cloudera
Ask Bigger Questions

Cloudera Impala
Frequently Asked Questions

Important Notice

Cloudera, Inc.
1001 Page Mill Road Bldg 2
Palo Alto, CA 94304
info@cloudera.com
US: 1-888-789-1488
Intl: 1-650-362-0488
www.cloudera.com

Release Information

Version: 1.4.x
Date: December 4, 2014

Table of Contents

Cloudera Impala Frequently Asked Questions

Trying Impala

How do I try Cloudera Impala out?

To look at the core features and functionality on Impala, the easiest way to try out Impala is to download the Cloudera QuickStart VM and start the Impala service through Cloudera Manager, then use `impala-shell` in a terminal window or the Impala Query UI in the Hue web interface.

To do performance testing and try out the management features for Impala on a cluster, you need to move beyond the QuickStart VM with its virtualized single-node environment. Ideally, download the Cloudera Manager software to set up the cluster, then install the Impala software through Cloudera Manager.

Does Cloudera offer a VM for demonstrating Impala?

Cloudera offers a demonstration VM called the QuickStart VM, available in VMWare, VirtualBox, and KVM formats. For more information, see the Cloudera QuickStart VM. After booting the QuickStart VM, many services are turned off by default; in the Cloudera Manager UI that appears automatically, turn on Impala and any other components that you want to try out.

Where can I find Impala documentation?

Starting with Impala 1.3.0, Impala documentation is integrated with the CDH 5 documentation, in addition to the standalone Impala documentation for use with CDH 4. For CDH 5, the core Impala developer and administrator information remains in the associated Impala documentation portion. Information about Impala release notes, installation, configuration, startup, and security is embedded in the corresponding CDH 5 guides:

- New features
- Known and fixed issues
- Incompatible changes
- Installing Impala
- Upgrading Impala
- Configuring Impala
- Starting Impala
- Security for Impala
- CDH Version and Packaging Information

Information about the latest CDH 4-compatible Impala release remains at the Impala Documentation page.

Where can I get more information about Impala?

More product information is available here:

- O'Reilly e-book: Cloudera Impala: Bringing the SQL and Hadoop Worlds Together
- Blog: Cloudera Impala: Real-Time Queries in Apache Hadoop, For Real
- Webinar: Introduction to Impala
- Product website page: Cloudera Enterprise RTQ

To see the latest release announcements for Impala, see the Cloudera Announcements forum.

How can I ask questions and provide feedback about Impala?

- Join the Impala discussion forum and the Impala mailing list to ask questions and provide feedback.

- Use the Impala Jira project to log bug reports and requests for features.

Where can I get sample data to try?

You can get scripts that produce data files and set up an environment for TPC-DS style benchmark tests from this Github repository. In addition to being useful for experimenting with performance, the tables are suited to experimenting with many aspects of SQL on Impala: they contain a good mixture of data types, data distributions, partitioning, and relational data suitable for join queries.

Impala System Requirements

What are the software and hardware requirements for running Impala?

For information on Impala requirements, see Cloudera Impala Requirements. Note that there is often a minimum required level of Cloudera Manager for any given Impala version.

How much memory is required?

Although Impala is not an in-memory database, when dealing with large tables and large result sets, you should expect to dedicate a substantial portion of physical memory for the `impalad` daemon. Recommended physical memory for an Impala node is 128 GB or higher. The amount of memory required for an Impala operation depends on several factors:

- The file format of the table. Different file formats represent the same data in more or fewer data files. The compression and encoding for each file format might require a different amount of temporary memory to decompress the data for analysis.
- Whether the operation is a `SELECT` or an `INSERT`. For example, Parquet tables require relatively little memory to query, because Impala reads and decompresses data in 8 MB chunks. Inserting into a Parquet table is a more memory-intensive operation because the data for each data file (with a maximum size of 1 GB) is stored in memory until encoded, compressed, and written to disk.
- Whether the table is partitioned or not, and whether a query against a partitioned table can take advantage of partition pruning.
- Whether the final result set is sorted by the `ORDER BY` clause. Each Impala node scans and filters a portion of the total data, and applies the `LIMIT` to its own portion of the result set. In Impala 1.4.0 and higher, if the sort operation requires more memory than is available on any particular host, Impala uses a temporary disk work area to perform the sort. The intermediate result sets are all sent back to the coordinator node, which does the final sorting and then applies the `LIMIT` clause to the final result set.

  For example, if you execute the query:

  ```
  select * from giant_table order by some_column limit 1000;
  ```

  and your cluster has 50 nodes, then each of those 50 nodes will transmit a maximum of 1000 rows back to the coordinator node. The coordinator node needs enough memory to sort (`LIMIT` * *cluster_size*) rows, although in the end the final result set is at most `LIMIT` rows, 1000 in this case.

  Likewise, if you execute the query:

  ```
  select * from giant_table where test_val > 100 order by some_column;
  ```

  then each node filters out a set of rows matching the `WHERE` conditions, sorts the results (with no size limit), and sends the sorted intermediate rows back to the coordinator node. The coordinator node might need substantial memory to sort the final result set, and so might use a temporary disk work area for that final phase of the query.

- The size of the result set. When intermediate results are being passed around between nodes, the amount of data depends on the number of columns returned by the query. For example, it is more memory-efficient to query only the columns that are actually needed in the result set rather than always issuing `SELECT *`.
- The mechanism by which work is divided for a join query.

Impala currently does not "spill to disk" if intermediate results being processed on a node exceed the memory reserved for Impala on that node. If this is an issue for your use case (for example, joins between very large tables), more memory will be beneficial.

See Hardware Requirements for more details and recommendations about Impala hardware prerequisites.

What processor type and speed does Cloudera recommend?

Impala makes use of SSE4.2 instructions. This corresponds to Nehalem+ for Intel chips and Bulldozer+ for AMD chips. Impala runs perfectly fine on older machines with the SSE3 instruction set, but will not achieve the best performance.

What EC2 instances are recommended for Impala?

For large storage capacity and large I/O bandwidth, consider the `hs1.8xlarge` and `cc2.8xlarge` instance types. Impala I/O patterns typically do not benefit enough from SSD storage to make up for the lower overall size. For performance and security considerations for deploying CDH and its components on AWS, see Cloudera Enterprise Reference Architecture for AWS Deployments.

Supported and Unsupported Functionality In Impala

What are the main features of Impala?

- A large subset of SQL and HiveQL commands, including SELECT and INSERT, with joins. For more information, see Impala SQL Language Reference.
- Distributed, high-performance queries. See Tuning Impala for Performance for information about Impala performance optimizations and tuning techniques for queries.
- Using Cloudera Manager, you can deploy and manage your Impala services. Cloudera Manager is the best way to get started with Impala on your cluster. Each release of Impala has a required corresponding level of Cloudera Manager; See Installing Impala with Cloudera Manager for the prerequisite for the current release. For more information about using Cloudera Manager with Impala, see the topic on Installing Impala with Cloudera Manager in the Cloudera Manager Installation Guide.
- Using Hue for queries.
- Appending and inserting data into tables through the INSERT statement. See How Impala Works with Hadoop File Formats for the details about which operations are supported for which file formats.
- ODBC: Impala is certified to run against MicroStrategy and Tableau, with restrictions. For more information, see Configuring Impala to Work with ODBC.
- Querying data stored in HDFS and HBase in a single query. See Using Impala to Query HBase Tables for details.
- Concurrent client requests. Each Impala daemon can handle multiple concurrent client requests. The effects on performance depend on your particular hardware and workload.
- Kerberos authentication. For more information, see Impala Security.
- Partitions. With Impala SQL, you can create partitioned tables with the `CREATE TABLE` statement, and add and drop partitions with the `ALTER TABLE` statement. Impala also takes advantage of the partitioning present in Hive tables. See Partitioning for details.

What features from relational databases or Hive are not available in Impala?

- Querying streaming data.

- Deleting individual rows. You delete data in bulk by overwriting an entire table or partition, or by dropping a table.
- Indexing (not currently). LZO-compressed text files can be indexed outside of Impala, as described in Using LZO-Compressed Text Files.
- Full text search on text fields. The Cloudera Search product is appropriate for this use case.
- Custom Hive Serializer/Deserializer classes (SerDes). Impala supports a set of common native file formats that have built-in SerDes in CDH. See How Impala Works with Hadoop File Formats for details.
- Checkpointing within a query. That is, Impala does not save intermediate results to disk during long-running queries. Currently, Impala cancels a running query if any host on which that query is executing fails. When one or more hosts are down, Impala reroutes future queries to only use the available hosts, and Impala detects when the hosts come back up and begins using them again. Because a query can be submitted through any Impala node, there is no single point of failure. In the future, we will consider adding additional work allocation features to Impala, so that a running query would complete even in the presence of host failures.
- Encryption of data transmitted between Impala daemons.
- Window functions.
- Hive indexes.
- Non-Hadoop data stores, such as relational databases.

For the detailed list of features that are different between Impala and HiveQL, see SQL Differences Between Impala and Hive.

Does Impala support generic JDBC?

Impala supports the HiveServer2 JDBC driver.

Is Avro supported?

Yes, Avro is supported. Impala can query Avro tables. Currently, you must create such tables and load the data within Hive. See Using the Avro File Format with Impala Tables for details.

Impala Use Cases

What are good use cases for Impala as opposed to Hive or MapReduce?

Impala is well-suited to executing SQL queries for interactive exploratory analytics on large data sets. Hive and MapReduce are appropriate for very long running, batch-oriented tasks such as ETL.

Is MapReduce required for Impala? Will Impala continue to work as expected if MapReduce is stopped?

Impala does not use MapReduce at all.

Can Impala be used for complex event processing?

For example, in an industrial environment, many agents may generate large amounts of data. Can Impala be used to analyze this data, checking for notable changes in the environment?

Complex Event Processing (CEP) is usually performed by dedicated stream-processing systems. Impala is not a stream-processing system, as it most closely resembles a relational database.

Is Impala intended to handle real time queries in low-latency applications or is it for ad hoc queries for the purpose of data exploration?

Ad-hoc queries are the primary use case for Impala. We anticipate it being used in many other situations where low-latency is required. Whether Impala is appropriate for any particular use-case depends on the workload, data size and query volume. See Impala Benefits for the primary benefits of using Impala.

Questions about Impala And Hive

How does Impala compare to Hive and Pig?

Impala is different from Hive and Pig because it uses its own daemons that are spread across the cluster for queries. Because Impala does not rely on MapReduce, it avoids the startup overhead of MapReduce jobs, allowing Impala to return results in real time.

Can I do transforms or add new functionality?

Impala adds support for UDFs in Impala 1.2. You can write your own functions in C++, or reuse existing Java-based Hive UDFs. The UDF support includes scalar functions and user-defined aggregate functions (UDAs). User-defined table functions (UDTFs) are not currently supported.

Impala does not currently support an extensible serialization-deserialization framework (SerDes), and so adding extra functionality to Impala is not as straightforward as for Hive or Pig.

Can any Impala query also be executed in Hive?

Yes. There are some minor differences in how some queries are handled, but Impala queries can also be completed in Hive. Impala SQL is a subset of HiveQL, with some functional limitations such as transforms. For details of the Impala SQL dialect, see SQL Statements. For the Impala built-in functions, see Built-in Functions. For the detailed list of differences between Impala and HiveQL, see SQL Differences Between Impala and Hive.

Can I use Impala to query data already loaded into Hive and HBase?

There are no additional steps to allow Impala to query tables managed by Hive, whether they are stored in HDFS or HBase. Make sure that Impala is configured to access the Hive metastore correctly and you should be ready to go. Keep in mind that `impalad`, by default, runs as the `impala` user, so you might need to adjust some file permissions depending on how strict your permissions are currently.

See Using Impala to Query HBase Tables for details about querying data in HBase.

Is Hive an Impala requirement?

The Hive metastore service is a requirement. Impala shares the same metastore database as Hive, allowing Impala and Hive to access the same tables transparently.

Hive itself is optional, and does not need to be installed on the same nodes as Impala. Currently, Impala supports a wider variety of read (query) operations than write (insert) operations; you use Hive to insert data into tables that use certain file formats. See How Impala Works with Hadoop File Formats for details.

How do I?

How do I prevent users from seeing the text of SQL queries?

For instructions on making the Impala log files unreadable by unprivileged users, see Securing Impala Data and Log Files.

For instructions on password-protecting the web interface to the Impala log files and other internal server information, see Securing the Impala Web User Interface.

How do I know how many Impala nodes are in my cluster?

The Impala statestore keeps track of how many `impalad` nodes are currently available. You can see this information through the statestore web interface. For example, at the URL `http://statestore_host:25010/metrics` you might see lines like the following:

```
statestore.live-backends:3
statestore.live-backends.list:[host1:22000, host1:26000, host2:22000]
```

The number of `impalad` nodes is the number of list items referring to port 22000, in this case two. (Typically, this number is one less than the number reported by the `statestore.live-backends` line.) If an `impalad` node became unavailable or came back after an outage, the information reported on this page would change appropriately.

Impala Availability

Is Impala production ready?

Impala has finished its beta release cycle, and the 1.0, 1.1, and 1.2 GA releases are production ready. The 1.1.x series includes additional security features for authorization, an important requirement for production use in many organizations. The 1.2.x series includes important performance features, particularly for large join queries. Some Cloudera customers are already using Impala for large workloads.

The Impala 1.3.0 release is bundled with CDH 5.0.0. While most functionality with Impala 1.3 and CDH 5 is production-ready, the Llama component needed to use Impala in combination with the YARN resource manager is still in beta state.

How do I configure Hadoop High Availability (HA) for Impala?

You can set up a proxy server to relay requests back and forth to the Impala servers, for load balancing and high availability. See Using Impala through a Proxy for High Availability for details.

You can enable HDFS HA for the Hive metastore. See the CDH4 High Availability Guide CDH5 High Availability Guide for details.

What happens if there is an error in Impala?

There is not a single point of failure in Impala. All Impala daemons are fully able to handle incoming queries. If a machine fails however, all queries with fragments running on that machine will fail. Because queries are expected to return quickly, you can just rerun the query if there is a failure. See Impala Concepts and Architecture for details about the Impala architecture.

The longer answer: Impala must be able to connect to the Hive metastore. Impala aggressively caches metadata so the metastore host should have minimal load. Impala relies on the HDFS NameNode, and, in CDH4, you can configure HA for HDFS. Impala also has centralized services, known as the statestore and catalog services, that run on one host only. Impala continues to execute queries if the statestore host is down, but it will not get state updates. For example, if a host is added to the cluster while the statestore host is down, the existing instances of `impalad` running on the other hosts will not find out about this new host. Once the statestore process is restarted, all the information it serves is automatically reconstructed from all running Impala daemons.

What is the maximum number of rows in a table?

There is no defined maximum. Some customers have used Impala to query a table with over a trillion rows.

Can Impala and MapReduce jobs run on the same cluster without resource contention?

Yes. See Controlling Resource Usage for how to control Impala resource usage using the Linux cgroup mechanism, and Using YARN Resource Management with Impala (CDH 5 Only) for how to use Impala with the YARN resource

management framework. Impala is designed to run on the DataNode hosts. Any contention depends mostly on the cluster setup and workload.

For a detailed example of configuring a cluster to share resources between Impala queries and MapReduce jobs, see Setting up a Multi-tenant Cluster for Impala and MapReduce

Impala Internals

On which hosts does Impala run?

Cloudera strongly recommends running the `impalad` daemon on each DataNode for good performance. Although this topology is not a hard requirement, if there are data blocks with no Impala daemons running on any of the hosts containing replicas of those blocks, queries involving that data could be very inefficient. In that case, the data must be transmitted from one host to another for processing by "remote reads", a condition Impala normally tries to avoid. See Impala Concepts and Architecture for details about the Impala architecture. Impala schedules query fragments on all hosts holding data relevant to the query, if possible.

How are joins performed in Impala?

By default, Impala automatically determines the most efficient order in which to join tables using a cost-based method, based on their overall size and number of rows. (This is a new feature in Impala 1.2.2 and higher.) The `COMPUTE STATS` statement gathers information about each table that is crucial for efficient join performance. Impala chooses between two techniques for join queries, known as "broadcast joins" and "partitioned joins". See Joins for syntax details and Performance Considerations for Join Queries for performance considerations.

How does Impala process join queries for large tables?

Impala utilizes multiple strategies to allow joins between tables and result sets of various sizes. When joining a large table with a small one, the data from the small table is transmitted to each node for intermediate processing. When joining two large tables, the data from one of the tables is divided into pieces, and each node processes only selected pieces. See Joins for details about join processing, Performance Considerations for Join Queries for performance considerations, and Hints for how to fine-tune the join strategy.

What is Impala's aggregation strategy?

Impala currently only supports in-memory hash aggregation.

How is Impala metadata managed?

Impala uses two pieces of metadata: the catalog information from the Hive metastore and the file metadata from the NameNode. Currently, this metadata is lazily populated and cached when an `impalad` needs it to plan a query.

The REFRESH statement updates the metadata for a particular table after loading new data through Hive. The INVALIDATE METADATA Statement statement refreshes all metadata, so that Impala recognizes new tables or other DDL and DML changes performed through Hive.

In Impala 1.2 and higher, a dedicated `catalogd` daemon broadcasts metadata changes due to Impala DDL or DML statements to all nodes, reducing or eliminating the need to use the `REFRESH` and `INVALIDATE METADATA` statements.

What load do concurrent queries produce on the NameNode?

The load Impala generates is very similar to MapReduce. Impala contacts the NameNode during the planning phase to get the file metadata (this is only run on the host the query was sent to). Every `impalad` will read files as part of normal processing of the query.

How does Impala achieve its performance improvements?

These are the main factors in the performance of Impala versus that of other Hadoop components and related technologies.

Impala avoids MapReduce. While MapReduce is a great general parallel processing model with many benefits, it is not designed to execute SQL. Impala avoids the inefficiencies of MapReduce in these ways:

- Impala does not materialize intermediate results to disk. SQL queries often map to multiple MapReduce jobs with all intermediate data sets written to disk.
- Impala avoids MapReduce start-up time. For interactive queries, the MapReduce start-up time becomes very noticeable. Impala runs as a service and essentially has no start-up time.
- Impala can more naturally disperse query plans instead of having to fit them into a pipeline of map and reduce jobs. This enables Impala to parallelize multiple stages of a query and avoid overheads such as sort and shuffle when unnecessary.

Impala uses a more efficient execution engine by taking advantage of modern hardware and technologies:

- Impala generates runtime code. Impala uses LLVM to generate assembly code for the query that is being run. Individual queries do not have to pay the overhead of running on a system that needs to be able to execute arbitrary queries.
- Impala uses available hardware instructions when possible. Impala uses the latest set of SSE (SSE4.2) instructions which can offer tremendous speedups in some cases.
- Impala uses better I/O scheduling. Impala is aware of the disk location of blocks and is able to schedule the order to process blocks to keep all disks busy.
- Impala is designed for performance. A lot of time has been spent in designing Impala with sound performance-oriented fundamentals, such as tight inner loops, inlined function calls, minimal branching, better use of cache, and minimal memory usage.

What happens when the data set exceeds available memory?

Currently, if the memory required to process intermediate results on a node exceeds the amount available to Impala on that node, the query is cancelled. You can adjust the memory available to Impala on each node, and you can fine-tune the join strategy to reduce the memory required for the biggest queries. We do plan on supporting external joins and sorting in the future.

Keep in mind though that the memory usage is not directly based on the input data set size. For aggregations, the memory usage is the number of rows *after* grouping. For joins, the memory usage is the combined size of the tables *excluding* the biggest table, and Impala can use join strategies that divide up large joined tables among the various nodes rather than transmitting the entire table to each node.

What are the most memory-intensive operations?

If a query fails with an error indicating "memory limit exceeded", you might suspect a memory leak. The problem could actually be a query that is structured in a way that causes Impala to allocate more memory than you expect, exceeded the memory allocated for Impala on a particular node. Some examples of query or table structures that are especially memory-intensive are:

- `INSERT` statements using dynamic partitioning, into a table with many different partitions. (Particularly for tables using Parquet format, where the data for each partition is held in memory until it reaches 1 GB in size before it is written to disk.) Consider breaking up such operations into several different `INSERT` statements, for example to load data one year at a time rather than for all years at once.
- `GROUP BY` on a unique or high-cardinality column. Impala allocates some handler structures for each different value in a `GROUP BY` query. Having millions of different `GROUP BY` values could exceed the memory limit.
- Queries involving very wide tables, with thousands of columns, particularly with many `STRING` columns. Because Impala allows a `STRING` value to be up to 32 KB, the intermediate results during such queries could require substantial memory allocation.

When does Impala hold on to or return memory?

Impala allocates memory using tcmalloc, a memory allocator that is optimized for high concurrency. Once Impala allocates memory, it keeps that memory reserved to use for future queries. Thus, it is normal for Impala to show high memory usage when idle. If Impala detects that it is about to exceed its memory limit (defined by the `-mem_limit` startup option or the `MEM_LIMIT` query option), it deallocates memory not needed by the current queries.

When issuing queries through the JDBC or ODBC interfaces, make sure to call the appropriate close method afterwards. Otherwise, some memory associated with the query is not freed.

Impala Performance

Are results returned as they become available, or all at once when a query completes?

Impala streams results whenever they are available, when possible. Certain SQL operations (aggregation or `ORDER BY`) require all of the input to be ready before Impala can return results.

Why does my query run slowly?

There are many possible reasons why a given query could be slow. Use the following checklist to diagnose performance issues with existing queries, and to avoid such issues when writing new queries, setting up new nodes, creating new tables, or loading data.

- Immediately after the query finishes, issue a `SUMMARY` command in `impala-shell`. You can check which phases of execution took the longest, and compare estimated values for memory usage and number of rows with the actual values.
- Immediately after the query finishes, issue a `PROFILE` command in `impala-shell`. The numbers in the `BytesRead`, `BytesReadLocal`, and `BytesReadShortCircuit` should be identical for a specific node. For example:

  ```
  - BytesRead: 180.33 MB
  - BytesReadLocal: 180.33 MB
  - BytesReadShortCircuit: 180.33 MB
  ```

  If `BytesReadLocal` is lower than `BytesRead`, something in your cluster is misconfigured, such as the `impalad` daemon not running on all the data nodes. If `BytesReadShortCircuit` is lower than `BytesRead`, short-circuit reads are not enabled properly on that node; see Post-Installation Configuration for Impala for instructions.
- If the table was just created, or this is the first query that accessed the table after an `INVALIDATE METADATA` statement or after the `impalad` daemon was restarted, there might be a one-time delay while the metadata for the table is loaded and cached. Check whether the slowdown disappears when the query is run again. When doing performance comparisons, consider issuing a `DESCRIBE` *`table_name`* statement for each table first, to make sure any timings only measure the actual query time and not the one-time wait to load the table metadata.
- Is the table data in uncompressed text format? Check by issuing a `DESCRIBE FORMATTED` *`table_name`* statement. A text table is indicated by the line:

  ```
  InputFormat: org.apache.hadoop.mapred.TextInputFormat
  ```

  Although uncompressed text is the default format for a `CREATE TABLE` statement with no `STORED AS` clauses, it is also the bulkiest format for disk storage and consequently usually the slowest format for queries. For data where query performance is crucial, particularly for tables that are frequently queried, consider starting with or converting to a compact binary file format such as Parquet, Avro, RCFile, or SequenceFile. For details, see How Impala Works with Hadoop File Formats.
- If your table has many columns, but the query refers to only a few columns, consider using the Parquet file format. Its data files are organized with a column-oriented layout that lets queries minimize the amount of

I/O needed to retrieve, filter, and aggregate the values for specific columns. See Using the Parquet File Format with Impala Tables for details.

- If your query involves any joins, are the tables in the query ordered so that the tables or subqueries are ordered with the one returning the largest number of rows on the left, followed by the smallest (most selective), the second smallest, and so on? That ordering allows Impala to optimize the way work is distributed among the nodes and how intermediate results are routed from one node to another. For example, all other things being equal, the following join order results in an efficient query:

```
select some_col from
    huge_table join small_table join medium_table join big_table
  where
    huge_table.id = big_table.id
    and big_table.id = medium_table.id
    and medium_table.id = small_table.id;
```

  See Performance Considerations for Join Queries for performance tips for join queries.
- Also for join queries, do you have table statistics for the table, and column statistics for the columns used in the join clauses? Column statistics let Impala better choose how to distribute the work for the various pieces of a join query. See How Impala Uses Statistics for Query Optimization for details about gathering statistics.
- Does your table consist of many small data files? Impala works most efficiently with data files in the multi-megabyte range; Parquet, a format optimized for data warehouse-style queries, uses 1 GB files with a 1 GB block size. Use the `DESCRIBE FORMATTED` *`table_name`* statement in `impala-shell` to see where the data for a table is located, and use the `hadoop fs -ls` or `hdfs dfs -ls` Unix commands to see the files and their sizes. If you have thousands of small data files, that is a signal that you should consolidate into a smaller number of large files. Use an `INSERT ... SELECT` statement to copy the data to a new table, reorganizing into new data files as part of the process. Prefer to construct large data files and import them in bulk through the `LOAD DATA` or `CREATE EXTERNAL TABLE` statements, rather than issuing many `INSERT ... VALUES` statements; each `INSERT ... VALUES` statement creates a separate tiny data file. If you have thousands of files all in the same directory, but each one is megabytes in size, consider using a partitioned table so that each partition contains a smaller number of files. See the following point for more on partitioning.
- If your data is easy to group according to time or geographic region, have you partitioned your table based on the corresponding columns such as `YEAR`, `MONTH`, and/or `DAY`? Partitioning a table based on certain columns allows queries that filter based on those same columns to avoid reading the data files for irrelevant years, postal codes, and so on. (Do not partition down to too fine a level; try to structure the partitions so that there is still sufficient data in each one to take advantage of the multi-megabyte HDFS block size.) See Partitioning for details.

Why does my SELECT statement fail?

When a `SELECT` statement fails, the cause usually falls into one of the following categories:

- A timeout because of a performance, capacity, or network issue affecting one particular node.
- Excessive memory use for a join query, resulting in automatic cancellation of the query.
- A low-level issue affecting how native code is generated on each node to handle particular `WHERE` clauses in the query. For example, a machine instruction could be generated that is not supported by the processor of a certain node. If the error message in the log suggests the cause was an illegal instruction, consider turning off native code generation temporarily, and trying the query again.
- Malformed input data, such as a text data file with an enormously long line, or with a delimiter that does not match the character specified in the `FIELDS TERMINATED BY` clause of the `CREATE TABLE` statement.

Why does my INSERT statement fail?

When an `INSERT` statement fails, it is usually the result of exceeding some limit within a Hadoop component, typically HDFS.

- An `INSERT` into a partitioned table can be a strenuous operation due to the possibility of opening many files and associated threads simultaneously in HDFS. Impala 1.1.1 includes some improvements to distribute the

work more efficiently, so that the values for each partition are written by a single node, rather than as a separate data file from each node.

- Certain expressions in the `SELECT` part of the `INSERT` statement can complicate the execution planning and result in an inefficient `INSERT` operation. Try to make the column data types of the source and destination tables match up, for example by doing `ALTER TABLE ... REPLACE COLUMNS` on the source table if necessary. Try to avoid `CASE` expressions in the `SELECT` portion, because they make the result values harder to predict than transferring a column unchanged or passing the column through a built-in function.
- Be prepared to raise some limits in the HDFS configuration settings, either temporarily during the `INSERT` or permanently if you frequently run such `INSERT` statements as part of your ETL pipeline.
- The resource usage of an `INSERT` statement can vary depending on the file format of the destination table. Inserting into a Parquet table is memory-intensive, because the data for each partition is buffered in memory until it reaches 1 gigabyte, at which point the data file is written to disk. Impala can distribute the work for an `INSERT` more efficiently when statistics are available for the source table that is queried during the `INSERT` statement. See How Impala Uses Statistics for Query Optimization for details about gathering statistics.

Does Impala performance improve as it is deployed to more hosts in a cluster in much the same way that Hadoop performance does?

Yes. Impala scales with the number of hosts. It is important to install Impala on all the data nodes in the cluster, because otherwise some of the nodes must do remote reads to retrieve data not available for local reads. Data locality is an important architectural aspect for Impala performance. See this Impala performance blog post for background. Note that this blog post refers to benchmarks with Impala 1.1.1; Impala has added even more performance features in the 1.2.x series.

Is the HDFS block size reduced to achieve faster query results?

No. Impala does not make any changes to the HDFS or HBase data sets.

The default Parquet block size is relatively large (1 GB), and you can control the block size when creating Parquet files using the PARQUET_FILE_SIZE query option.

Does Impala use caching?

Impala does not cache data but it does cache some table and file metadata. Although queries might run faster on subsequent iterations because the data set was cached in the OS buffer cache, Impala does not explicitly control this.

SQL

Is there an UPDATE statement?

Impala does not currently have an `UPDATE` statement, which would typically be used to change a single row, a small group of rows, or a specific column. The HDFS-based files used by typical Impala queries are optimized for bulk operations across many megabytes of data at a time, making traditional `UPDATE` operations inefficient or impractical.

You can use the following techniques to achieve the same goals as the familiar `UPDATE` statement, in a way that preserves efficient file layouts for subsequent queries:

- Replace the entire contents of a table or partition with updated data that you have already staged in a different location, either using `INSERT OVERWRITE`, `LOAD DATA`, or manual HDFS file operations followed by a `REFRESH` statement for the table. Optionally, you can use built-in functions and expressions in the `INSERT` statement to transform the copied data in the same way you would normally do in an `UPDATE` statement, for example to turn a mixed-case string into all uppercase or all lowercase.

- To update a single row, use an HBase table, and issue an `INSERT ... VALUES` statement using the same key as the original row. Because HBase handles duplicate keys by only returning the latest row with a particular key value, the newly inserted row effectively hides the previous one.

Can Impala do user-defined functions (UDFs)?

Impala 1.2 and higher does support UDFs and UDAs. You can either write native Impala UDFs and UDAs in C++, or reuse UDFs (but not UDAs) originally written in Java for use with Hive. See User-Defined Functions (UDFs) for details.

Why do I have to use REFRESH and INVALIDATE METADATA, what do they do?

In Impala 1.2 and higher, there is much less need to use the `REFRESH` and `INVALIDATE METADATA` statements:

- The new `impala-catalog` service, represented by the `catalogd` daemon, broadcasts the results of Impala DDL statements to all Impala nodes. Thus, if you do a `CREATE TABLE` statement in Impala while connected to one node, you do not need to do `INVALIDATE METADATA` before issuing queries through a different node.
- The catalog service only recognizes changes made through Impala, so you must still issue a `REFRESH` statement if you load data through Hive or by manipulating files in HDFS, and you must issue an `INVALIDATE METADATA` statement if you create a table, alter a table, add or drop partitions, or do other DDL statements in Hive.
- Because the catalog service broadcasts the results of `REFRESH` and `INVALIDATE METADATA` statements to all nodes, in the cases where you do still need to issue those statements, you can do that on a single node rather than on every node, and the changes will be automatically recognized across the cluster, making it more convenient to load balance by issuing queries through arbitrary Impala nodes rather than always using the same coordinator node.

Why is space not freed up when I issue DROP TABLE?

Impala deletes data files when you issue a `DROP TABLE` on an internal table, but not an external one. By default, the `CREATE TABLE` statement creates internal tables, where the files are managed by Impala. An external table is created with a `CREATE EXTERNAL TABLE` statement, where the files reside in a location outside the control of Impala. Issue a `DESCRIBE FORMATTED` statement to check whether a table is internal or external. The keyword `MANAGED_TABLE` indicates an internal table, from which Impala can delete the data files. The keyword `EXTERNAL_TABLE` indicates an external table, where Impala will leave the data files untouched when you drop the table.

Even when you drop an internal table and the files are removed from their original location, you might not get the hard drive space back immediately. By default, files that are deleted in HDFS go into a special trashcan directory, from which they are purged after a period of time (by default, 6 hours). For background information on the trashcan mechanism, see http://archive.cloudera.com/cdh4/cdh/4/hadoop/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. For information on purging files from the trashcan, see http://archive.cloudera.com/cdh4/cdh/4/hadoop/hadoop-project-dist/hadoop-common/FileSystemShell.html.

When Impala deletes files and they are moved to the HDFS trashcan, they go into an HDFS directory owned by the `impala` user. If the `impala` user does not have an HDFS home directory where a trashcan can be created, the files are not deleted or moved, as a safety measure. If you issue a `DROP TABLE` statement and find that the table data files are left in their original location, create an HDFS directory `/user/impala`, owned and writeable by the `impala` user. For example, you might find that `/user/impala` is owned by the `hdfs` user, in which case you would switch to the `hdfs` user and issue a command such as:

```
hdfs dfs -chown -R impala /user/impala
```

Is there a DUAL table?

You might be used to running queries against a single-row table named DUAL to try out expressions, built-in functions, and UDFs. Impala does not have a DUAL table. To achieve the same result, you can issue a SELECT statement without any table name:

```
select 2+2;
select substr('hello',2,1);
select pow(10,6);
```

Partitioned Tables

How do I load a big CSV file into a partitioned table?

To load a data file into a partitioned table, when the data file includes fields like year, month, and so on that correspond to the partition key columns, use a two-stage process. First, use the LOAD DATA or CREATE EXTERNAL TABLE statement to bring the data into an unpartitioned text table. Then use an INSERT ... SELECT statement to copy the data from the unpartitioned table to a partitioned one. Include a PARTITION clause in the INSERT statement to specify the partition key columns. The INSERT operation splits up the data into separate data files for each partition. For examples, see Partitioning. For details about loading data into partitioned Parquet tables, a popular choice for high-volume data, see Loading Data into Parquet Tables.

Can I do INSERT ... SELECT * into a partitioned table?

When you use the INSERT ... SELECT * syntax to copy data into a partitioned table, the columns corresponding to the partition key columns must appear last in the columns returned by the SELECT *. You can create the table with the partition key columns defined last. Or, you can use the CREATE VIEW statement to create a view that reorders the columns: put the partition key columns last, then do the INSERT ... SELECT * from the view.

HBase

What kinds of Impala queries or data are best suited for HBase?

HBase tables are ideal for queries where normally you would use a key-value store. That is, where you retrieve a single row or a few rows, by testing a special unique key column using the = or IN operators.

HBase tables are not suitable for queries that produce large result sets with thousands of rows. HBase tables are also not suitable for queries that perform full table scans because the WHERE clause does not request specific values from the unique key column.

Use HBase tables for data that is inserted one row or a few rows at a time, such as by the INSERT ... VALUES syntax. Loading data piecemeal like this into an HDFS-backed table produces many tiny files, which is a very inefficient layout for HDFS data files.

If the lack of an UPDATE statement in Impala is a problem for you, you can simulate single-row updates by doing an INSERT ... VALUES statement using an existing value for the key column. The old row value is hidden; only the new row value is seen by queries.

HBase tables are often wide (containing many columns) and sparse (with most column values NULL). For example, you might record hundreds of different data points for each user of an online service, such as whether the user had registered for an online game or enabled particular account features. With Impala and HBase, you could look up all the information for a specific customer efficiently in a single query. For any given customer, most of these columns might be NULL, because a typical customer might not make use of most features of an online service.