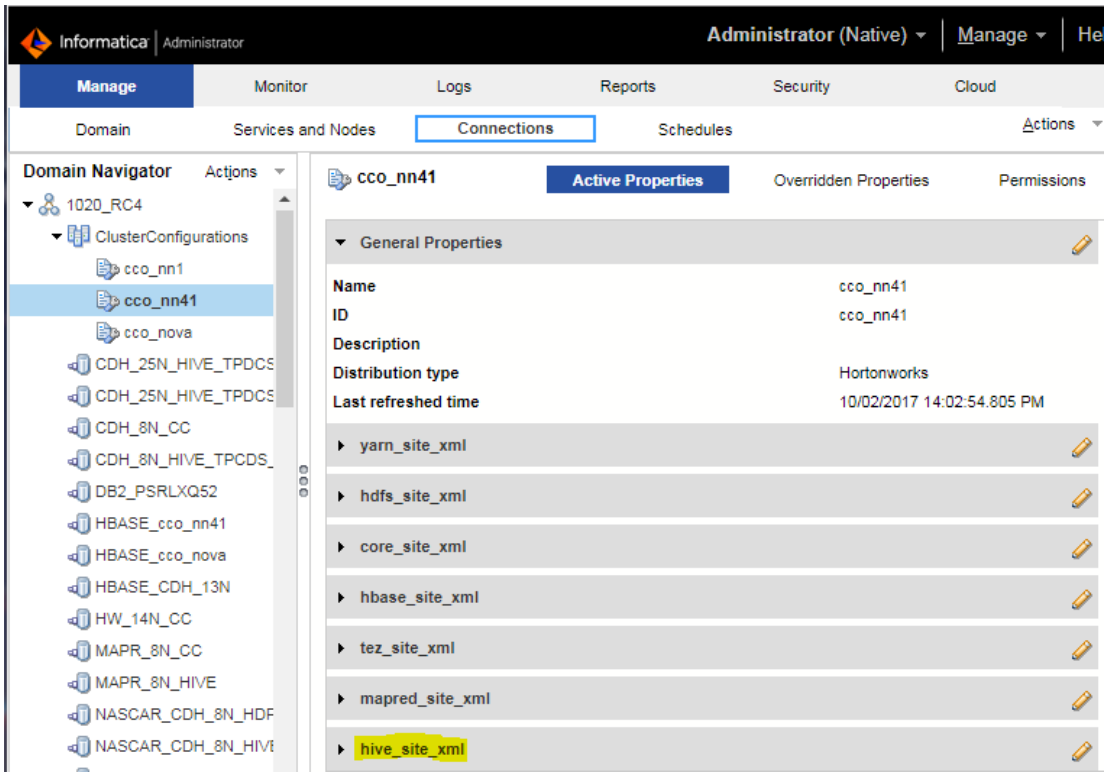
You can tune Hadoop configuration properties for the Data Integration Service in the following files:

- hive-site.xml
- hadoopEnv.properties

hive-site.xml

In the Administrator tool, you can tune the hive-site.xml file properties.

The following figure shows the properties of a cluster configuration in the **Connections** tab:



Configure the following properties in hive-site.xml:

mapred.compress.map.output

Determines whether the map phase's output is compressed or not. Set to false by default. Informatica recommends turning it on for better performance by setting this parameter to true.

mapred.map.output.compression.codec

Specifies the compression codec to be used for map output compression. Default is set to org.apache.hadoop.io.compress.DefaultCodec. Snappy codec is recommended for better performance. For more information, refer to the following codec:

org.apache.hadoop.io.compress.SnappyCodec.

mapred.map.tasks.speculative.execution

Specifies whether the map tasks can be speculatively executed. Default is set to true. With speculative map task execution, duplicate tasks are spawned for the tasks that are not making much progress. Original and speculative tasks are considered alike. The task that completes first is considered and the other is killed.

Informatica recommends keeping the default value set to true for better performance.

mapred.reduce.tasks.speculative.execution

Specifies whether the reduce tasks can be speculatively executed. Default is set to true. This is similar to map task speculative execution in functionality.

Informatica recommends setting mapred.reduce.tasks.speculative.execution to false to disable the property.

mapred.min.split.size and mapred.max.split.size

Use these two properties in conjunction with the dfs.block.size property. These parameters impact the number of input splits and hence the parallelism.

Informatica recommends using the following formula for each map task on a data block:

```
mapred.min.split.size < dfs.block.size < mapred.max.split.size
```

The input split size is calculated by the following formula:

```
max(minimumSize, min(maximumSize, blockSize))
```

hive.exec.compress.intermediate

Determines whether the results of intermediate map and reduce jobs in a Hive query are compressed or not. Default is set to false. This should not be confused with the mapred.compress.map.output property that deals with the compression of the output of map task.

Informatica recommends setting the hive.exec.compress.intermediate property to true to enable the property. hive.exec.compress.intermediate uses the same codec specified by mapred.output.compression.codec, and SnappyCodec is recommended for better performance.

Number of Reduce Tasks

You can use the following properties to determine the number of reduce tasks for a mapping:

mapred.reduce.tasks

Specifies the number of reduce tasks per job. Informatica provides a default value set to -1. This enables Hive to automatically determine the number of reducers for a job.

hive.exec.reducers.max

Determines the maximum number of reduce tasks that can be created for a job when the mapred.reduce.tasks property is set to '-1'.

hive.exec.reducers.bytes.per.reducer

Determines the number of reducers by way of specifying the size of data per reducer. Default value is 1GB. For example, for an input size of 10GB, 10 reducers are used.

For properties related to reduce tasks, default values work well for most mappings. However, for mappings that use a Data Processor transformation and a complex file writer, the values of these properties might require an increased number of reduce tasks.

hadoopEnv.properties

hadoopEnv.properties is located in the following location: $INFA_HOME/services/shared/hadoop/<distr>/infaconf/hadoopEnv.properties.

Configure the following property in the hadoopEnv.properties file:

mapred.[reduce].child.java.opts

Since many map and reduce tasks run in parallel, Informatica recommends setting the max heap size based on the system memory to reduce or avoid swapping tasks.

For a parallel garbage collector, Informatica recommends setting the -XX:ParallelGCThreads property to a smaller value such as 2 to reduce context switching overhead since many map and reduce tasks are executed by a task tracker.

For example, assume that a slave node has 32GB RAM, and it is configured to have 32 total slots including map slots and reduce slots. In this case, the JVM maximum heap size value '-Xmx' can be specified up to a maximum of 1GB.

However, this does not rule out swapping. Other operating system processes can compete for memory and the DTM process running on the Hadoop node consumes memory outside the JVM heap. The default value of '-Xmx512M' is good enough for most of the cases. However, Informatica recommends setting the correct value by experimenting with different values to ensure swapping does not impact performance.

Mapping Optimization

You can tune the following components for a mapping:

- Precision for ports
- User-Defined Function Calls
- DTM buffer block size and DTM buffer size
- Lookup transformation
- Join reordering
- HBase Read or Write
- Storage Handler for Hive and HBase
- HParser and integrated data transformation
- Staging and compression
- Parquet and Avro
- TDCH

Precision for Ports

Performance degrades when mappings contain large precision ports. Informatica recommends setting the precision judiciously for ports in sources, transformations, and targets.

For example, when a string port can handle data of a maximum of 200 characters, do not set the precision of the port to 1000.

See "Case Study: Port Precision" on page 13.

User-Defined Function Calls

Every user-defined function (UDF) call invokes a C++ component through the JNI call. As a rule of thumb, avoid UDF calls whenever possible.

For example, the following cases generate UDF calls:

- Informatica UDF calls are generated when the precision values of the propagating ports do not match.
- UDF calls are generated for data type conversions. Avoid unnecessary data type conversions. For example, from string to date and vice versa.

Lookup Transformation

In a Lookup transformation, Informatica runs a single MapReduce job to build lookup data, compress it, and store it in the Hadoop distributed cache.

Informatica does not support variable length data type handling. Therefore, memory is allocated based on the precision set to the corresponding port. For example, if the precision is set to 500 for a string port and the data has 50 characters, Informatica allocates memory to hold 500 characters both on disk and in memory.

The number of columns from the Lookup transformation, lookup source data volume, and the lookup condition determine the size of the lookup cache and index files. For example, for a 1.7 GB source, the lookup data cache size on disk is around 20 GB and the index file cache size is around 408 MB. This data is compressed to 1 GB before moving it to the Hadoop distributed cache.

Lookup provides a competitive edge over join when the lookup data volume is less than 1 GB or 2 GB. When lookup data volumes are greater than 2 GB, the cost of building the lookup data cache and index cache overshadows the benefit of lookup. In this case, joiner performs better than lookup.

Consider the following rules and guidelines to tune the lookup performance:

- Set the right precision.
- Propagate only the desired set of ports from the Lookup transformation.
- Verify that lookup source data volume is not several GBs.

See "Case Study: Lookup vs. Joiner Transformations" on page 17, which compares the performance of lookups with joiners.

DTM Buffer Block Size and DTM Buffer Size

With large data volumes and larger row sizes, you create mappings that define columns with large precision. To account for the larger column size, Informatica recommends setting custom properties to override the default DTM buffer block size and the DTM buffer size.

Note: These settings apply only to mappings that run on the Hive engine.

Use the Administrator tool to configure custom properties for the Data Integration Service Process.

For example, set the following custom properties in the **Data Integration Service Process** Properties:

```
ExecutionContextOptions.PDO.DTM.BufferBlockSize = 8276800
ExecutionContextOptions.PDO.DTM.BufferSize = 132428800
```

Joiner Transformation

When a mapping contains multiple Joiner transformations, Informatica recommends reordering the Joiner transformations so that the sources with smaller amounts of data are joined before joining sources with larger amounts of data.

See "Case Study: Join Reordering" on page 11.

HBase Read or Write

When the Data Integration Service needs to read from an HBase table, it creates one map task per region of the table. For better performance, Informatica recommends suitably splitting the table.

Writes to HBase tables are also heavily impacted by table splits. The table can be pre-split with pre-created regions for better performance. Splitting should be done in such a way that one specific region does not get much higher requests than the others. The data should be distributed across the regions and the split strategy should not just be based on the key range but also on the keys that are being written.

If you use PowerExchange® for HBase to write, you can disable auto flush for the writer for improved performance.

See "Case Study: HBase Read and Write" on page 20.

Hive Storage Handler for HBase

To read from or write to HBase tables, you can use Informatica PowerExchange® for HBase or the Hive Storage Handler for HBase from Apache Hive.

With the Hive Storage Handler, the Hive table is super imposed on the HBase table and can be used like a normal Hive table in the Big Data Management mappings.

Informatica recommends experimenting and comparing the performance of Informatica PowerExchange for HBase and Hive Storage Handler before arriving at a conclusion.

See "Case Study: Hive Storage Handler for HBase" on page 23.

HParser and Integrated Data Transformation

When comparing HParser with the integrated Data Transformation that comes with Big Data Management, HParser was found to perform better. Integrated Data Transformation was about ~2.57X slower in some internal tests. Informatica recommends trying out and comparing the performance of HParser and integrated Data Transformation before arriving at a conclusion.

If you use Data Transformation, Informatica recommends using a splittable input format if available and design the Data Processor transformation's streaming service to process a batch of records using the count property of the streamer.

For more information, see the following topics in the *Informatica PowerExchange for HDFS User Guide*:

- HDFS Data Extraction
- Complex Files Partitioning

See "Case Study: HParser vs. Integrated Data Transformation" on page 23

Staging and Compression

Some Big Data Management mappings need staging. Informatica stages the mapping data in sequence file format. To save HDFS space for transient data and for better performance, Informatica recommends using the Snappy compression codec. This codec can be configured as a property in the Hive connection.

See "Case Study: Staging and Compression" on page 24.

Parquet and Avro

You use Big Data Management's Data Transformation to read or write data in the Parquet or Avro file format. Informatica recommends using Data Transformation only if Data Transformation is truly required. For better performance, use serde-backed Hive tables to read or write data in the Parquet or Avro formats.

TDCH

The fastest way to load data from Teradata to HDFS is to use the TDCH option. To enable the TDCH option, set the following property in the Data Integration Service:

```
EnableTdch=True
```

For configuration information, see the topic "Loading to Teradata" in the chapter "External Loading" in the *PowerCenter® Advanced Workflow Guide*.

Also, refer to the following How-To Library article:

Implementing Teradata Utilities in a PowerCenter Mapping

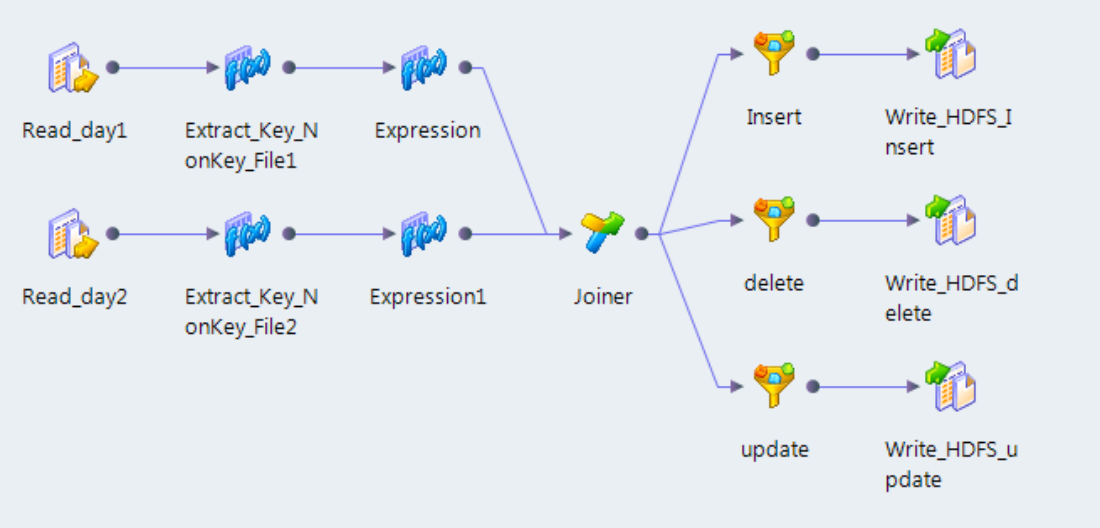
Case Studies

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

Case Study: Compression and DTM Buffer Tuning

This example demonstrates the performance benefit of enabling compression and increasing DTM buffer sizes.

The following image shows a mapping for the sLog Parsing use case (CDC - Change Data Capture):



The customer has two log files and wants to identify the matching records, new records and deleted records so that the records can be moved to the appropriate files. The log files being parsed have close to 600 tokens that are delimited by a special character (Ç). MD5 hashing is calculated on the substring formed by the first five tokens from both logs. Each line of the log file is considered as a non-key value and the MD5 hash value acts as the key on which a full outer join is performed before filtering and moving the records to the right files.

The log files have the following characteristics:

Data Volume : Each log file is approximately 100 GB.

Precision of the Data Ports : 4000

Optimization 1: Compression

Since the column width has large precision of 4000, the key-value (K, V) pairs emitted by the mapper will be wide. Since the MD5 hashing is calculated on a subset of the tokens obtained via substring functions, the CPU load is minimal. However, because the column width is large, there is a lot of network I/O. This network I/O can be reduced by enabling compression at different levels.

The following compression parameters were applied for optimization:

- Map compression (mapred.compress.map.output=true)
- Hive intermediate compression (hive.exec.compress.intermediate=true)
- Snappy compression coded selected for compression (mapred.map.output.compression.codec=org.apache.hadoop.io.compress.SnappyCodec)

Optimization-2: DTM Buffer tuning for large precision data types

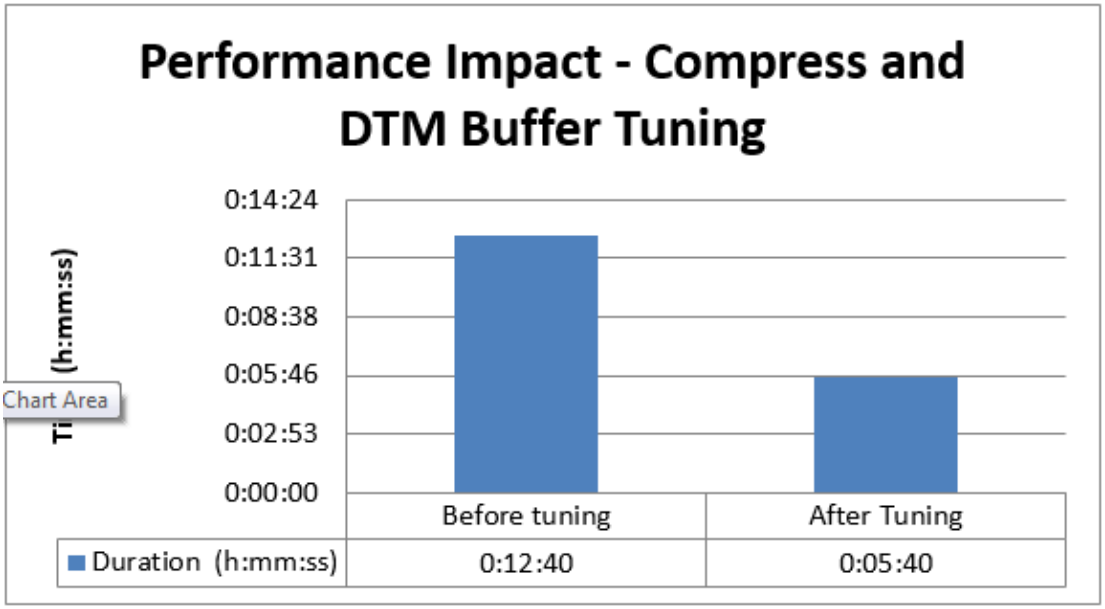
Since the column width is large, the default DTM buffer block size and the DTM buffer size were increased as follows:

1. a) ExecutionContextOptions.PDO.DTM.BufferBlockSize=8276800

ExecutionContextOptions.PDO.DTM.BufferSize=132428800

Result

Applying these optimizations resulted in a ~2.2X improvement in performance.



Case Study: Join Reordering

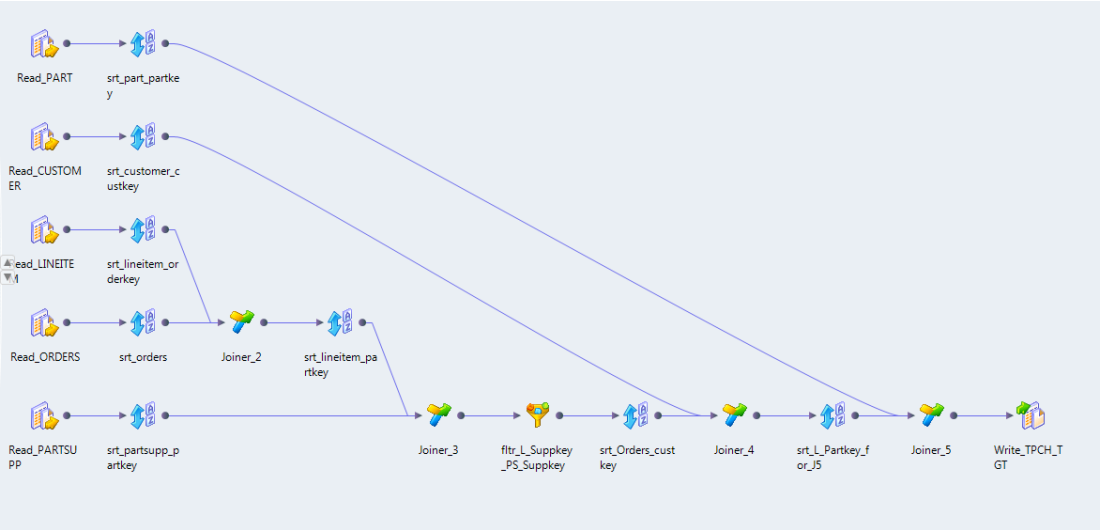
This example demonstrates the performance benefit of join re-ordering.

Joining Five Tables Before Tuning for Performance

The mapping uses the following files:

Number	File Name	Size
1	PART	23.1 GB
2	CUSTOMER	23.08 GB
3	LINEITEM	757.86 GB
4	ORDERS	168.5 GB
5	PARTSUPP	115.2 GB

The following image shows the mapping that is not optimized:

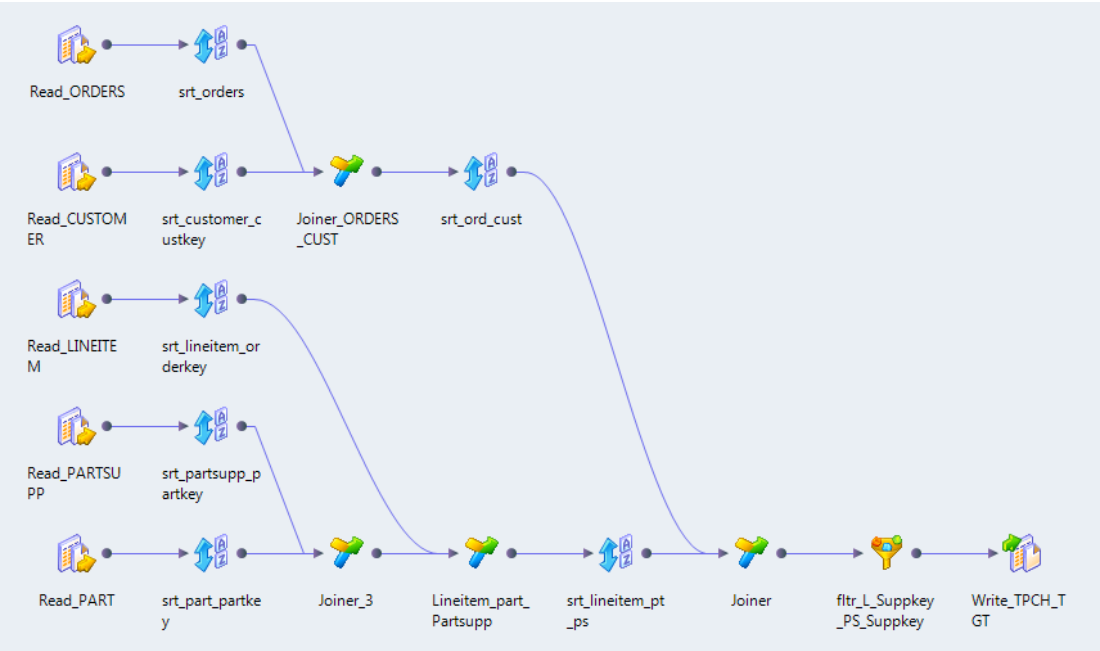


The largest source file that the mapping uses is LINEITEM of size 757.86 GB, which is many times larger than the other files that the mapping uses. In the mapping, LINEITEM is successively joined three times with other files.

Joining Five Tables After Tuning for Performance

The mapping is tuned by reordering the Joiner transformations in such a way that the smaller tables are joined before they get joined with LINEITEM.

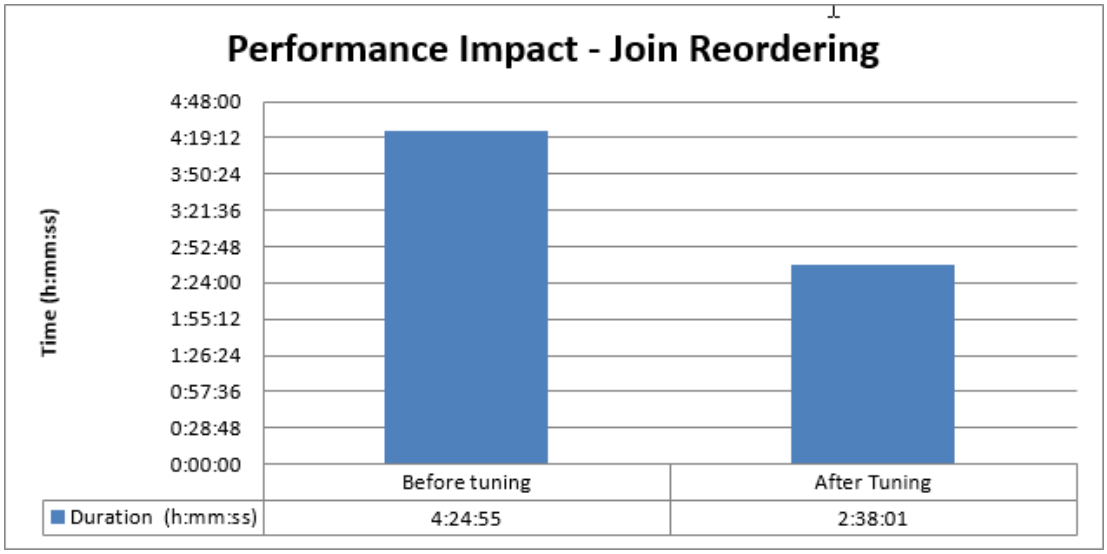
The following image shows the mapping that is optimized for join reordering:



The mapping joins the smaller tables ORDERS (168.5 GB) and CUSTOMER (23.08 GB) as well as PART (23.1 GB) and PARTSUPP (115.2 GB) before it joins the LINEITEM table.

Result

The mapping optimized for join reordering completed 40% faster than the mapping that was not optimized.



Case Study: Port Precision

This example demonstrates the performance benefits of configuring the correct precision setting for ports.

Expression Evaluation Mapping Before Optimization

The mapping reads data from an HDFS flat file, applies an expression to the data, and writes the results data to an HDFS flat file.

The mapping has a higher precision of 2000 for string ports but the actual source data being read from the file does not require larger port sizes.

The following image shows the columns with precision 2000 in the flat file source used in the mapping:

Columns

	Name	Native Type	Precision	Scale	Format	Visibility	Description
1	l_orderkey	double	15	0	<default>	Read and Write	
2	l_partkey	double	15	0	<default>	Read and Write	
3	l_suppkey	double	15	0	<default>	Read and Write	
4	l_linenumber	double	15	0	<default>	Read and Write	
5	l_quantity	double	15	0	<default>	Read and Write	
6	l_extendedprice	double	15	2	<default>	Read and Write	
7	l_discount	double	15	2	<default>	Read and Write	
8	l_tax	double	15	2	<default>	Read and Write	
9	l_returnflag	string	2000	0		Read and Write	
10	l_linestatus	string	2000	0		Read and Write	
11	l_shipdate	string	2000	0		Read and Write	
12	l_commitdate	string	2000	0		Read and Write	
13	l_receiptdate	string	2000	0		Read and Write	
14	l_shipinstruct	string	2000	0		Read and Write	
15	l_shipmode	string	2000	0		Read and Write	
16	l_comment	string	2000	0		Read and Write	

Expression Evaluation Mapping After Optimization

To tune the mapping, the port sizes for the String columns were reduced to a lower value.

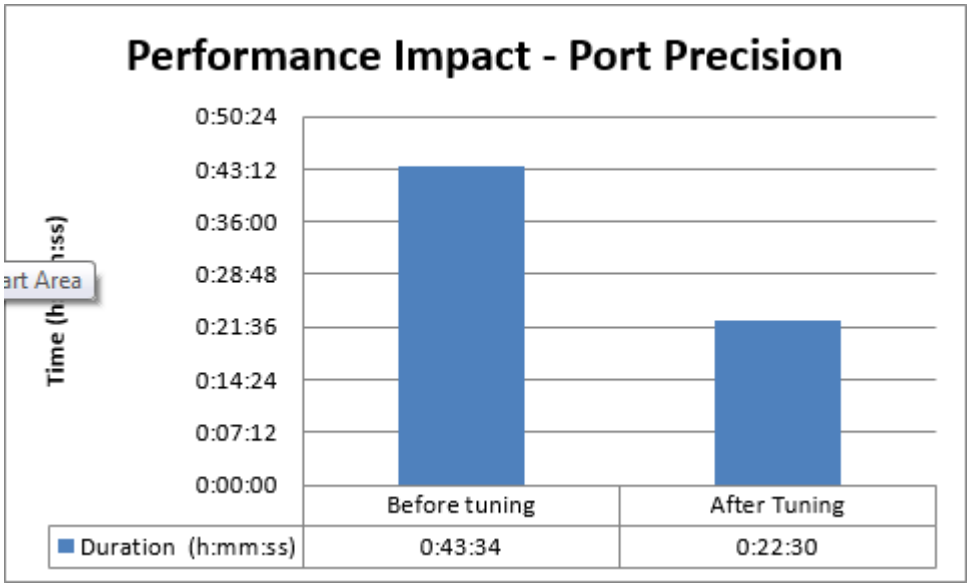
The following image shows the columns with reduced precision in the flat file source used in the mapping:

Columns

	Name	Native Type	Precision	Scale	Format	Visibility	Description
1	l_orderkey	double	15	0	<default>	Read and Write	
2	l_partkey	double	15	0	<default>	Read and Write	
3	l_suppkey	double	15	0	<default>	Read and Write	
4	l_linenumber	double	15	0	<default>	Read and Write	
5	l_quantity	double	15	0	<default>	Read and Write	
6	l_extendedprice	double	15	2	<default>	Read and Write	
7	l_discount	double	15	2	<default>	Read and Write	
8	l_tax	double	15	2	<default>	Read and Write	
9	l_returnflag	string	100	0		Read and Write	
10	l_linestatus	string	100	0		Read and Write	
11	l_shipdate	string	100	0		Read and Write	
12	l_commitdate	string	100	0		Read and Write	
13	l_receiptdate	string	100	0		Read and Write	
14	l_shipinstruct	string	100	0		Read and Write	
15	l_shipmode	string	100	0		Read and Write	
16	l_comment	string	200	0		Read and Write	

Result

The mapping execution time reduced by 50%.

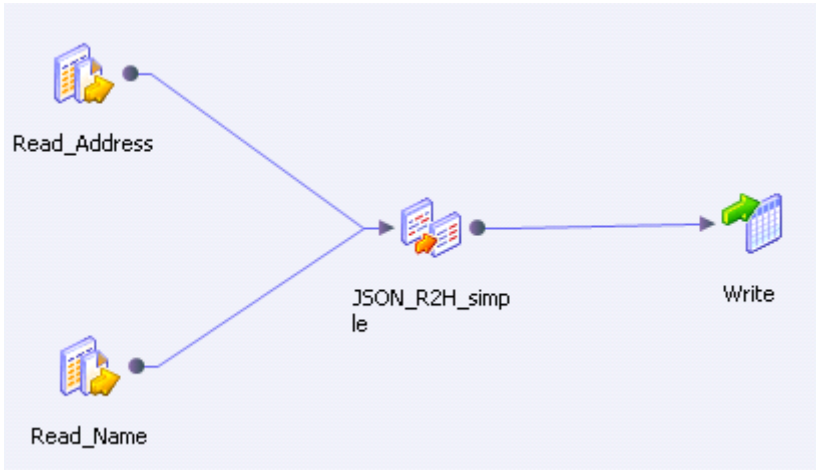


Case Study: Increasing the Number of Reduce Tasks

This example demonstrates the benefit of modifying the value of 'mapred.reduce.tasks'.

JSON R2H Mapping

The following image shows the mapping:



The mapping reads two files, Name and Address, that contain normalized data and uses the Data Processor transformation to convert the relational input from the two source files to a JSON denormalized format. It then uses a complex file writer to write the result to an HDFS flat file target.

The Name file is 23 GB and the Address file is 40 GB.

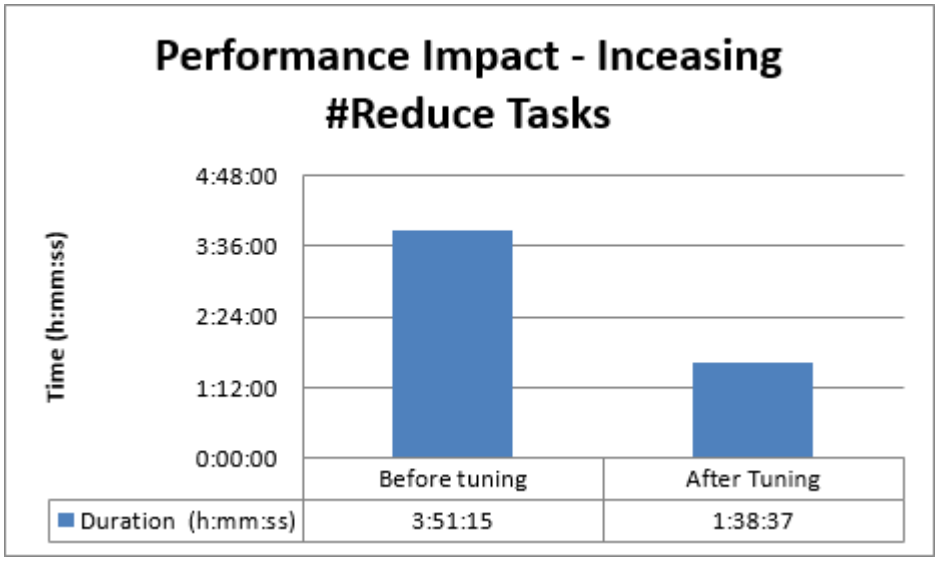
The Data Processor conversion of relational data to a hierarchical JSON object happens during the reduce phase. As the combined input size is about 63 GB, approximately 63 reduce tasks are created with the default value of '-1' for 'mapred.reduce.tasks'.

The Data Processor transformation is CPU intensive. Therefore, increasing the number of reduce tasks can improve performance.

To improve performance, the parameter 'mapred.reduce.task' was modified from '-1' to '210'.

Result

Increasing the number of reduce tasks resulted in a 56% improvement in performance.

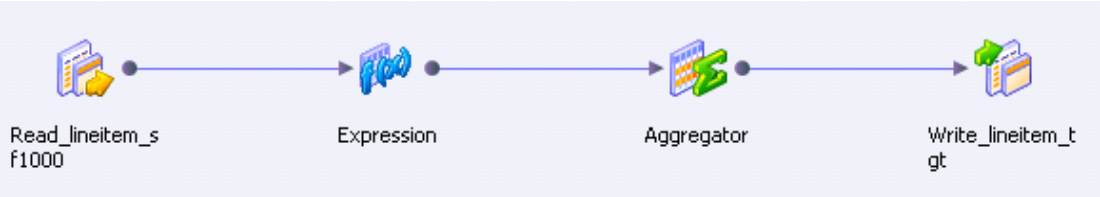


Case Study: Storage Handler Improvements

Effective in version 9.6.0, Informatica storage handler includes optimizations that can result in better performance.

When running the mapping with HDFS as source, the storage handler can be used against Hive when a Hive table is used as a source.

The following figure shows the mapping with an Aggregator transformation:

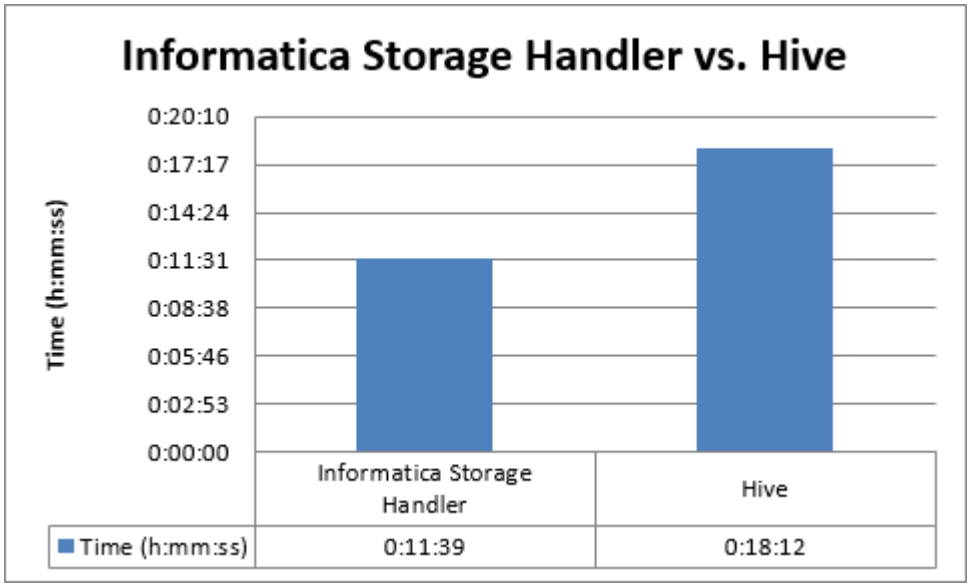


The mapping reads a 750 GB file, applies an expression to each row of data, and stores the aggregated result in a file.

To test performance, the mapping is run with the Informatica storage handler that uses HDFS as source and target. The mapping is also run with the Hive storage handler that uses Hive as source and target.

Result

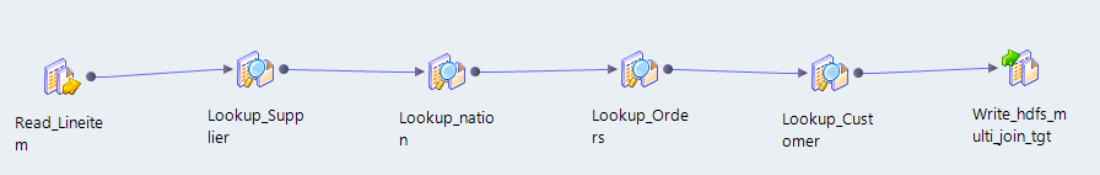
The performance for Informatica's storage handler was approximately 56 percent faster than Hive's storage handler.



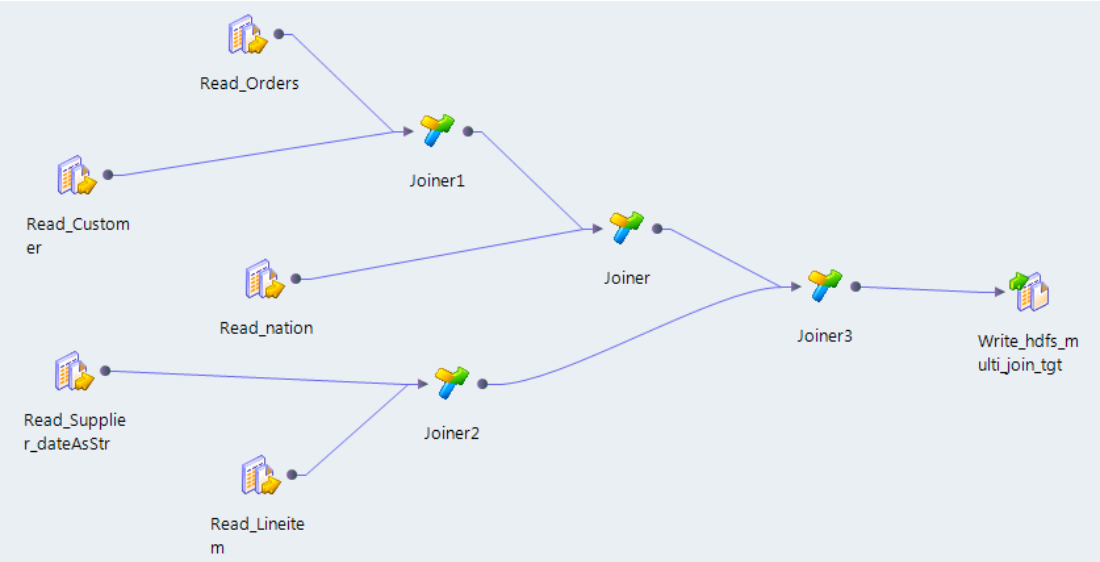
Case Study: Lookup vs. Joiner Transformations

To illustrate how the lookup source data volume, number of columns projecting from the Lookup transformation, and the port precisions impact the lookup performance, the following two mappings were run by varying the lookup data volume and columns.

The following image shows the mapping that contains Lookup transformations:



The following image shows the mapping that contains detailed outer Joiner transformations:

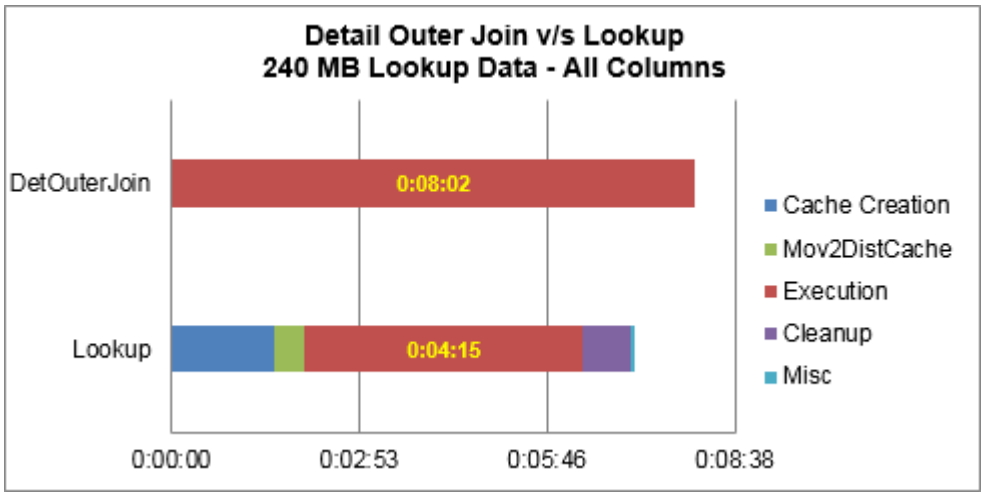


Result

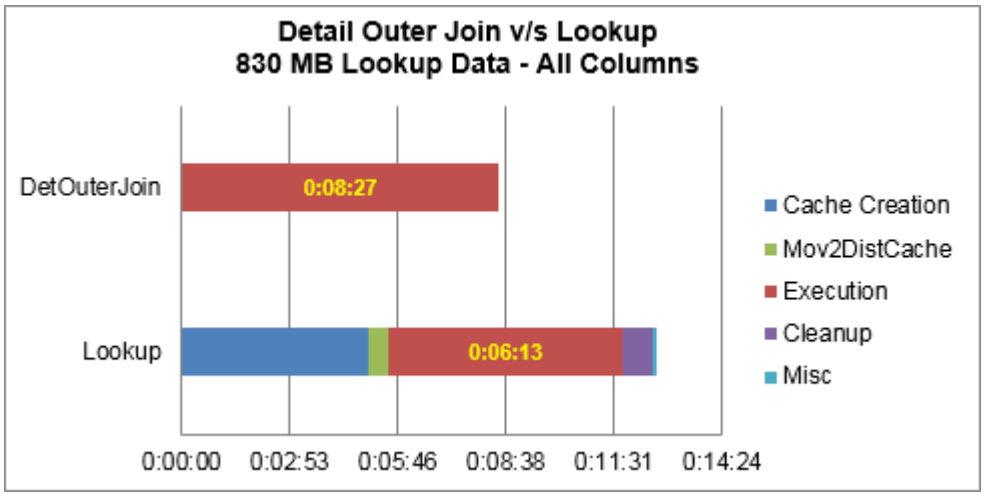
When the lookup data size grows beyond 2 GB, joiners outperform lookups.

The following two graphs show that as the lookup data size grows over 800 MB, lookups start to lose their competitive edge over the joiners. But more realistic lookup use cases will project only a subset of the columns from the lookup transformations.

Graph for 240 MB lookup data before optimization:

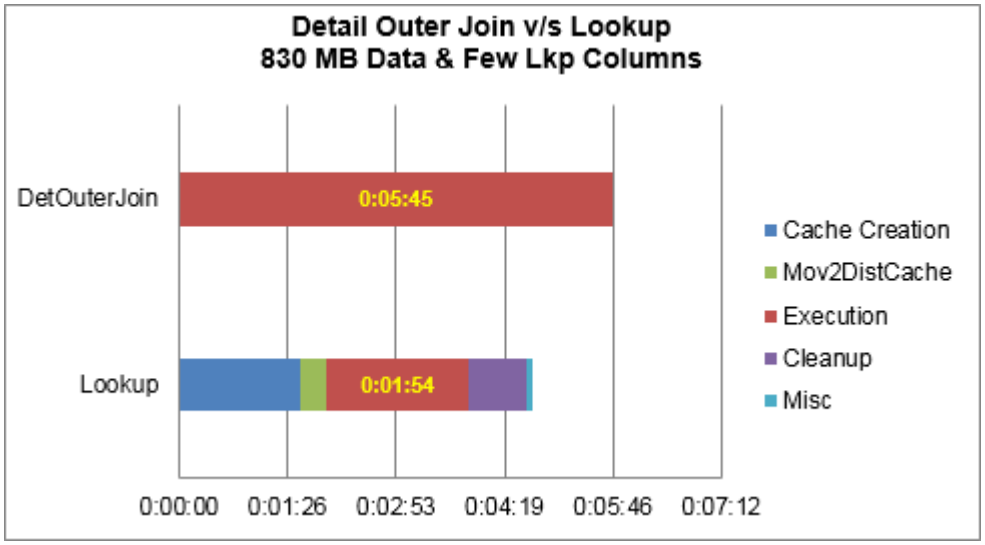


Graph for 830 MB lookup data before optimization:

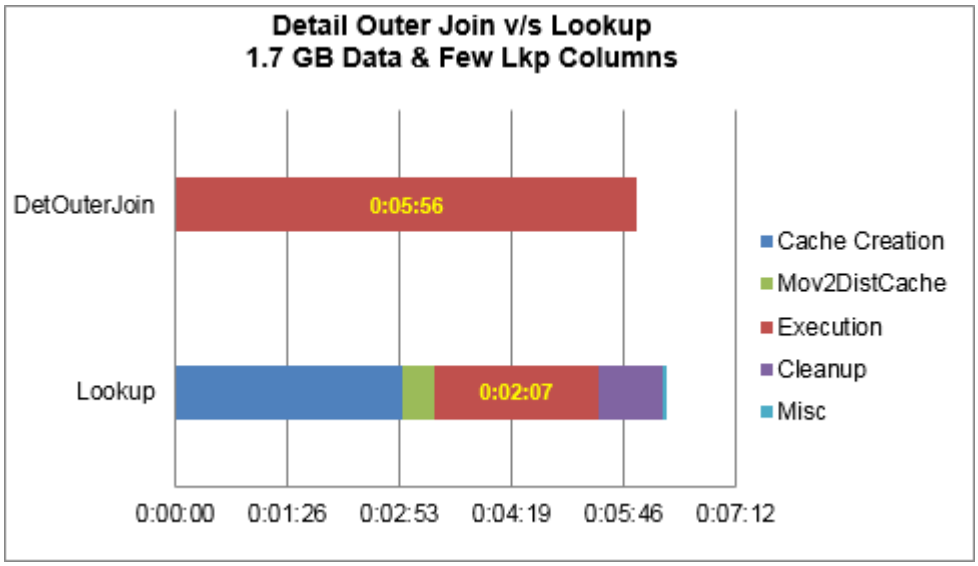


The following two graphs show that configuring the right precision for ports and propagating only the desired ports positively impact the performance of lookups.

Graph for 830 MB data after optimization:



Graph for 1.7 GB data after optimization:



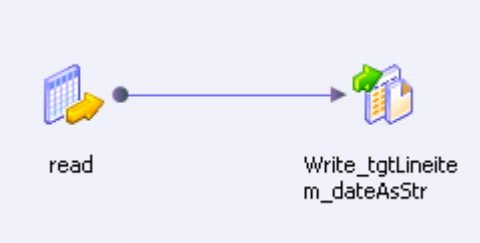
Case Study: HBase Read and Write

This example demonstrates the HBase read and write optimization.

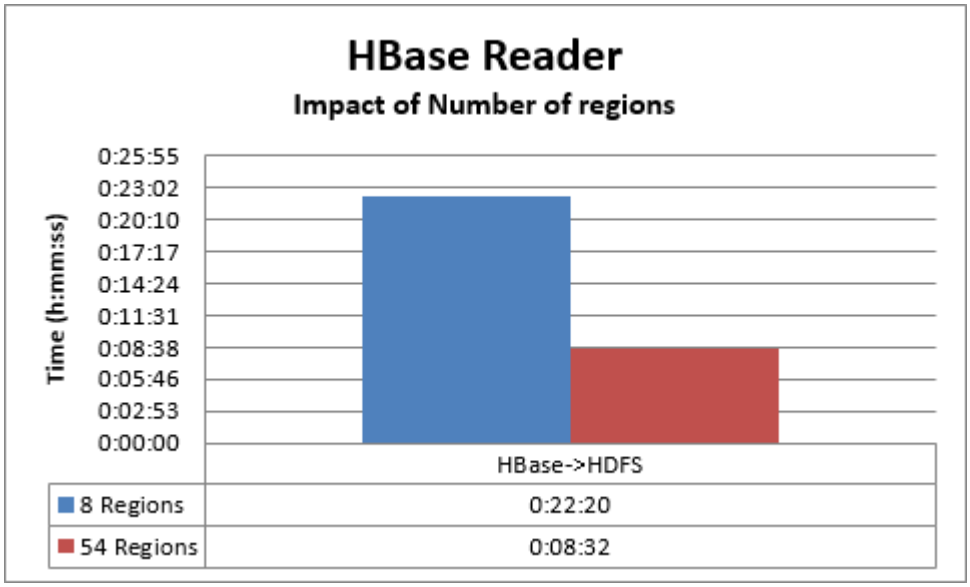
HBase Reader Mapping

The mapping reads data from an HBase table and writes the data to an HDFS file that contains approximately 7.5GB of data from 60 million relational records. The HBase table originally had 8 regions. When the table is split into 54 regions, the performance improves by 62%.

The following image shows the mapping:



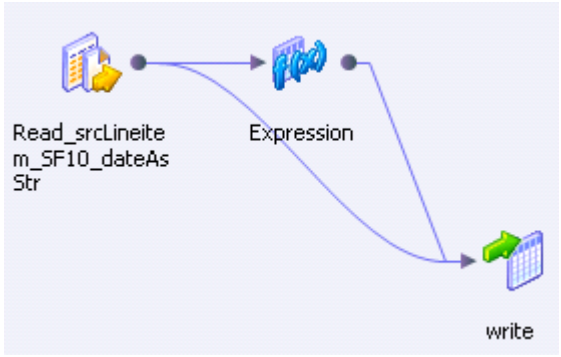
Result



HBase Writer Mapping

The mapping reads data from an HDFS File that contains approximately 7.5GB of data from 60 million relational records. The mapping uses an Expression transformation to generate the HBase Rowkey and writes the data to an HBase table.

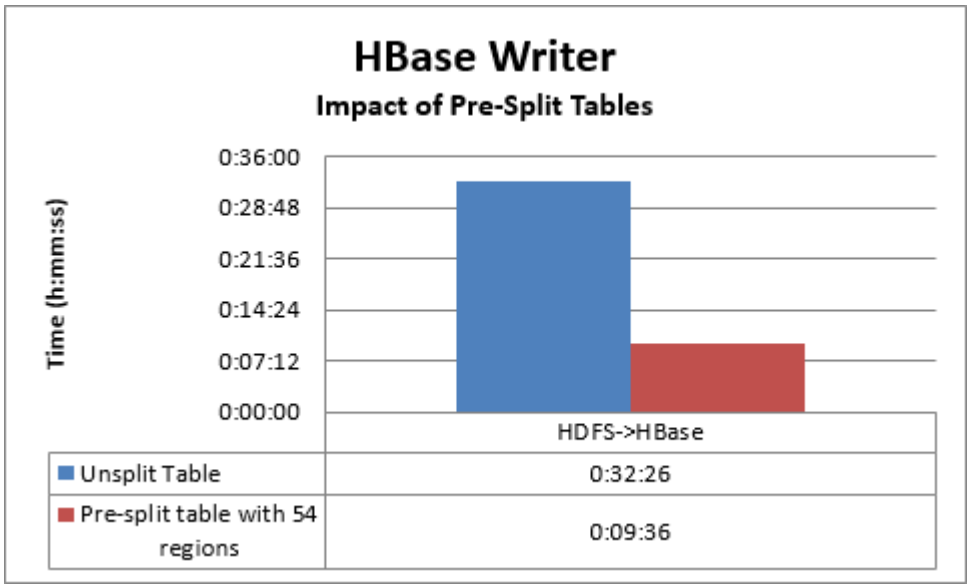
The following image shows the mapping:



Pre-splitting and Result

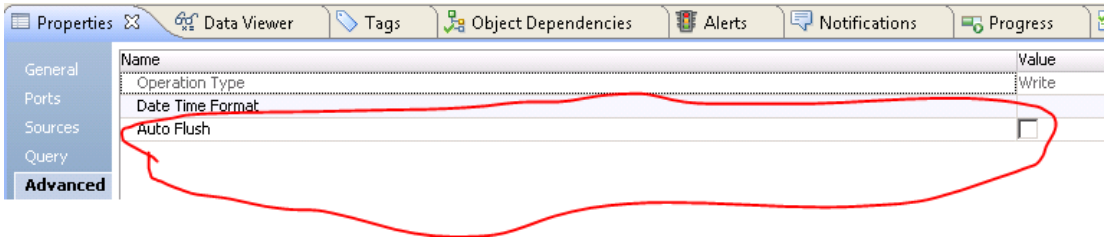
The HBase table was not originally split. When the HBase table is pre-split into 54 regions, the performance improves by 60%.

The following image shows the result for a pre-split table:

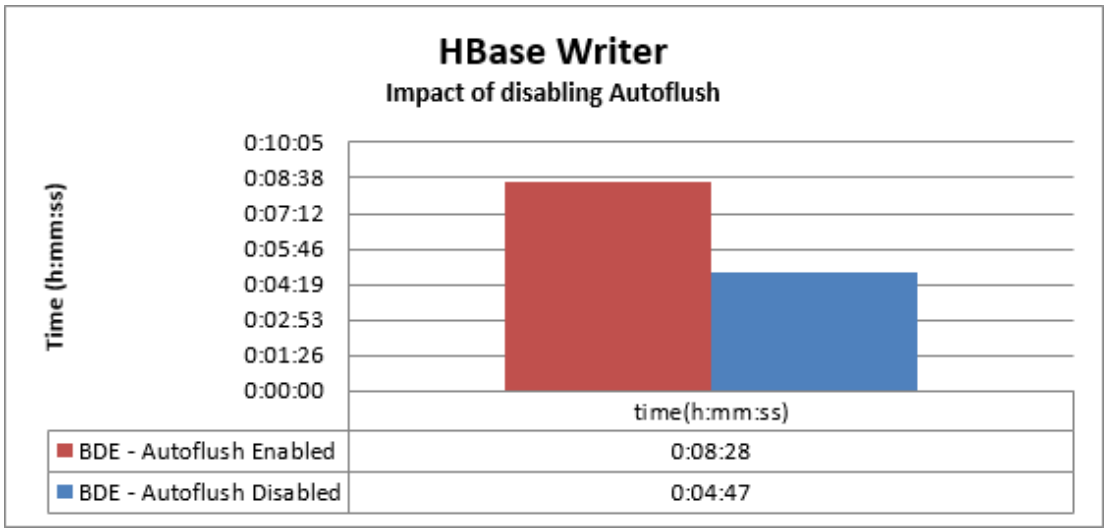


Disabling Auto Flush

When you use PowerExchange for HBase to write data to HBase and disable Auto Flush, performance improves by about 43%.



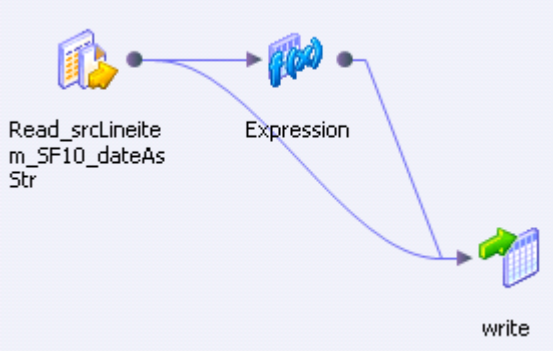
The following image shows the result after disabling auto flush:



Case Study: Hive Storage Handler for HBase

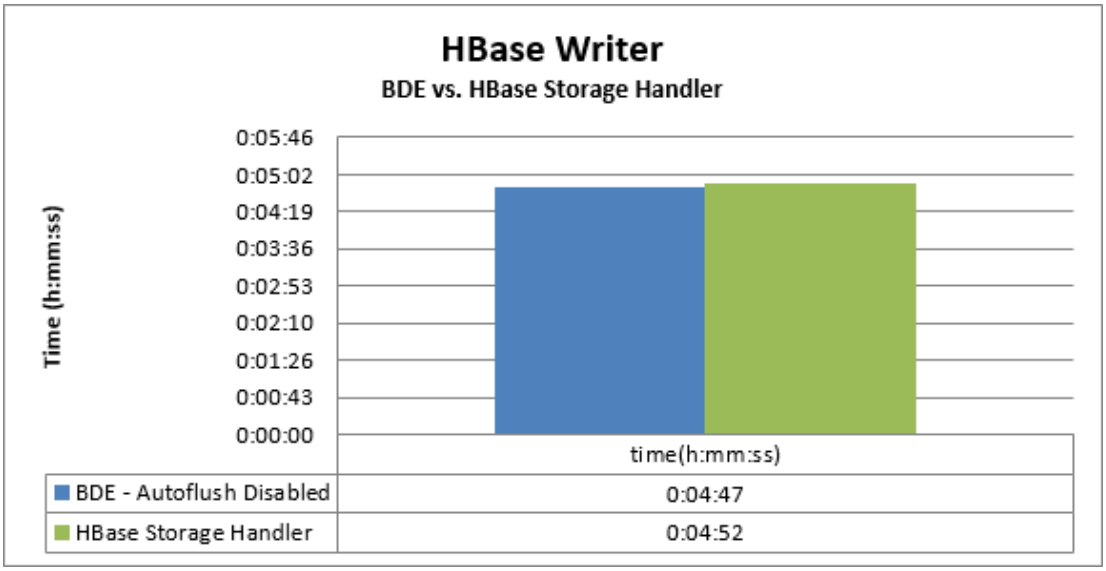
The same HBase Writer mapping used in the HBase write case Study was compared with the HBase storage handler.

The mapping reads data from an HDFS File that contains approximately 7.5GB of data from 60 million relational records. The mapping uses an Expression transformation to generate the HBase Rowkey and writes the data to an HBase table.



Result

PowerExchange for HBase and HBase Storage handler was observed to perform on par.

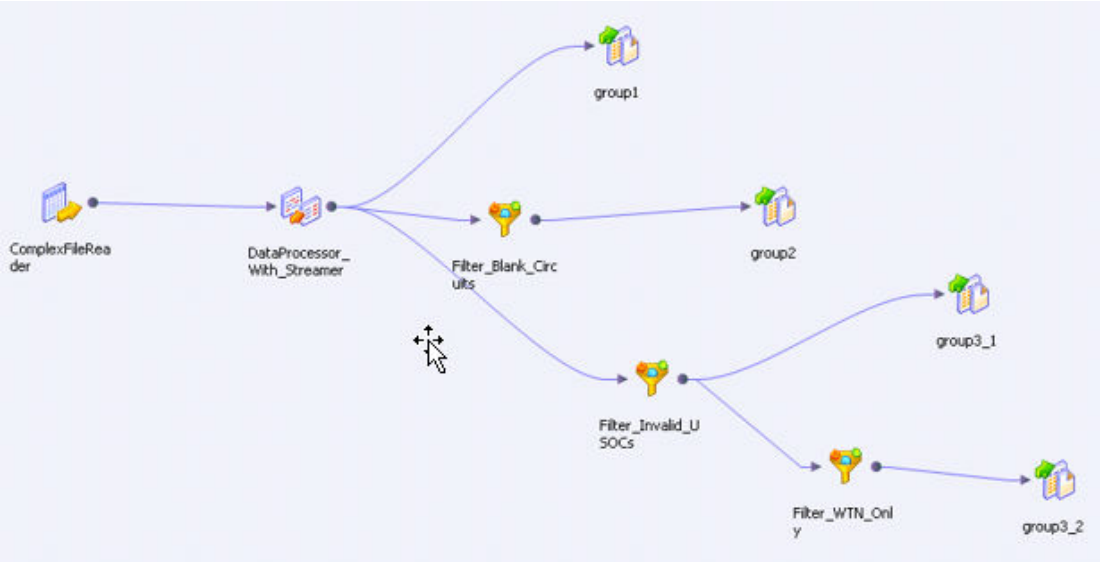


Case Study: HParser vs. Integrated Data Transformation

The mapping reads a file using complex file reader, parses and splits the file into three groups, and writes to different targets based on filter conditions.

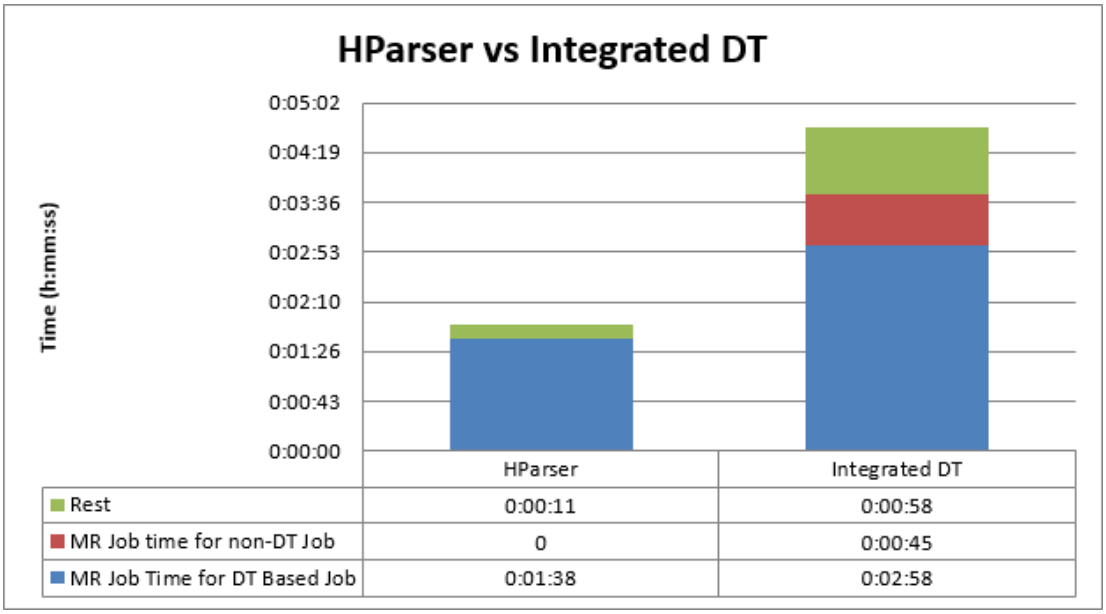
Nine map tasks were involved in reading approximately 500MB of file with 64 MB block size.

The following image shows the mapping:



Result

HParser was observed to perform better. Big Data Management was approximately 2.57X slower in this specific case.

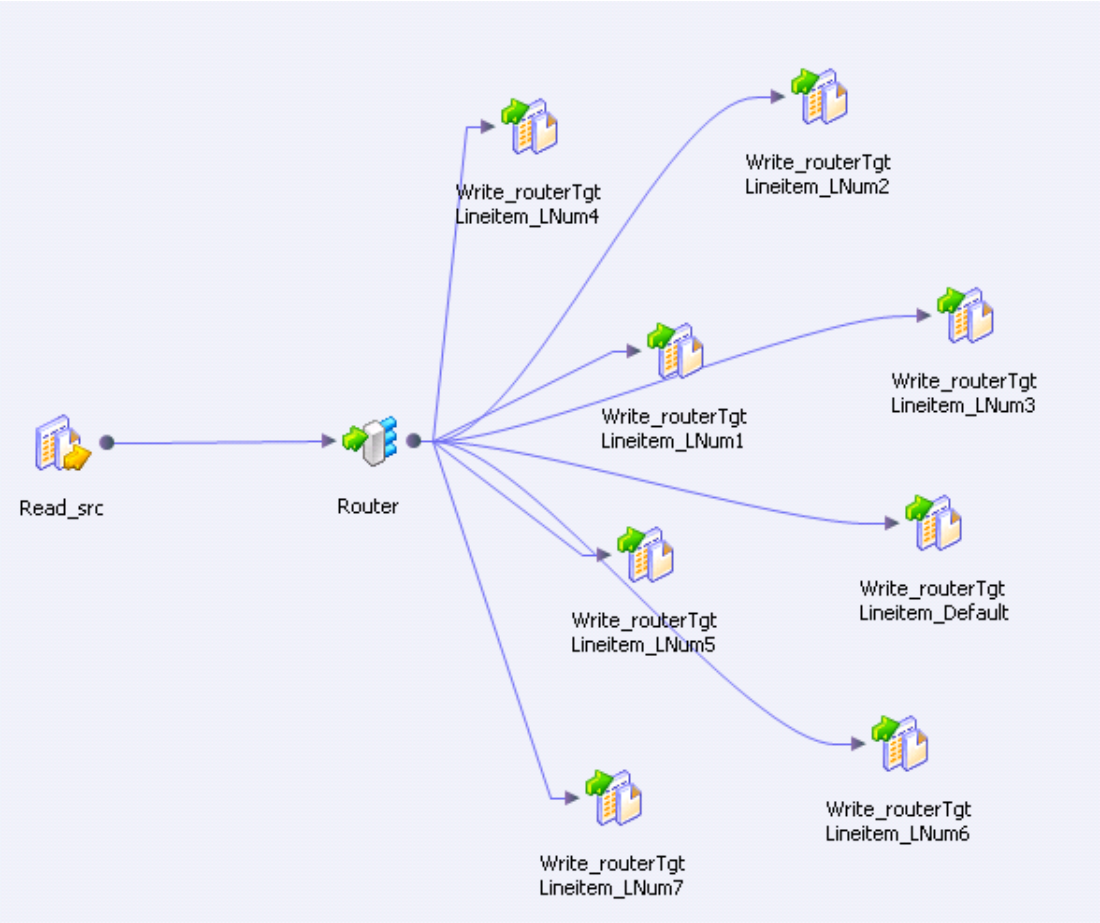


Case Study: Staging and Compression

The mapping reads about approximately 376GB from HDFS and writes it to 8 different HDFS targets based on the value of a column. This involves staging the data that is read from the HDFS source.

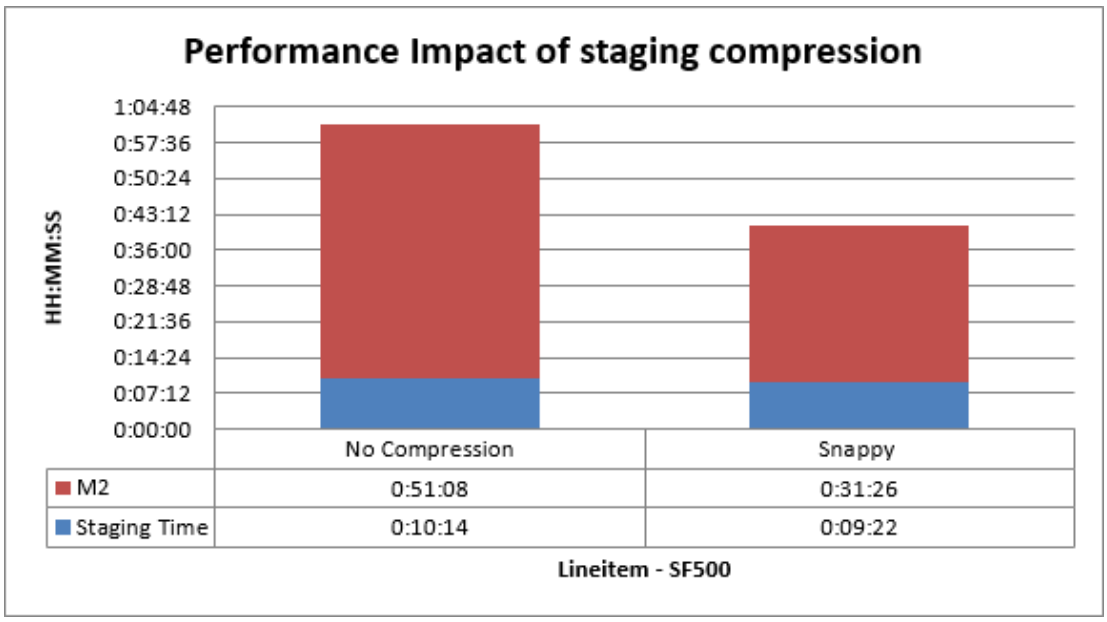
Enabling compression with Snappy codec improved the performance by 34%.

The following image shows a mapping that contains Router transformation:



Result

The following image shows the performance numbers for staging and compression:



Documentation Reference

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

Article	Description
Informatica big data products	
Tuning the Hardware and Hadoop Clusters for Informatica Big Data Products	Provides tuning recommendations for the hardware and the Hadoop cluster for better performance of Informatica big data products.
Big Data Management	
Performance Tuning and Sizing Guidelines for Big Data Management 10.2	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	Provides tuning recommendations to run mappings on the Hive engine, best practices to design efficient mappings, and case studies.
Strategies for Incremental Updates on Hive	Describes alternative solutions to the Update Strategy transformation for updating Hive tables to support incremental loads.
Intelligent Data Lake	
Performance Tuning and Sizing Guidelines for Intelligent Data Lake 10.2	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.
Intelligent Streaming	
Performance Tuning and Sizing Guidelines for Informatica Intelligent Streaming 10.2	Provides sizing recommendations and tuning guidelines for Informatica Intelligent Streaming.

Authors

Anand Sridharan

Vishal Kamath

Informatica Performance Engineering Team

Indra Sivakumar