UNIVERSITY of WASHINGTON

BIGDATA 210: Introduction to Data Engineering

Autumn 2018

Module 2: Hadoop and the Hadoop Ecosystem

Jerry Kuch

jkuch@uw.edu

© Jason Kolter

Week 2 Agenda

- Week 1 Review / Questions?
- Introduction to Scalable Computing
- The Hadoop Ecosystem
- Sandbox Environment / Assignment 1
  - Hortonworks Hadoop cluster sandbox
  - Ambari cluster manager
  - Lab assignment: *Fun with Hive*

© Jason Kolter

Last Week…

- Objectives, Schedule, Logistics
- Introduction to Big Data
- History and Context for our Coverage:
  - Remarks on Classic DBs
  - GFS/MapReduce, Hadoop…
- Cloud Computing / Enabling Technologies:
  - Utility computing and pay as you go via…
  - Virtualization
  - Containers

Last Week…

- Who did the suggested reading?
- Any comments or thoughts?
- Questions?
- Moving on…

Introduction to Scalable Computing

Distributed Systems, Big Data
and the Hadoop Ecosystem

Big Data and Distributed Computing

- Recall core of our definition for *Big Data…*
- Too big to fit on a single machine
- Must be *distributed* across multiple machines
- So:
  - Why is that a new problem?
  - What makes it hard?
  - What does it force us to do differently?

© Jason Kolter

Big Data and Distributed Computing

- *The Fallacies of Distributed Computing*
    1. The network is reliable
    2. Latency is zero
    3. Bandwidth is infinite
    4. The network is secure
    5. Topology doesn't change
    6. There is one administrator
    7. Transport cost is zero
    8. The network is homogeneous

© Jason Kolter

Big Data and Distributed Computing

- *Point of the Fallacies:* Distributed computing is hard!
- Data gets big enough => must be distributed.
- How do the fallacies affect us in practice?
- Consider an ***Example Scenario***:
  - You need to process a set of log files monthly to bill customers for service use

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

© Jason Kolter

Big Data:
A Motivating Problem

- **Case 1:** One reasonable sized file
  - Shell script/program reads data in memory and outputs to screen or file

- **Issues:**
  - Something goes wrong? Run job again

Big Data:
A Motivating Problem

- **Case 2a:** Many reasonable sized files
  - Shell script/program reads data in memory and outputs to screen or file
- **Issues and Responses Have Multiplied:**
  - Application crashes processing one file?
    - How many already processed?
    - How many left to process?
    - Current totals for each resource need maintained
    - Might run out of memory
    - **Serial processing is slow… so maybe parallelize?**

Big Data:
A Motivating Problem

- **Case 2b:** Many reasonable sized files
  - Multiple shell script/programs reads data in memory and outputs to screen or file
- **Issues**
  - Application crashes processing one file
    - How many already processed?
    - How many left to process?
    - Current totals for each resource need maintained
    - Might run out of memory
    - Post-process or coordinate parallel jobs

Big Data: A Motivating Problem

- **Case 3:** Very large files
  - Split into smaller files and run multiple scripts/ programs to read data in memory….
- **Issues**
  - Application crashes processing one file
    - How many already processed?
    - How many left to process?
    - Current totals for each resource need maintained
    - Need a pre-processing step to split files
    - Post-process or coordinate jobs

© Jason Kolter

Big Data: A Motivating Problem

- Common things that made our motivating problem harder as it got bigger:
  - Dealing with failure
    - Processes fail?
    - Things that store data die?
  - Coordinating activity
    - Starting, monitoring and restarting
    - Collecting results

Big Data: Solving these Problems Enter GFS and MapReduce

- Google File System (SOSP 2003)
  - https://research.google.com/archive/gfs.html

- Google MapReduce (OSDI 2004)
  - https://research.google.com/archive/mapreduce.html

© Jason Kolter

Big Data: Hadoop's Answers to GFS and MapReduce

Our Motivating Problem: Revisited in a Hadoop World

- **Our Problem Before:** *Handle very large files*
  - *HDFS* splits files across a *distributed file system*
  - *MapReduce* engine processes each chunk of data and aggregates results
- **Issues**
  - Complexity emerges in…
    - The code users must write
    - The operational work administrators must do

© Jason Kolter

HDFS

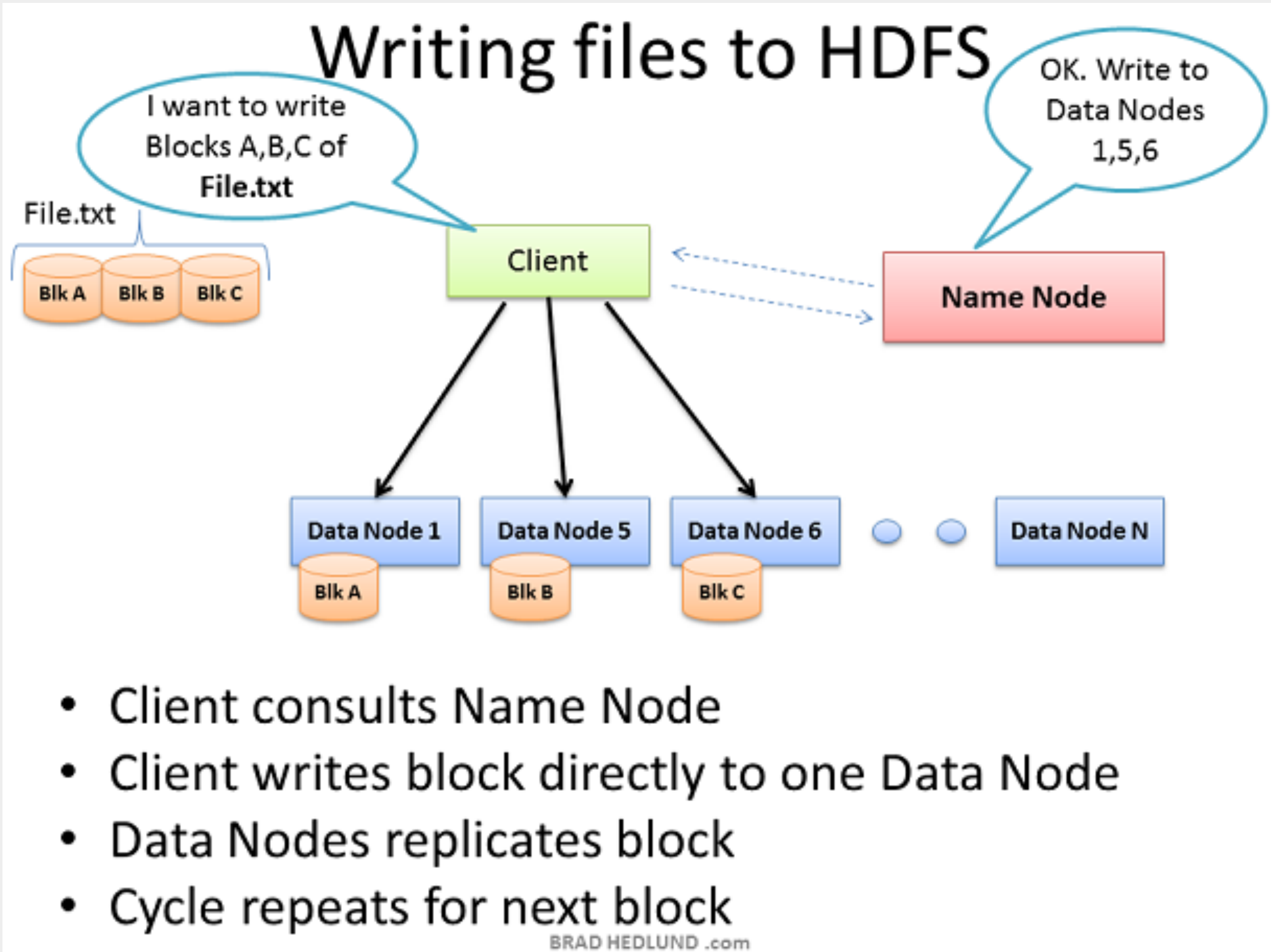
- **Purpose:** *Distribute very large files across multiple nodes of a cluster*
- **Components of a Running HDFS Cluster:**
  - **Name Node(s)**
    - Maintain filesystem *metadata*
    - Keep a standby NameNode for HA and redundancy
  - **Data Node(s)**
    - Store file blocks on local filesystems on nodes
    - Replicate blocks to multiple data nodes for redundancy

© Jason Kolter

HDFS: Immutability

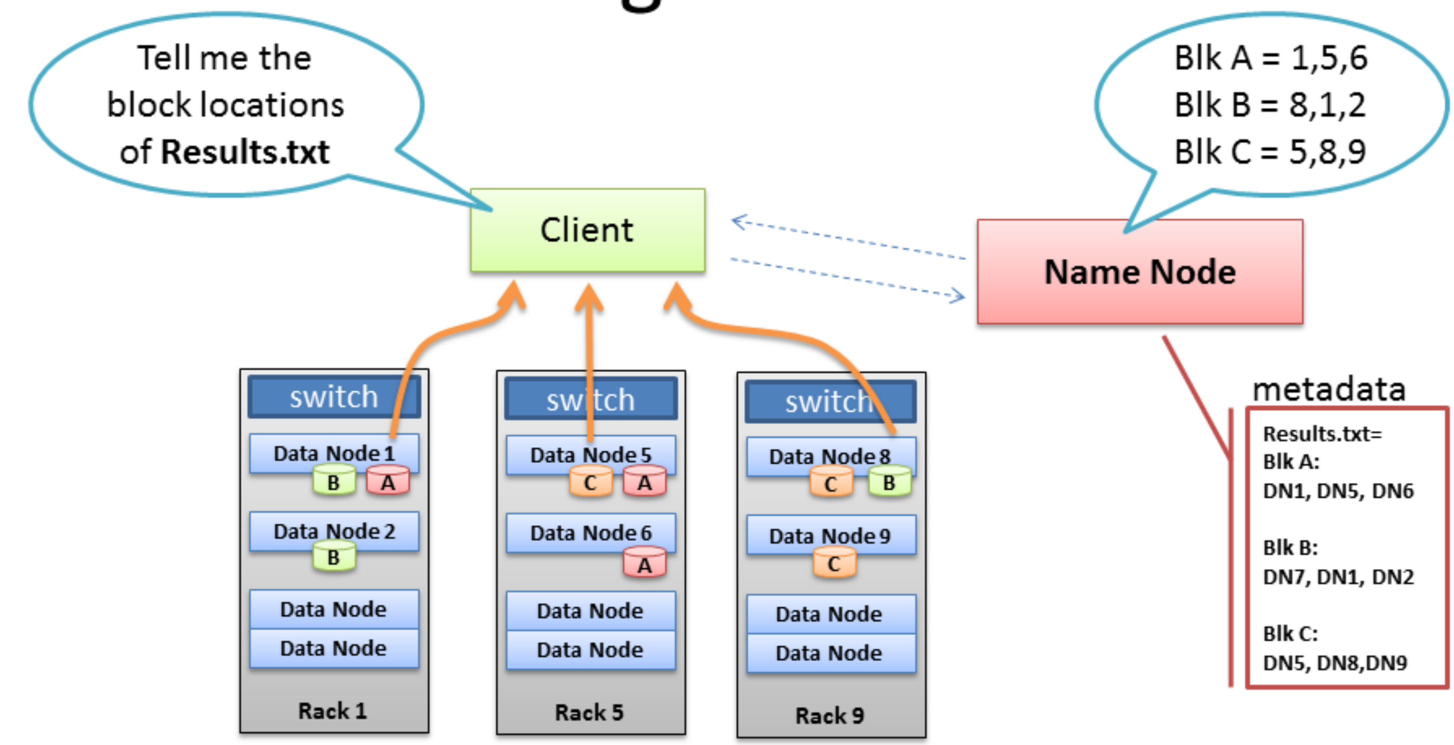
- ***Immutability:*** *once something is written, don’t change it in place!*
  - if you aren't modifying a file in place and you fail you don’t leave it messed up
  - if some or all of the file is replicated in multiple places and it doesn’t mutate synchronization is easy
- Immutability eases many things:
  - Replication
  - Concurrency
  - Fault tolerance
  - Data integrity, …
- …but at the potential price of:
  - Lots of I/O,
  - Lots of intermediate files in multi-stage jobs

HDFS: Writing a File



HDFS: Reading a File

Client reading files from HDFS



- Client receives Data Node list for each block
- Client picks first Data Node for each block
- Client reads blocks sequentially

BRAD HEDLUND .com

HDFS: The Command Line

- HDFS Command Line Client:
  - In the sandbox, meet the ‘hadoop’ command
  - Command various things, now we’ll look at HDFS access
  - `hadoop fs  # ‘fs’ subcommand`
    - `ls`
    - `rm`
    - `put`
    - `etc….`
  - ‘hadoop fs’ operations are generally Unix-styled
  - `hadoop fs –put local_file hdfs_dir`

HDFS: Downsides, Issues and Things To Note

- Replication requires lots of space
  - Default: replicate 3 times => 200% overhead
  - Improved with Hadoop 3.0+
- Slow Performance
- Limits to file size
  - Improved using multiple name nodes with federated namespaces
  - Increase block size to cut down on number of blocks

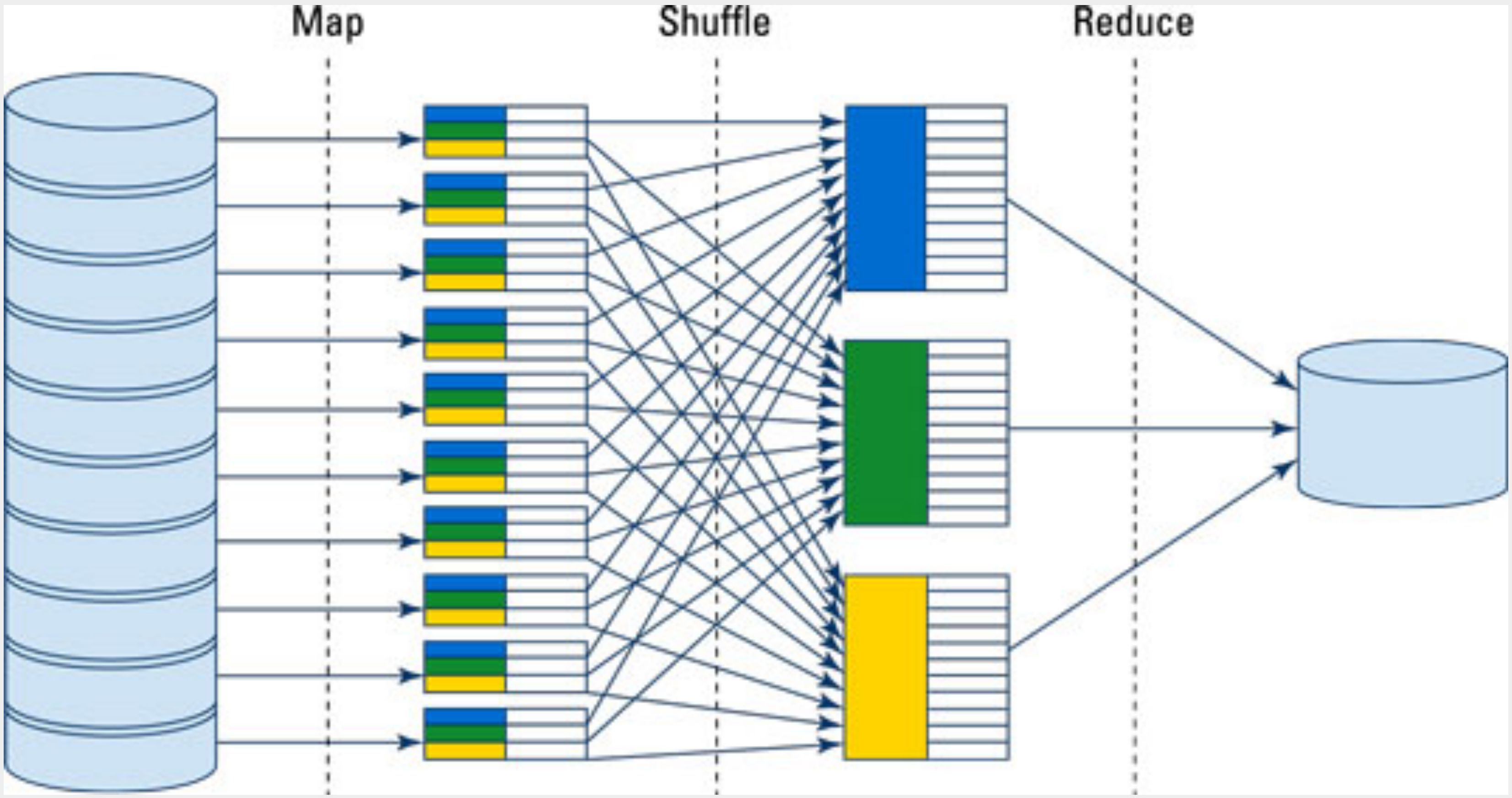
© Jason Kolter

Hadoop MapReduce

- Distributes data processing across a cluster of machines and handles:
  - Coordinating tasks
  - Splitting input data
  - Failure recovery
- Components
  - Resource Manager (YARN)
    - Multiple for HA
  - Node Manager(s)

© Jason Kolter

MapReduce



MapReduce: Example—Input Data

- Example:
  - Find average price per zip code from housing data
  - The data's in a CSV file that looks like…

```
id,date,price,bedrooms,bathrooms,sqft_living,sqft_lot,floors,waterfront,view,condition,grade,sqft
_above,sqft_basement,yr_built,yr_renovated,zipcode,lat,long,sqft_living15,sqft_lot15
 "7129300520","20141013T000000",221900,"3","1",1180,5650,"1",0,0,3,7,1180,0,1955,0,"98178",
47.5112,-122.257,1340,5650
```

MapReduce: Example—Mapping

- **Map:**
  - For each input line emit a key/value pair of zipcode/price
  - Example input line:

In: `"7129300520","20141013T000000",221900,"3","1", 1180,5650,"1",0,0,3,7,1180,0,1955,0,"98178", 47.5112,-122.257,1340,5650`

Out: (98178, 221900)

MapReduce: Example—Reducing

- Reduce:
  - For each key, compute average of list of prices
  - After the shuffle phase we end up with:

In: (98008, [450000,346000,799000,….])

Out: (98008, 526454)

© Jason Kolter

MapReduce Code: Old School Hideous Mapper

```
public class HomePriceZipCodeMapper extends
          Mapper<LongWritable, Text, IntWritable, IntWritable> {

    private final static IntWritable keyOut = new IntWritable();
    private final static IntWritable valueOut = new IntWritable();
    private final CSVParser parser = new
                                    CSVParserBuilder().withSeparator(',’)
                                         .withQuoteChar('"').build();

    @Override
    protected void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
       try {
          CSVReader reader = new CSVReaderBuilder(new StringReader(
                         value.toString())).withCSVParser(parser).build();
          String[] line = reader.readNext();
          int zipCode = Integer.parseInt(line[16]);
          int price = Integer.parseInt(line[2]);
          keyOut.set(zipCode);
          valueOut.set(price);
          context.write(keyOut, valueOut);
     } catch (Exception ex) {
          // just swallow error for demo
     }
   }
}
```

MapReduce Code: Old School Hideous Reducer

```
public class HomePriceZipCodeAverager extends Reducer<IntWritable, IntWritable,
                              IntWritable, DoubleWritable> {

   @Override
   protected void reduce(IntWritable key, Iterable<IntWritable> values,
   Context context) throws IOException, InterruptedException {

     int sum = 0;
     int count = 0;

     for (IntWritable value : values) {
          sum += value.get();
          count++;
     }
     double average = sum / (double) count;
     context.write(key, new DoubleWritable(average));
  }
}
```

MapReduce: Things to Note So Far

– Rigid, clumsy programming model
  - Key/value pairs
  - Types? Positions? Offsets? Oh my!

– Dependent on YARN and HDFS

– Multiple passes on same data require separate applications/code

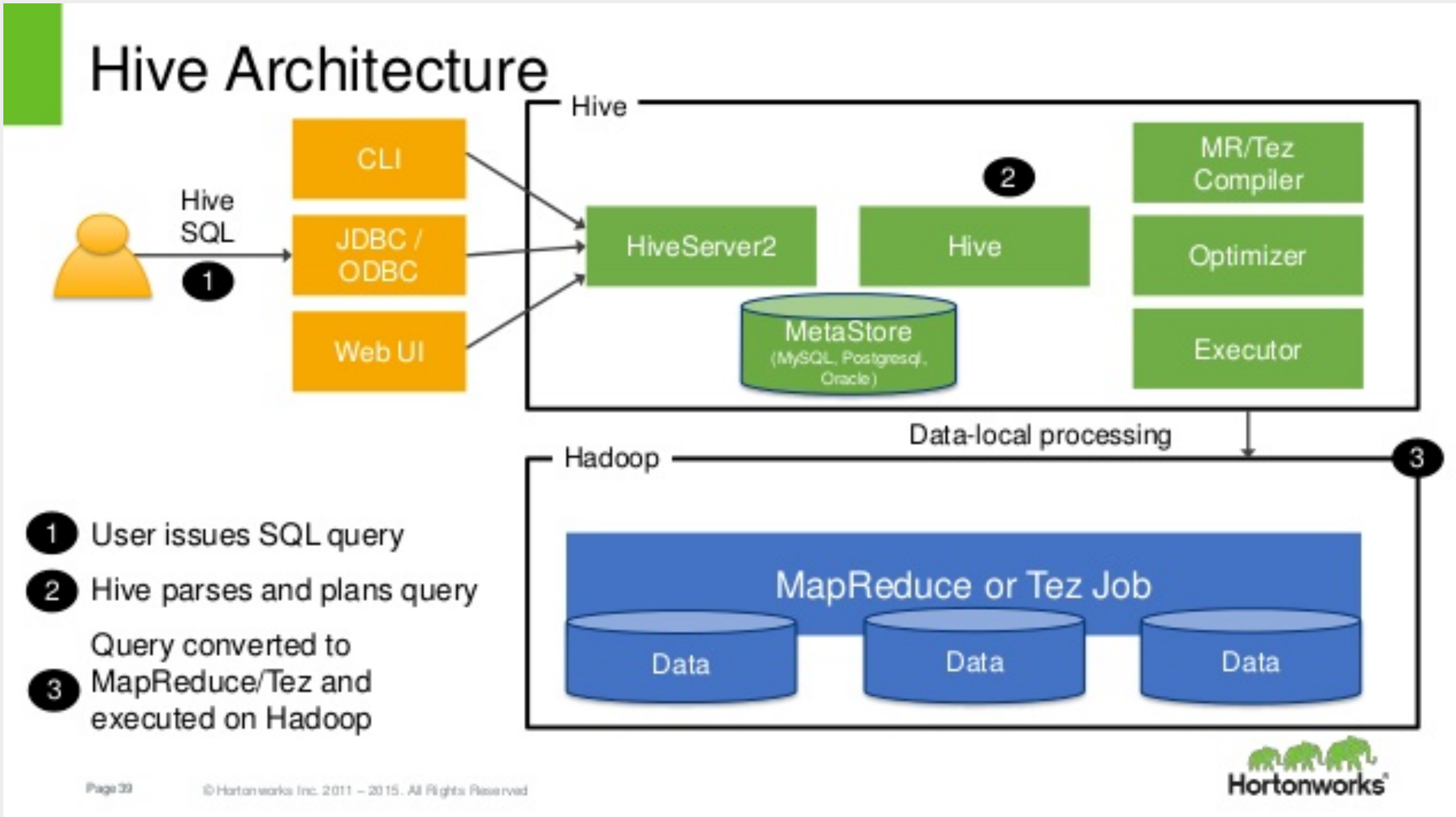
– Multiple passes may read/write a lot of data to and from HDFS

Hive: Making Hadoop More Humane

- RDBMSes show us *declarative* query languages:
  - Are productive
  - Are well tolerated and usable by people
- So Hadoop grew *Catalog metadata* to:
  - Add structure to data over data in HDFS
  - Support SQL style queries in Hadoop
- Enabling components ape some of the pieces of an RDBMS:
  - Metadata server
    - Proxies metadata requests to backing SQL storage
  - Hive server
    - Handle query parsing/planning/optimizing and send to execution engine
  - Execution Engine(s)
    - MR
    - Tez
    - Spark
- How do these components fit together to help us?

© Jason Kolter

Hive



Hive Architecture
Hive
Hive
SQL
1
CLI
JDBC /
ODBC
Web UI
HiveServer2
2
Hive
MetaStore
(MySQL, Postgresql,
Oracle)
MR/Tez
Compiler
Optimizer
Executor
Data-local processing
Hadoop
3
MapReduce or Tez Job
Data
Data
Data
1 User issues SQL query
2 Hive parses and plans query
3 Query converted to
MapReduce/Tez and
executed on Hadoop
Page 33
© Hortonworks Inc. 2011 – 2015. All Rights Reserved
Hortonworks

Hive:
Data Made to Look More SQL Table-is

- Example:
  - Find average price per zip code from housing data

ID	Date	price	bedrooms	bathrooms	floors	view	Zipcode	lat	lon
1231	2015121	444000	3	3	2	Y	98008	40.22	-80.12

Hive:
A SQL-ish Query Language

- HiveQL: looks mostly like SQL…
- Compared to those MapReduce Java classes!

```
select zipcode, avg(price) from home_data group by zipcode;
```

© Jason Kolter

Hive:
Ways to Access It

- Hive CRUD operations and queries can be done from:
  - Command line
    - NOTE: Original 'hive' command line deprecated..
    - Now: Use the 'beeline' command.
  - Java or any JDBC/ODBC application
    - Via JDBC Java code can talk to Hive like it would another SQL DB
    - Excel and many BI tools can speak ODBC
  - Spark (we'll see a lot later)

© Jason Kolter

Hive:
Example of Table Creation

```
• CREATE TABLE `home_data`(
•   `id` bigint,
•   `date` string,
•   `price` int,
•   `bedrooms` int,
•   `bathrooms` double,
•   `sqft_living` int,
•   `sqft_lot` int,
•   `floors` int,
•   `waterfront` int,
•   `view` int,
•   `condition` int,
•   `grade` int,
•   `sqft_above` int,
•   `sqft_basement` int,
•   `yr_built` int,
•   `yr_renovated` int,
•   `zipcode` int,
•   `lat` double,
•   `long` double,
•   `sqft_living15` int,
•   `sqft_lot15` int)
• ROW FORMAT SERDE
•   'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
• STORED AS INPUTFORMAT
•   'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
• OUTPUTFORMAT
•   'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
• LOCATION
•   'hdfs://sandbox.hortonworks.com:8020/apps/hive/warehouse/home_data'
```

Hive:
How does it all work underneath?

- We saw the components and how they ape RDBMS structure earlier
- Two Table options:
  - **Internal**
    - Completely managed by Hive
    - Data is copied into HDFS warehouse dir
    - Data is DELETED from HDFS when table dropped
  - **External**
    - Only metadata is managed by Hive
    - Data not copied from source
    - Options besides HDFS
    - Data not deleted when table is dropped

© Jason Kolter

Hive:
How does it work underneath?

- File Formats—what does the stuff look like on HDFS storage?
  - ORC File
    - The “Hive” file format
    - Efficient column projection
  - AVRO
    - Portable container file format
    - Good for cross platform functionality
  - Other Hadoop InputFormats
    - Legacy

© Jason Kolter

Hive:
How does it work underneath?

- Partitioned table
  - Physically separates the data on HDFS for more efficient queries
  - Example: Frequent queries by date and zipcode
    - Create partitioned hive table by date then by zipcode
    - Creates physical subdirectories for those slices of data
    - Don't over-partition
      - Small partitions and very high cardinality data will become less efficient

© Jason Kolter

Week 2: Summary

- A motivating example
- Bigness => distribution
- Distribution => challenges
- Hadoop ecosystem pillars:
    - HDFS
    - MapReduce
- MapReduce and its discontents
- A first step forward: Hive

Assignment 2

- Assignment to be posted to Canvas tomorrow
  - Log into your *Azure VM*
  - See *Hortonworks sandbox* running; explore!
  - Get sample data from course website into your sandbox
  - *Hive warmup exercise* (actual data munging part should be easy for SQL sophisticates)
- *Try to start early!* The first week out is where we may encounter hiccups.

© Jason Kolter