BIG DATA ANALYTICS

Hive

Introduction |

Comparison with Traditional Databases

https://www.youtube.com/c/RASINENIMADANAMOHANA

Hive: *Outline*

- Introduction
- Installing Hive
  - The Hive Shell
- An Example
- Running Hive
- Comparison with Traditional Databases
- HiveQL
- Tables
- Querying Data
- User-Defined Functions
  - Writing a User Defined Functions
  - Writing a User Defined Aggregate Function

Hive: Introduction

- In "Information Platforms and the Rise of the Data Scientist," Jeff Hammerbacher describes Information Platforms as "the focus of their organization's efforts to ingest, process, and generate information," and how they "serve to accelerate the process of learning from empirical data."
- One of the biggest ingredients in the Information Platform built by Jeff 's team at Facebook was Apache Hive, a framework for data warehousing on top of Hadoop.
- Hive grew from a need to manage and learn from the huge volumes of data that Facebook was producing every day from its growing social network.
- After trying a few different systems, the team chose Hadoop for storage and processing, since it was cost effective and met the scalability requirements.

Hive: Introduction

- Hive was created to make it possible for analysts with strong SQL skills (but inadequate Java programming skills) to run queries on the huge volumes of data that Facebook stored in HDFS.
- Today, Hive is a successful Apache project used by many organizations as a general-purpose, scalable data processing platform.
- Certainly, SQL isn't ideal for every big data problem-it's not a good fit for building complex machine-learning algorithms, for example-but it's great for many analyses, and it has the huge advantage of being very well known in the industry. What's more, SQL serves as the universal language across business intelligence tools (ODBC is a common bridge, for example), so Hive is well placed to integrate with these products.

Hive: *Outline*

- Installing Hive
  - The Hive Shell
- An Example
- Running Hive
- Comparison with Traditional Databases
- Hive QL
- Tables
- Querying Data
- User-Defined Functions
  - Writing a User Defined Functions
  - Writing a User Defined Aggregate Function

Hive: Installing Hive

- In normal use, Hive runs on our workstation and converts our SQL query into a series of jobs for execution on a Hadoop cluster.
- Hive organizes data into tables, which provide a means for attaching structure to data stored in HDFS.
- Metadata-such as table schemas-is stored in a database called the *metastore*.
- When starting out with Hive, it is convenient to run the metastore on our local machine. In this configuration, which is the default, the Hive table definitions that we create will be local to our machine, so we can't share them with other users.

Hive: Installing Hive

- Installation of Hive is straightforward.
- As a prerequisite, we need to have the same version of Hadoop installed locally that our cluster is running. We may choose to run Hadoop locally, either in standalone or pseudodistributed mode, while getting started with Hive.
- Download a release, and unpack the tarball in a suitable place on our workstation:

```
% tar xzf apache-hive-x.y.z-bin.tar.gz
```

- It's handy to put Hive on our path to make it easy to launch:

```
% export HIVE_HOME=~/sw/apache-hive-x.y.z-bin
% export PATH=$PATH:$HIVE_HOME/bin
```

- Now type `hive` to launch the Hive shell:

```
% hive
hive>
```

Hive: Installing Hive

The Hive Shell

- The shell is the primary way that we will interact with Hive, by issuing commands in **HiveQL**.
- **HiveQL** is Hive's query language, a language of SQL.
- It is heavily influenced by MySQL, so if we are familiar with MySQL, we should feel at home using Hive.
- When starting Hive for the first time, we can check that it is working by listing its tables - there should be none.
- The command must be terminated with a semicolon to tell Hive to execute it:

```
hive> SHOW TABLES;
OK
Time taken: 0.473 seconds
```

Like SQL, **HiveQL** is generally case insensitive (except for string comparisons), so `show tables;` works equally well here. The Tab key will autocomplete Hive keywords and functions.

Hive: Installing Hive

The Hive Shell

- We can also run the Hive shell in noninteractive mode. The `-f` option runs the commands in the specified file, which is `script.q` in this example:

```
% hive -f script.q
```

- For short scripts, you can use the `-e` option to specify the commands inline, in which case the final semicolon is not required:

```
% hive -e 'SELECT * FROM dummy'
OK
X
Time taken: 1.22 seconds, Fetched: 1 row(s)
```

Hive: Installing Hive

The Hive Shell

- In both interactive and noninteractive mode, Hive will print information to standard error-such as the time taken to run a query-during the course of operation.
- We can suppress these messages using the `-S` option at launch time, which has the effect of showing only the output result for queries:

```
% hive -S -e 'SELECT * FROM dummy'
X
```

- Other useful Hive shell features include the ability to run commands on the host operating system by using a `!` prefix to the command and the ability to access Hadoop filesystems using the `dfs` command.

Hive: *Outline*

- Installing Hive
  - The Hive Shell
- An Example
- Running Hive
- Comparison with Traditional Databases
- Hive QL
- Tables
- Querying Data
- User-Defined Functions
  - Writing a User Defined Functions
  - Writing a User Defined Aggregate Function

Hive: An Example

Let's see how to use Hive to run a query on the weather dataset

- The *first step* is to load the data into Hive's managed storage.
- Here we'll have Hive use the local filesystem for storage; later we'll see how to store tables in HDFS.
- Just like an RDBMS, Hive organizes its data into tables.
- We create a table to hold the weather data using the **`CREATE TABLE`** statement:

```
CREATE TABLE records (year STRING, temperature INT, quality INT)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t';
```

The first line declares a `records` table with three columns: `year`, `temperature`, and `quality`. The type of each column must be specified, too. Here the `year` is a string, while the other two columns are integers.

Hive: An Example

- So far, the SQL is familiar. The ROW FORMAT clause, however, is particular to HiveQL. This declaration is saying that each row in the data file is tab-delimited text. Hive expects there to be three fields in each row, corresponding to the table columns, with fields separated by tabs and rows by newlines.
- We can populate Hive with the data. This is just a small sample, for exploratory purposes:

```
LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'
OVERWRITE INTO TABLE records;
```

Running this command tells Hive to put the specified local file in its warehouse directory. This is a simple filesystem operation.

In this example, we are storing Hive tables on the local filesystem (`fs.defaultFS` is set to its default value of `file:///`). Tables are stored as directories under Hive's warehouse directory, which is controlled by the `hive.metastore.warehouse.dir` property and defaults to `/user/hive/warehouse`.

Hive: An Example

- Thus, the files for the records table are found in the `/user/hive/warehouse/records` directory on the local filesystem:

```
% ls /user/hive/warehouse/records/
```

```
sample.txt
```

The OVERWRITE keyword in the LOAD DATA statement tells Hive to delete any existing files in the directory for the table. If it is omitted, the new files are simply added to the table's directory (unless they have the same names, in which case they replace the old files).

Now that the data is in Hive, we can run a query against it:

```
hive> SELECT year, MAX(temperature) FROM records WHERE temperature
!= 9999 AND quality IN (0, 1, 4, 5, 9) GROUP BY year;
```

```
1949 111
```

```
1950 22
```

Hive: *Outline*

- Installing Hive
  - The Hive Shell
- An Example
- Running Hive 
- Comparison with Traditional Databases
- Hive QL
- Tables
- Querying Data
- User-Defined Functions
  - Writing a User Defined Functions
  - Writing a User Defined Aggregate Function

Running Hive

- Configuring Hive
- Hive Services
- The Metastore

Running Hive: Configuring Hive

- Hive is configured using an XML configuration file like Hadoop's.
- The file is called ***hivesite.xml*** and is located in Hive's conf directory.
- This file is where we can set properties that we want to set every time we run Hive.
- The same directory contains ***hivedefault.xml***, which documents the properties that Hive exposes and their default values.
- We can override the configuration directory that Hive looks for in ***hive-site.xml*** by passing the **`--config`** option to the hive command:

```
% hive --config /Users/tom/dev/hive-conf
```

Note that this option specifies the containing directory, not ***hive-site.xml*** itself. It can be useful when we have multiple site files—for different clusters, say-that we switch between on a regular basis. Alternatively, we can set the **`HIVE_CONF_DIR`** environment variable to the configuration directory for the same effect.

Running Hive: Configuring Hive

- The ***hive-site.xml*** file is a natural place to put the cluster connection details: we can specify the filesystem and resource manager using the usual Hadoop properties, **fs.defaultFS** and **yarn.resourcemanager.address**
- If not set, they default to the local filesystem and the local (inprocess) job runner-just like they do in Hadoop-which is very handy when trying out Hive on small trial datasets. Metastore configuration settings are commonly found in ***hive-site.xml***, too.
- Hive also permits us to set properties on a per-session basis, by passing the **-hiveconf** option to the hive command.
- For example, the following command sets the cluster (in this case, to a pseudodistributed cluster) for the duration of the session:

```
% hive -hiveconf fs.defaultFS=hdfs://localhost \
-hiveconf mapreduce.framework.name=yarn \
-hiveconf yarn.resourcemanager.address=localhost:8032
```

Running Hive: Configuring Hive

- We can change settings from within a session, too, using the SET command. This is useful for changing Hive settings for a particular query.
- For example, the following command ensures buckets are populated according to the table definition:

```
hive> SET hive.enforce.bucketing=true;
```

- To see the current value of any property, use SET with just the property name:

```
hive> SET hive.enforce.bucketing;
hive.enforce.bucketing=true
```

By itself, SET will list all the properties (and their values) set by Hive.

Note that the list will not include Hadoop defaults, unless they have been explicitly overridden.

Use **`SET -v`** to list all the properties in the system, including Hadoop defaults.

Running Hive: Configuring Hive

- There is a precedence hierarchy to setting properties.
- In the following list, lower numbers take precedence over higher numbers:
  1. The `Hive SET` command
  2. The command-line `-hiveconf` option
  3. `hive-site.xml` and the Hadoop site files (`core-site.xml`, `hdfs-site.xml`, `mapredsite.xml`, and `yarn-site.xml`)
  4. The Hive defaults and the Hadoop default files (`core-default.xml`, `hdfs-default.xml`, `mapred-default.xml`, and `yarn-default.xml`)

Running Hive: Configuring Hive

Execution engines

- Hive was originally written to use MapReduce as its execution engine, and that is still the default.
- It is now also possible to run Hive using Apache Tez as its execution engine, and work is underway to support Spark, too.
- Both Tez and Spark are general directed acyclic graph (DAG) engines that offer more flexibility and higher performance than MapReduce.
- For example, unlike MapReduce, where intermediate job output is materialized to HDFS, Tez and Spark can avoid replication overhead by writing the intermediate output to local disk, or even store it in memory (at the request of the Hive planner).

Running Hive: Configuring Hive

Execution engines

- The execution engine is controlled by the **`hive.execution.engine`** property, which defaults to **`mr`** (for MapReduce).
- It's easy to switch the execution engine on a per-query basis, so we can see the effect of a different engine on a particular query.
- Set Hive to use Tez as follows:

```
hive> SET hive.execution.engine=tez;
```

Note that Tez needs to be installed on the Hadoop cluster first.

Running Hive: Configuring Hive

Logging

- We can find Hive's error log on the local filesystem at ***${java.io.tmpdir}/${user.name}/hive.log***.
- It can be very useful when trying to diagnose configuration problems or other types of error. Hadoop's MapReduce task logs are also a useful resource for troubleshooting.
- On many systems, ***${java.io.tmpdir}*** is ***/tmp***, but if it's not, or if we want to set the logging directory to be another location, then use the following:

```
% hive -hiveconf hive.log.dir='/tmp/${user.name}'
```

- The logging configuration is in ***conf/hive-log4j.properties***, and we can edit this file to change log levels and other logging-related settings.
- However, often it's more convenient to set logging configuration for the session.
- For example, the following handy invocation will send debug messages to the console:

```
% hive -hiveconf hive.root.logger=DEBUG,console
```

Running Hive

- Configuring Hive
- Hive Services
- The Metastore

Running Hive: Hive Services

- The Hive shell is only one of several services that we can run using the hive command.
- We can specify the service to run using the `--service option`.
- Type `hive -service help` to get a list of available service names.
- Some of the most useful ones are described below:

`cli`

The command-line interface to Hive (the shell). This is the default service.

Running Hive: Hive Services

Hiveserver2

Runs Hive as a server exposing a Thrift service, enabling access from a range of clients written in different languages. **HiveServer 2** improves on the original Hive-Server by supporting authentication and multiuser concurrency. Applications using the Thrift, JDBC, and ODBC connectors need to run a Hive server to communicate with Hive. Set the **hive.server2.thrift.port** configuration property to specify the port the server will listen on (defaults to 10000).

Beeline

A command-line interface to Hive that works in embedded mode (like the regular CLI), or by connecting to a **HiveServer 2** process using JDBC.

Running Hive: Hive Services

hwi

The Hive Web Interface. A simple web interface that can be used as an alternative to the CLI without having to install any client software. See also Hue for a more fully featured Hadoop web interface that includes applications for running Hive queries and browsing the Hive metastore.

jar

The Hive equivalent of **hadoop jar**, a convenient way to run Java applications that includes both Hadoop and Hive classes on the classpath.

metastore

By default, the metastore is run in the same process as the Hive service. Using this service, it is possible to run the metastore as a standalone (remote) process. Set the `METASTORE_PORT` environment variable (or use the **-p** command-line option) to specify the port the server will listen on (defaults to `9083`).

Running Hive: Hive Services

Hive clients

If we run Hive as a server (`hive --service hiveserver2`), there are a number of different mechanisms for connecting to it from applications (the relationship between Hive clients and Hive services is illustrated in below Figure):

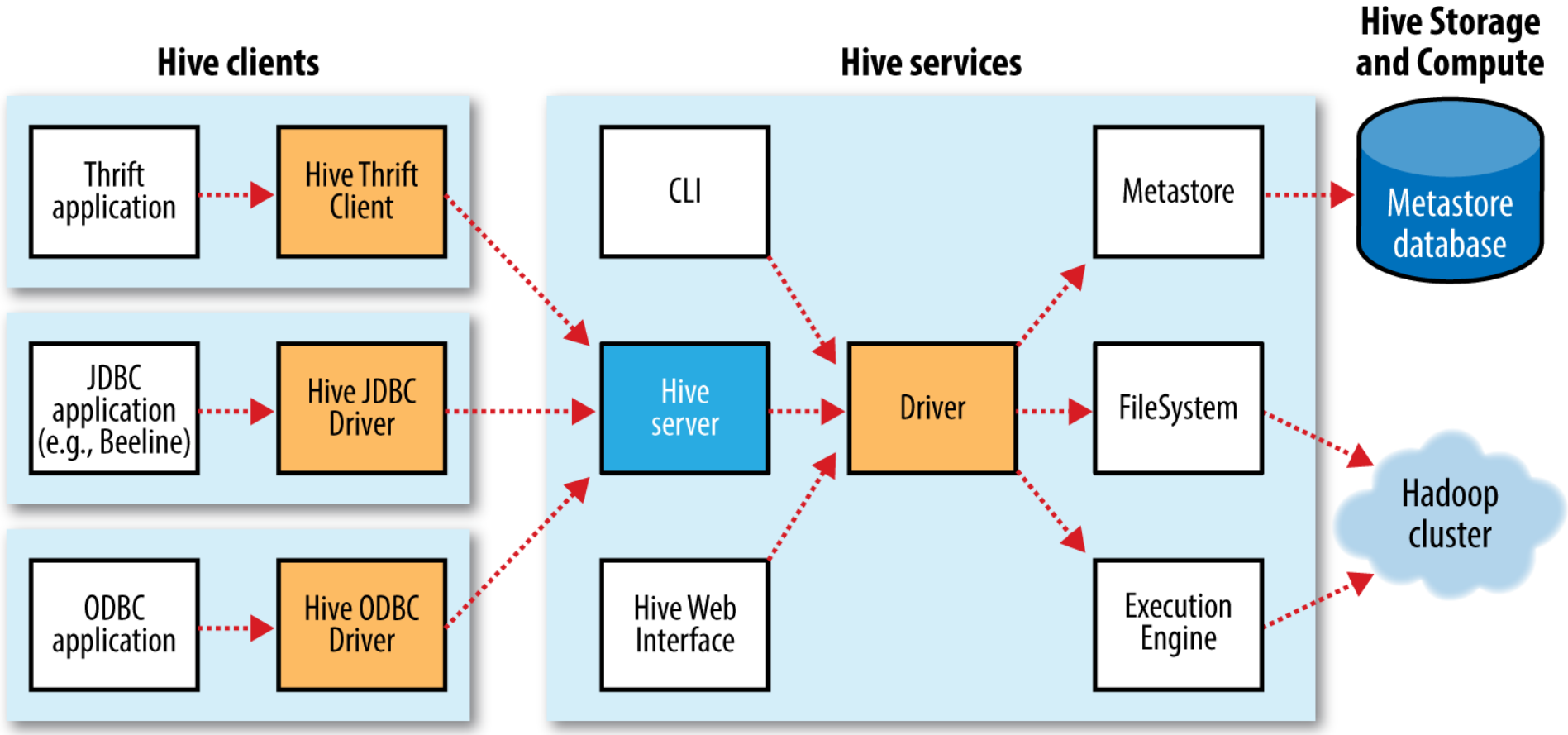

Figure. Hive architecture

Running Hive: Hive Services

Hive clients

Thrift Client

The Hive server is exposed as a Thrift service, so it's possible to interact with it using any programming language that supports Thrift. There are third-party projects providing clients for Python and Ruby.

JDBC driver

Hive offers a Type 4 (pure Java) JDBC driver, which can be used to connect a Java application to a Hive server. The driver is defined in the class '`org.apache.hadoop.hive.jdbc.HiveDriver`'. We can connect using two methods:

1. Remote Mode: By configuring a JDBC URI in the form '`jdbc:hive2://host:port/dbname`', our Java application connects to a Hive server running on the specified host and port. The driver uses the Hive Thrift Client for communication.
2. Embedded Mode: By using the URI '`jdbc:hive2://`', Hive runs within the same JVM as the Java application. This mode doesn't require the Hive server to run as a separate process and doesn't use the Thrift service.

The Beeline CLI uses this JDBC driver to interact with Hive.

Running Hive: Hive Services

Hive clients

ODBC driver

An ODBC driver allows applications that support the ODBC protocol (such as business intelligence software) to connect to Hive. The Apache Hive distribution does not ship with an ODBC driver, but several vendors make one freely available. (Like the JDBC driver, ODBC drivers use Thrift to communicate with the Hive server.)

Running Hive

- Configuring Hive
- Hive Services
- The Metastore

Running Hive: The Metastore

- The metastore is the central repository of Hive metadata.
- The metastore is divided into two pieces: a service and the backing store for the data.
- By default, the metastore service runs in the same JVM as the Hive service and contains an embedded Derby database instance backed by the local disk. This is called the **`embedded metastore`** configuration (see *below Figure*).

Running Hive: The Metastore

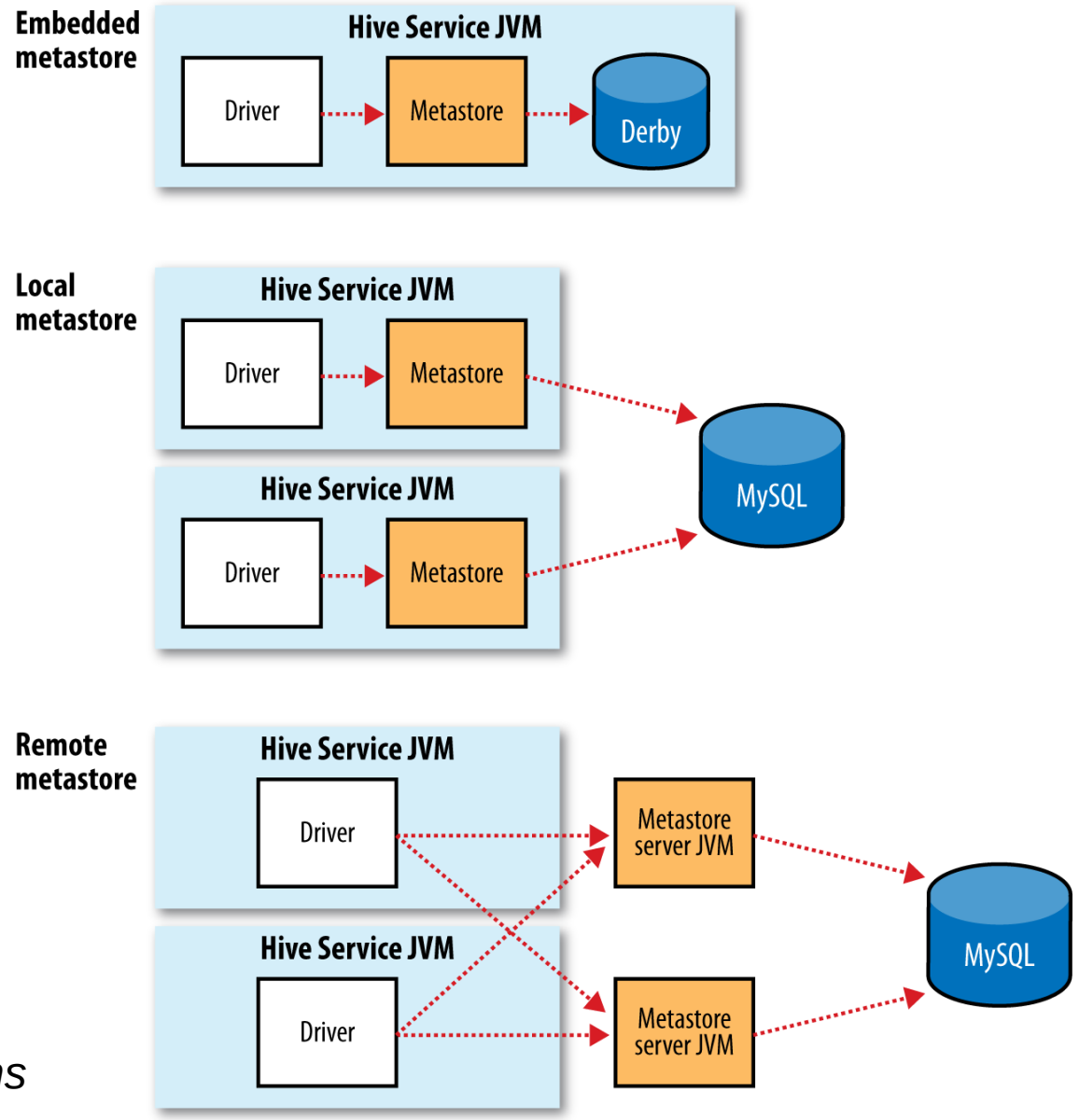

Figure. Metastore configurations

Running Hive: The Metastore

- Using an embedded metastore is a simple way to get started with Hive; however, only one embedded Derby database can access the database files on disk at any one time, which means we can have only one Hive session open at a time that accesses the same metastore. Trying to start a second session produces an error when it attempts to open a connection to the metastore.
- The solution to supporting multiple sessions (and therefore multiple users) is to use a standalone database. This configuration is referred to as a local metastore, since the metastore service still runs in the same process as the Hive service but connects to a database running in a separate process, either on the same machine or on a remote machine. Any JDBC-compliant database may be used by setting the `javax.jdo.option`.
- Configuration properties listed in below Table.

Running Hive: The Metastore

Table: *Important metastore configuration properties*

Property name	Type	Default value	Description
hive.metastore.warehouse.dir	URI	/user/hive/ warehouse	The directory relative to fs.defaultFS where managed tables are stored.
hive.metastore.uris	Comma-separated URIs	Not set	If not set (the default), use an in-process metastore; otherwise, connect to one or more remote metastores, specified by a list of URIs. Clients connect in a round-robin fashion when there are multiple remote servers.
javax.jdo.option.ConnectionURL	URI	jdbc:derby:;databaseName=metastore_db;create=true	The JDBC URL of the metastore database.
javax.jdo.option.ConnectionDriverName	String	org.apache.derby.jdbc.EmbeddedDriver	The JDBC driver classname.
javax.jdo.option.ConnectionUserName	String	APP	The JDBC username.
javax.jdo.option.ConnectionPassword	String	mine	The JDBC password.

Running Hive: The Metastore

- MySQL is a popular choice for the standalone metastore.
- In this case, the `javax.jdo.option.ConnectionURL` property is set to `jdbc:mysql://host/dbname?createDatabaseIfNotExist=true`, and `javax.jdo.option.ConnectionDriverName` is set to `com.mysql.jdbc.Driver`. (The username and password should be set too, of course.)
- The JDBC driver JAR file for MySQL (Connector/J) must be on Hive's classpath, which is simply achieved by placing it in Hive's `lib` directory.
- There's another metastore configuration called a remote metastore, where one or more metastore servers run in separate processes to the Hive service. This brings better manageability and security because the database tier can be completely firewalled off, and the clients no longer need the database credentials.

Running Hive: The Metastore

- A Hive service is configured to use a remote metastore by setting `hive.metastore.uris` to the metastore server URI(s), separated by commas if there is more than one.
- Metastore server URIs are of the form `thrift://host:port`, where the port corresponds to the one set by METASTORE_PORT when starting the metastore server.

Hive: *Outline*

- Installing Hive
  - The Hive Shell
- An Example
- Running Hive
- Comparison with Traditional Databases 
- HiveQL
- Tables
- Querying Data
- User-Defined Functions
  - Writing a User Defined Functions
  - Writing a User Defined Aggregate Function

Hive: Comparison with Traditional Databases

The comparison between Hive and traditional databases focusing on the specific points of Schema Enforcement, Updates/Transactions/Indexes, and SQL-on-Hadoop Alternatives:

Feature	Hive	Traditional Databases
1. Schema Enforcement	**Schema on Read:** • Validates data at query time. • Faster initial load. • Flexible; supports multiple schemas for the same data.	**Schema on Write:** • Validates data during loading. • Slower initial load. • Rigid; schema must be defined before data load.
2. Updates, Transactions, and Indexes	**Updates & Transactions:** • Initially limited, now supports INSERT, UPDATE, DELETE. • Changes stored in delta files, merged periodically. • Requires transactions enabled on tables. **Indexes:** • Limited support with compact and bitmap indexes. • Primarily used for specific query optimizations. • Indexing is pluggable, allowing for custom implementations.	**Updates & Transactions:** • Full support for real-time updates, transactions. • In-place updates and consistent transaction handling. **Indexes:** • Comprehensive indexing options (e.g., B-tree, Hash). • Extensive indexing for performance optimization.

Hive: Comparison with Traditional Databases

The comparison between Hive and traditional databases focusing on the specific points of Schema Enforcement, Updates/Transactions/Indexes, and SQL-on-Hadoop Alternatives:

Feature	Hive	Traditional Databases
3. SQL-on-Hadoop Alternatives	**SQL Engines:** • Hive integrates with other SQL-on-Hadoop engines like Impala, Presto, and Spark SQL. • Alternative engines often outperform Hive in specific scenarios (e.g., interactive queries). • SQL-on-Hadoop engines offer various optimizations and execution models. **Execution Engines:** • Hive initially used MapReduce; now supports Tez and vectorized queries.	**SQL Engines:** • Typically uses a single, highly optimized SQL engine. • Designed for either OLTP or OLAP, not both. **Execution Engines:** • Custom-built for high performance and tailored to the specific database architecture.

Hive: Comparison with Traditional Databases

HIVE Vs SQL

HIVE	SQL
Hive is a SQL-like scripting language built on MapReduce	According to ANSI, SQL is the standard language for RDMBS, used to communicate with databases
Used for analytics	Used for transactional processing(OLTP) & analytics
Data per query in PBs	Data per query in GBs
Faster execution while performing analytics on Huge data sets compared to SQL	Slower execution while performing analytics on huge data sets compared to HIVE
No Normalization required	Supports Normalization

Hive: Comparison with Traditional Databases

Pig Vs. Hive

	PIG	HIVE
1	Pig is data flow language	HIVE is declarative language(HiveQL)
2	Pig is used for programming	Mainly used for creating reports
3	PIG is best for semi structured data	Hive is best for structured Data
4	Pig does not have any metadata database and schema is defined while loading data	Hive defines tables schema before + stores schema information in database
5	No web interface	It has HWI(Hive web interface)
6	Doesn't support JDBC	Provides support for JDBC
7	Schema is created implicitly	Schema should mentioned explictly
8	Suitable for programmers and researchers	Suitable for data analysts
9	Pig Hadoop Component operates on the client side of any cluster	Hive Hadoop Component operates on the server side of any cluster
10	Pig can be installed easily over Hive	Need some configuration to setup