SPA Conference 2016

Real-World Big Data in Action

Nick Rozanski
Eoin Woods
Chris Cooper-Bland

Number 2 in an occasional series

Goals of Today

Goals of Today

- Learn why Big Data is important
- Look at some of the techniques and technologies that underpin Big Data implementations
- Install, configure and run some Big Data software (Hadoop, Spark and Hive)
- Do some data science on real Big Data datasets
    - In particular, find out which are the most expensive and cheapest parts of London to live in

What Today *Isn't* About

- Detailed explanation of how Big Data works
- Learning how to configure Big Data software in real production environments
- I am not a Big Data technology expert!

Agenda

Agenda

- Introduction to Big Data (15 minutes)
- Exercise 1: Hadoop (60 minutes including setup)
- Exercise 2: Spark (30 minutes)
- Exercise 3: Hive (45 minutes)

Real-World Big Data in Action

Introduction to Big Data

History and Background

The Origins of Big Data

The "Three V's"

- First proposed by META Group analyst Doug Laney in 2001 (https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf)
- Driven by the (then) looming growth in e-commerce and the strain this would impose on computer systems

Volume

- The increase in depth and breadth of data

Velocity

- The speed at which data needs to be made available for use

Variety

- Problems caused by incompatible data formats, non-aligned data structures and inconsistent data semantics

- Subsequent analysts have added additional V's such as **Variability** and **Veracity** (data quality)

A Brief (and Partial) History of Big Data

Hadoop

- Doug Cutting and others started working on a web crawler called Nutch in 2002
- This eventually morphed into Hadoop and was adopted by Yahoo! in 2006
- It became a top-level Apache project and was adopted by Last.fm, Facebook and others
- The Yahoo web map comprised 100 billion nodes and 1 trillion edges by 2009
- Hadoop continued to break records for volume and velocity: in 2014, a team from Databricks sorted 100TB of data in 1,406 seconds on 207 nodes (4.27TB per min)

Fun Facts

- Facebook is rumoured to manage 100 PB of data
  - that's 21 million DVDs, or 70 billion 3.5" floppy disks
- Hadoop is a made-up name (the name of Doug Cutting's daughter's yellow toy elephant)

A Brief (and Partial) History of Big Data

Spark

- Started at UC Berkeley's AMPLab in 2009 and open sourced in 2010
- A Top-Level Apache Project since 2014
- Now used at many organisations (https://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark)

Hive

- Originally developed at Facebook around 2007-8
- Now used at Netflix, FINRA (UK regulator), CNET, Digg, eHarmony, last.fm … (https://cwiki.apache.org/confluence/display/Hive/PoweredBy)

Today's Big Data Landscape

All Big Data tools are required by EU Law to have ridiculous names

- Sqoop (data integration)
- Oozie (workflow)
- Pig (scripting)
- Impala (MPP SQL query engine)
- Flume (data ingestion)
- Parquet (columnar storage)
- ZooKeeper (distributed application management)

A Simple Big Data Stack

Hive

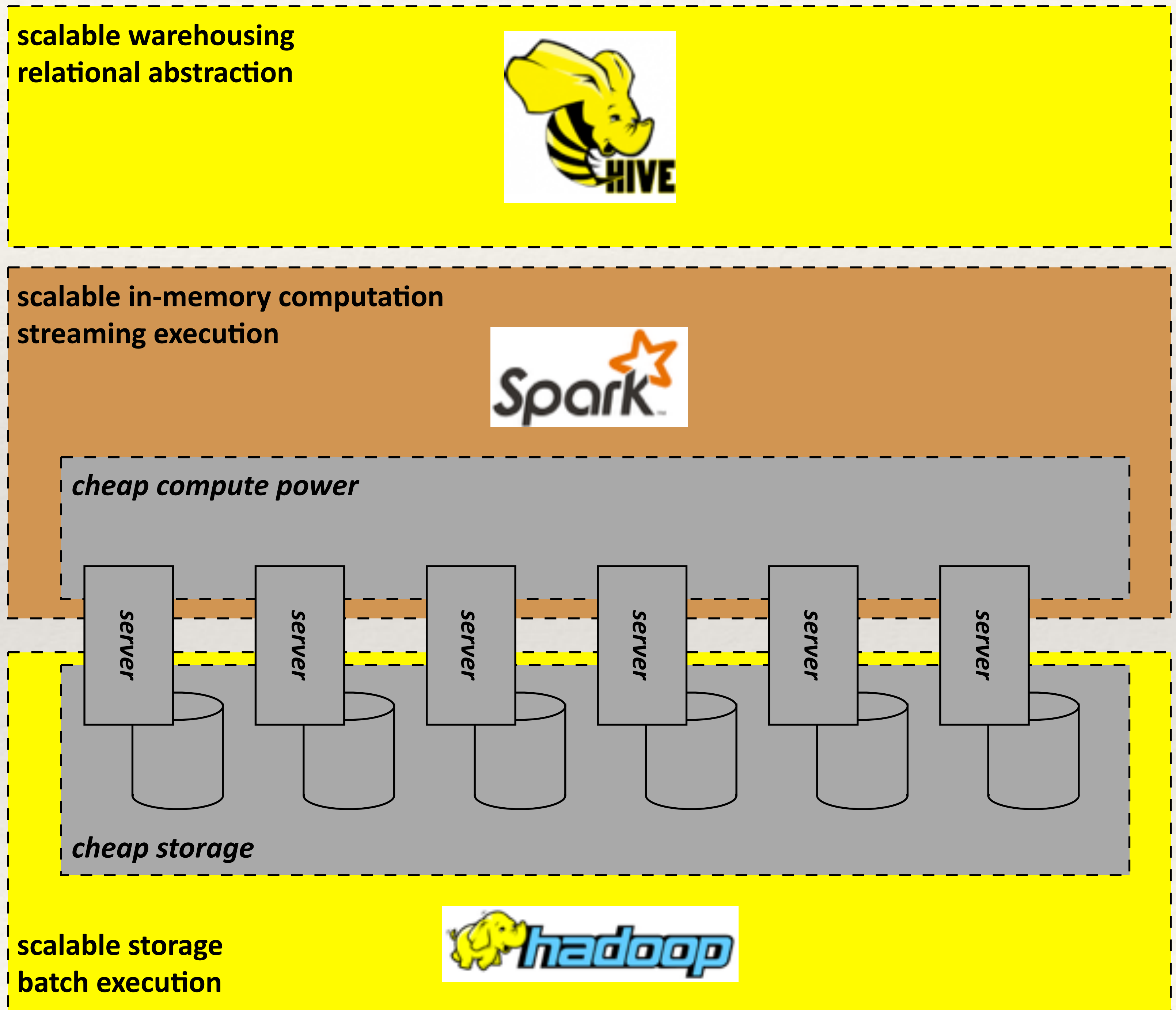
- scaleable data warehousing infrastructure providing a SQL abstraction on top of Hadoop and optionally Spark

Spark

- scaleable in-memory cluster computing framework with implicit data parallelism and fault-tolerance

Hadoop

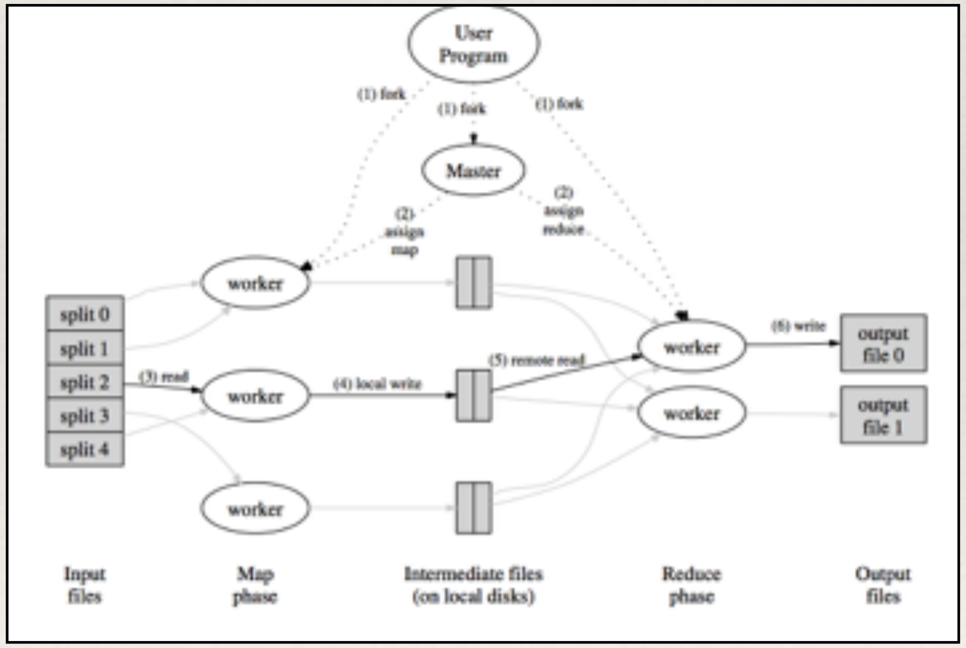
- massively scalable file management and batch execution on a commodity hardware platform



The MapReduce Paradigm

MapReduce

- Paper published by Google in 2004 (http://research.google.com/archive/mapreduce.html)
- Describes how they rewrote their production indexing system using MapReduce
- Provides automatic parallelization and distribution, fault-tolerance, I/O scheduling and status monitoring



Map Reduce Steps

- *Map step*: master breaks up query and distributes portions across a massive number of computers
- *Reduce step*: results collated and returned to requestor

Benchmark

- Scan 10 billion 100-byte records to extract records matching a rare pattern (92K matching records): once started up, 1800 machines read 1 TB of data at peak of ~31 GB/s

Real-World Big Data in Action

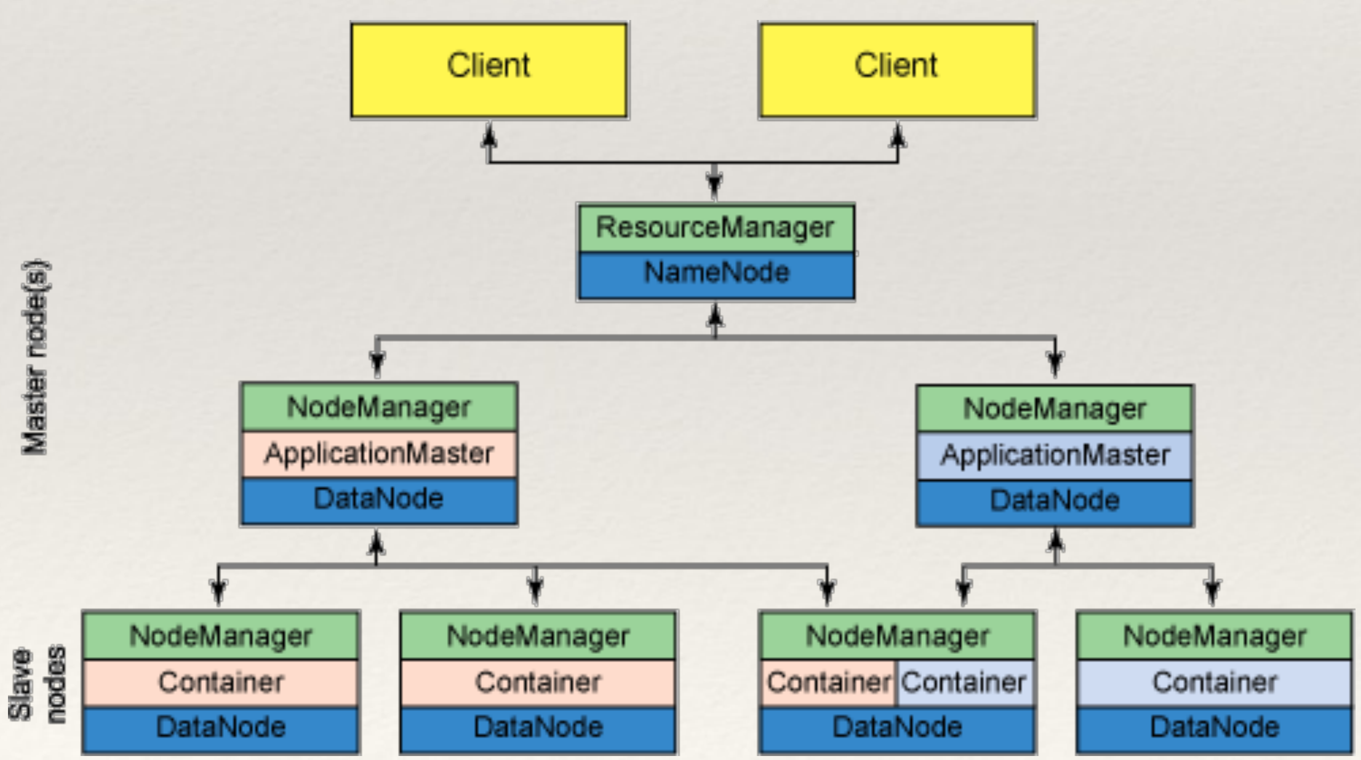
Hadoop

A Big Data Virtual Filesystem

Hadoop Overview

HDFS (Hadoop Distributed Filesystem)

- distributed, scalable, and portable file-system for Hadoop
- stores large files (gigabytes to terabytes or more) across multiple machines using commodity hardware
- achieves reliability by replicating the data across multiple hosts
- presents a POSIX-like filesystem API



https://www.ibm.com/developerworks/library/bd-hadoopyarn/

YARN (Yet Another Resource Negotiator)

- YARN is the new cluster resource management system introduced in Hadoop v2
- it includes the following components:

YARN Resource Manager

- orchestrates the division of resources to NodeManagers

Hadoop NameNode

- provides metadata services

YARN NodeManagers

- monitor and launch containers

ApplicationMaster

- manages an instance of an application

Hadoop DataNodes

- provide replicated storage services across the cluster

Containers

- runs an application on a node

Exercise 1: Hadoop

Goals of This Exercise

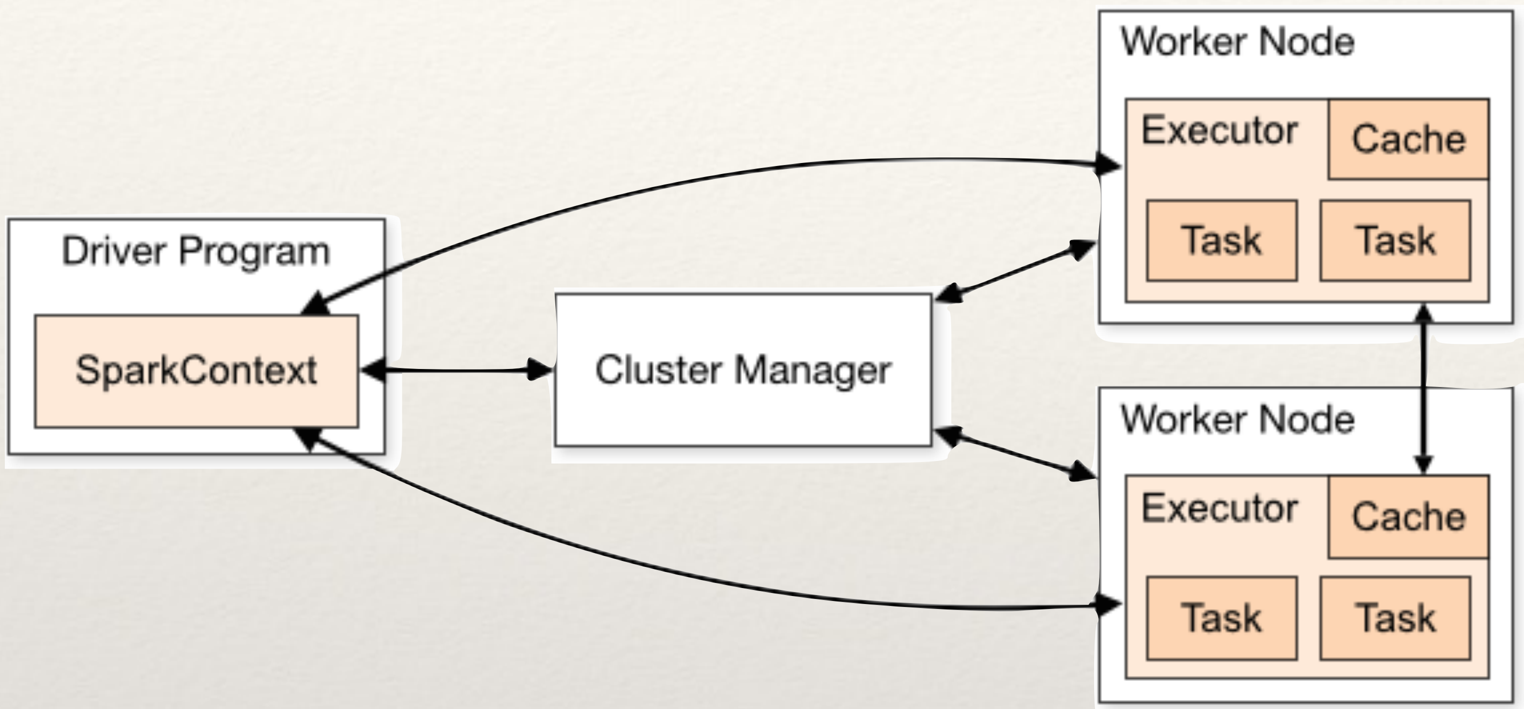
1. Install the Big Data software (Hadoop, Spark and Hive)
2. Configure the Big Data software
3. Start Hadoop and check it is running
4. Format the Hadoop filesystem
5. Load a file into Hadoop
6. Browse the Hadoop filesystem

Real-World Big Data in Action

Spark

A Big Data Cluster Computing Framework

Spark Overview



https://spark.apache.org/docs/latest/cluster-overview.html

- Spark has a master / slave architecture in which a *Driver* on the Spark Master communicates with a large number of *Executors* on Spark Workers (aka slaves)
- The Driver and Executors are together termed a *Spark application*
- Spark primitives support in-memory computing which can bring performance advantages over Hadoop for some workloads

Spark Applications

- Spark applications run as independent sets of processes on a cluster, coordinated by a `SparkContext` object in the main program
- the `SparkContext` connects to a cluster manager such as YARN
- Spark acquires *executors* on cluster (processes that run computations and store data for your application)
- It then sends the application code to the executors
- Finally, the SparkContext sends tasks to the executors to run

RDDs and Data Frames

Resilient Distributed Datasets (RDDs)

- The central data abstraction in Spark is the *Resilient Distributed Dataset* (RDD): a fault-tolerant collection of objects that is partitioned across multiple nodes in a cluster and can be operated on in parallel
- A Spark program typically has the following structure:
    - load one or more RDDs from an external source, such as Hadoop
    - perform one or more transformations on the RDDs
    - perform one or more actions on the resulting RDDs (such as computing a result or saving to persistent storage)

DataFrames

- A DataFrame is a distributed collection of data organized into named columns
- It is conceptually equivalent to a table in a relational database
- A program can access a DataFrame programmatically, or using Spark SQL / Hive SQL
- DataFrames claim to offer:
    - The ability to scale from kilobytes of data on a single laptop to petabytes on a large cluster
    - Support for a wide array of data formats and storage systems
    - State-of-the-art optimization and code generation through the Spark SQL Catalyst optimizer
    - Seamless integration with all big data tooling and infrastructure via Spark
    - APIs for Python, Java, Scala, and R
- (*see https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html*)

Spark Programming 101

Spark APIs

- Spark provides data processing APIs for Java, Scala and Python
- We'll be using the Python API today by running the PySpark tool
- The Spark methods generally take DataFrames as input and often return DataFrames as output
- They can be "chained" together as shown below

The `SqlContext` Class

- The entry point into Spark "relational" functionality is the `SqlContext` class
- PySpark automatically creates a `SqlContext` object for you called `sqlContext`:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc) # this is done for you
```

- You use this object to create a DataFrame from file using the `load()` method

```
myvar = sqlContext.read.format(…).option(…).
    load("hdfs://localhost:9000/path/to/file")
```

- You can set options by calling the `option()` method one or more times (eg to specify an input file format)
- There are other methods you can use to further process DataFrames

Spark Programming 101 (continued)

`sqlContext` Operations

- Once the `sqlContext` object is in memory, you call methods to perform relational operations on it:
  - `printSchema()`
  - `filter(myvar.column == somevalue)` # like a SQL WHERE clause
  - `limit(`*`number_of_rows_to_return`*`)`
  - `groupBy("column_name").count()` # SQL GROUP BY functionality
  - `rollup("column1", "column2", …)` # multidimensional GROUP BY
  - `sorted(), sort()` # sorts DataFrames
  - `show(`*`truncate=False`*`)` # prints rows to `stdout` (which is what we will do today)

Spark SQL

- You can run Spark SQL commands directly on structured data (eg parquet, a columnar data format which structures its data in directories) using `sqlContext.sql("SELECT …")`

Further Information

- The Spark API reference is here: https://spark.apache.org/docs/1.5.2/api/python/pyspark.sql.html#pyspark.sql.DataFrame

Exercise 2: Spark

Goals of This Exercise

1. Start Hadoop and Spark and check they are running
2. Start the Spark client
3. Load the Hadoop file into a Spark RDD
4. Do some data science!

Real-World Big Data in Action

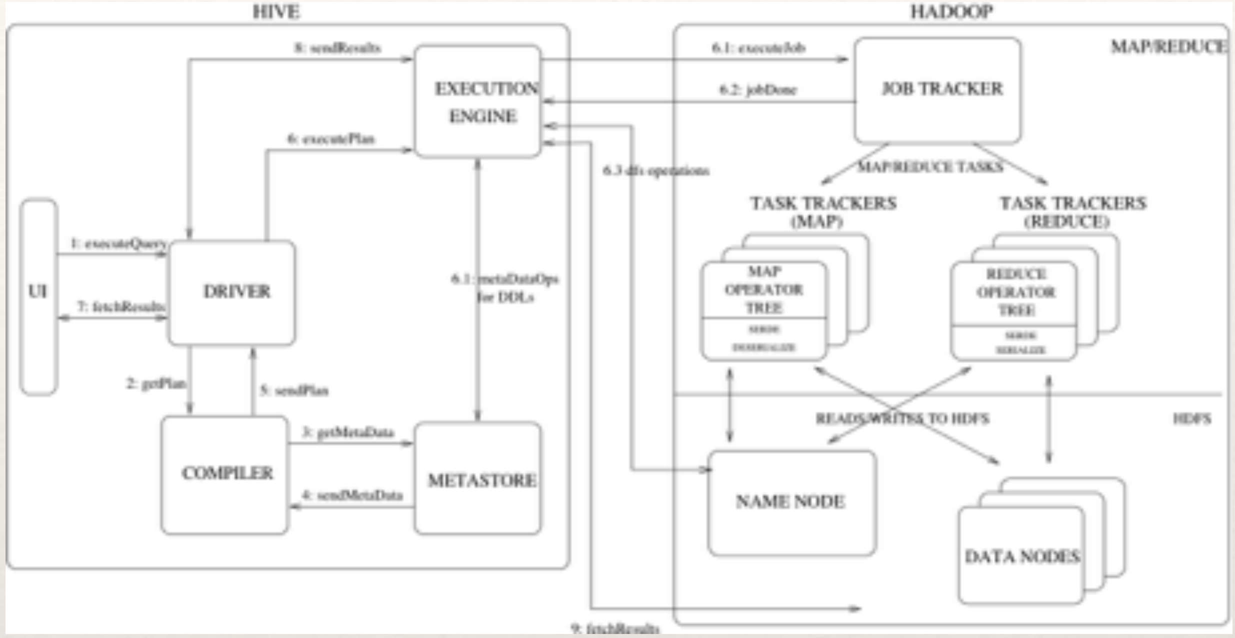
Hive

A Big Data data warehousing infrastructure

Hive Overview

Hive Capabilities

- Performs query analysis of large datasets stored in HDFS
- Driven by a SQL-like language called HiveQL
- Supports indexing to accelerate queries
- Based on schema on read (when query issued)
- Some update support (concurrency, locking, transactionality etc.) but not designed for online processing
- Incorporates a cost-base optimiser called Apache Calcite



https://cwiki.apache.org/confluence/display/Hive/Design

Hive Architecture

- Hive transparently converts queries to MapReduce, Spark or Tez jobs
- Hive usually moves data into its repository in HDFS (managed tables)
- External tables leave data in place (we will use these today for simplicity)
- Hive metadata is stored in a repository called the *metastore*
    - For this session we will use Apache Derby (a small-footprint embedded RDBMS)
    - In production you would typically use MySQL

Exercise 3: Hive

Goals of This Exercise

1. Start Hadoop, Spark and Hive and check they are running
2. Start the Hive client
3. Create an external Hive table from a Hadoop file
4. Do some data science!