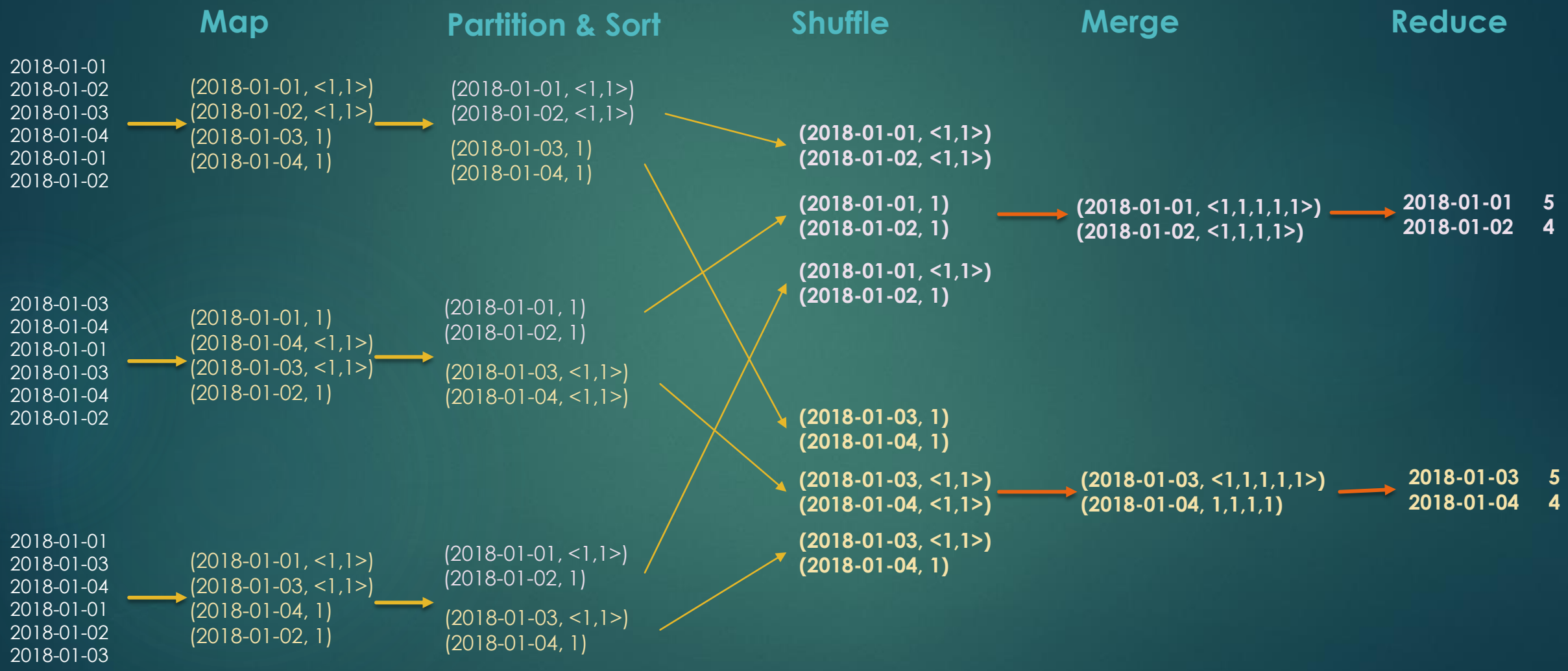
Recap

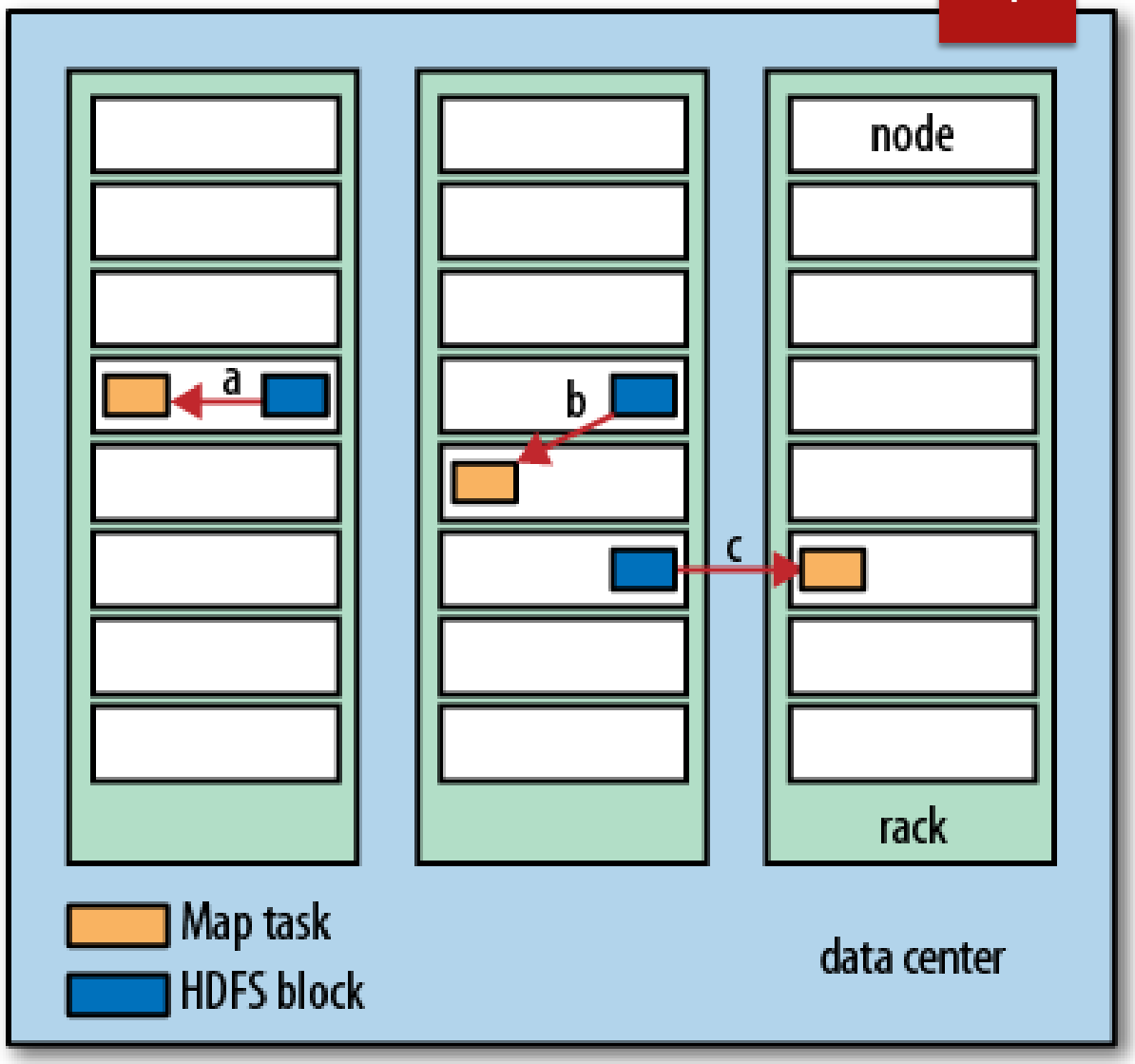
- MapReduce
- Hadoop 1 vs Hadoop 2
- YARN

Mapreduce: working example



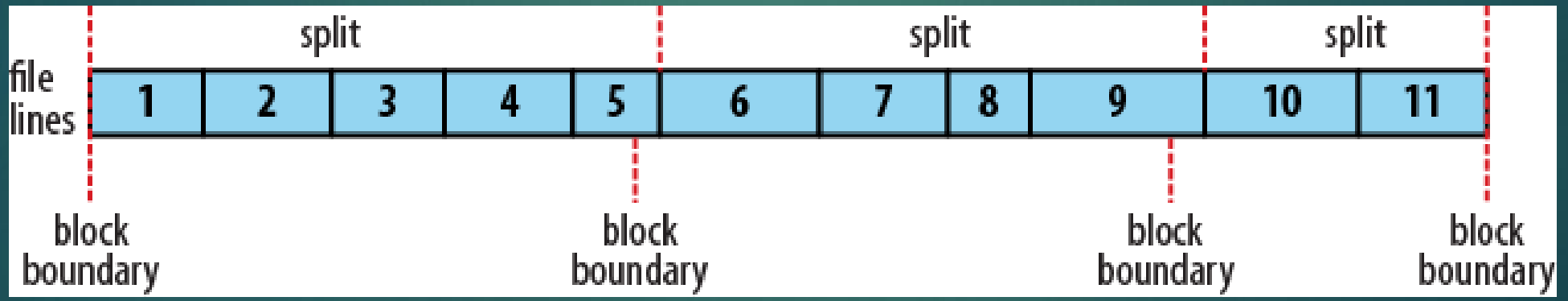
Task to node mapping

- Notion of data locality



Input Splits

- Blocks are of fixed size
- Good chances of records being split between two blocks



MapReduce: Mapper code


```
public class WebHitCounterMapper extends
Mapper<Input Key, Input Value, Output Key, Output Value>
{

    public void map(Input Key, Input Value, Context context)
throws IOException, InterruptedException {

     <MAP Logic goes here>

     context.write(Output Key, Output Value)
    }
}
```

MapReduce: Reducer code

```
public class WebHitCounterReducer extends
Reducer<Input Key, Input Value, Output Key, Output Value>
{
    public void reduce(Input Key, Iterable<Value Data type>
values,Context context) throws IOException,
InterruptedException {

     <REDUCE logic goes here>

        context.write(Output Key, Output Value);
    }
}
```

MapReduce: Driver Code


```
public class WebHitCounterMain {
    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Daily Web Hit Counter");

        job.setJarByClass(main.WebHitCounterMain.class);
        job.setMapperClass(mapper.WebHitCounterMapper.class);
        job.setReducerClass(reducer.WebHitCounterReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Performance tuning

- Cluster configuration
- Use compression technique
- Tuning # mappers and reducers
- Use combiner
- Appropriate data type
- Reuse objects
- Profiling

MapReduce Job chaining

- Two separate jobs
- Multiple mappers/reducers within same job

Hadoop 2

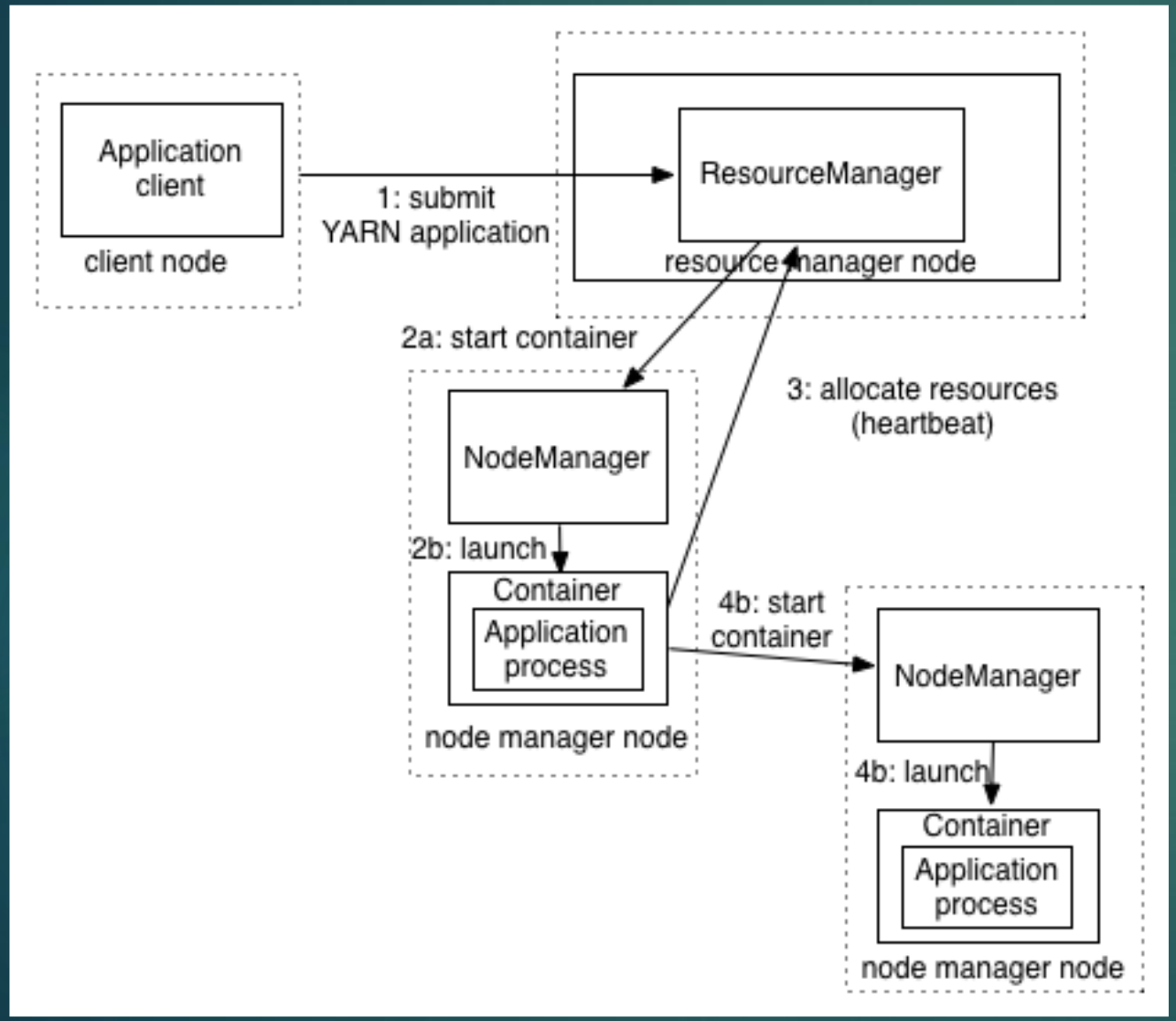
- Support for other data processing engines
- High Availability
- HDFS Federation
- HDFS Snapshot
- Introduced Streaming and Interactive analysis tools
- Support for various file formats
- Yarn

YARN

- Yet Another Resource Negotiator

MapReduce 1	YARN
Job Tracker	Resource Manager, Application Master and Timeline server
Task Tracker	Node Manager
Slot	Containers

YARN model


Application client
client node
1: submit YARN application
ResourceManager
resource manager node
2a: start container
3: allocate resources (heartbeat)
NodeManager
2b: launch
Container
Application process
node manager node
4b: start container
NodeManager
4b: launch
Container
Application process
node manager node

Agenda for today

- Installing Hadoop on single node
- Pig: Mapreduce scripting
- Routine SQL operations

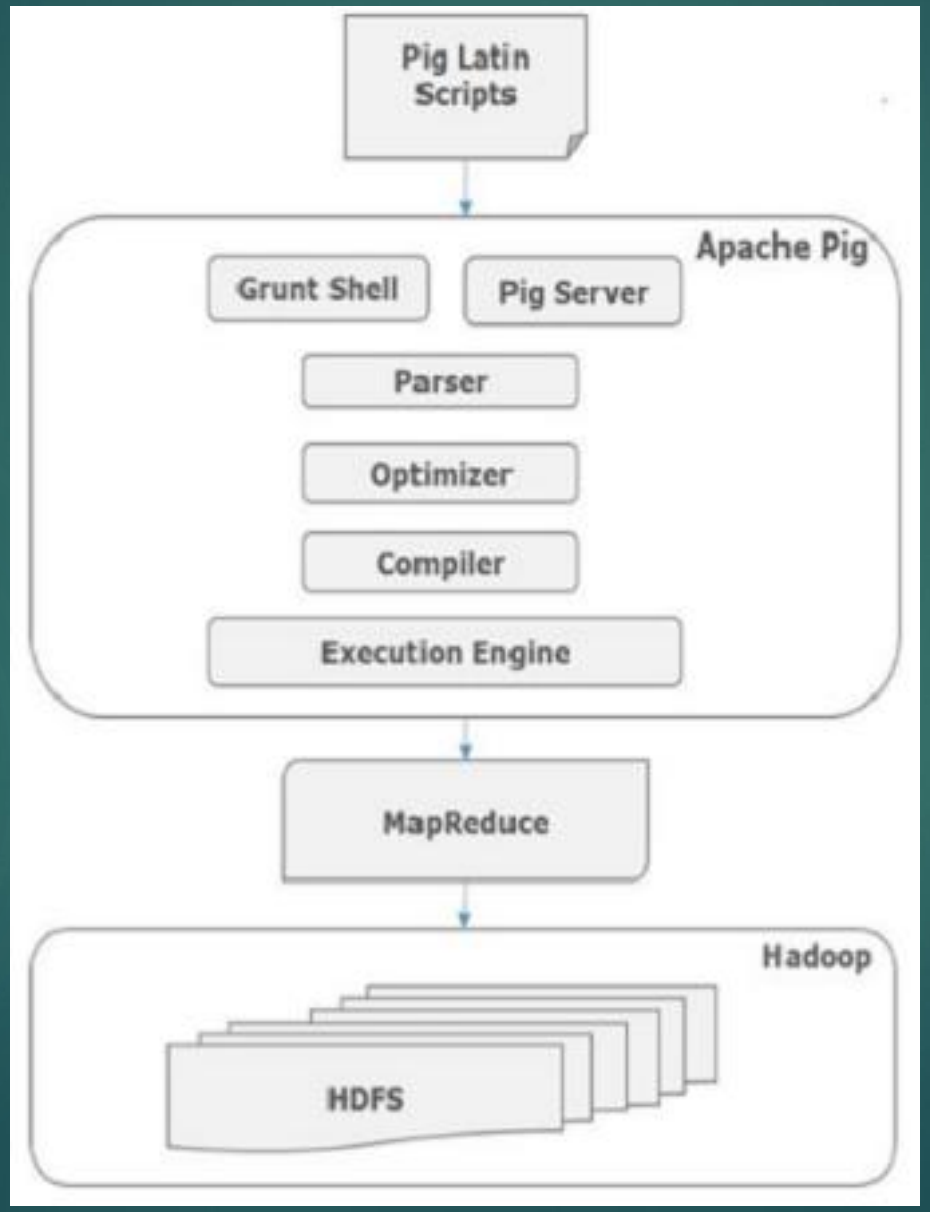
Hadoop Installation

Standalone	Everything in one JVM. No HDFS installation
Pseudo-distributed	Mimic a distributed cluster on single physical machine
Distributed	Fully distributed cluster with multiple physical machines

Introduction

- High Level Scripting Language developed by Yahoo originally
- Transforms SQL like language called Pig Latin into Java code
- Follows lazy evaluation
- Supports UDF written in multiple languages

Architecture



Execution

- Accessing approaches:

1. Batch mode: Run a script file
2. Interactive mode: Grunt, the pig shell
3. PigServer for Java program

- Execution mode:

1. Local mode: pig –x local
2. Mapreduce mode (default): pig –x mapreduce

Data types

- Scalar Types: Int, long, float, double, boolean, null, chararray, bytearray
- Complex Types: fields, tuples, bags, relations

Various loaders

- Supports various loader formats

1. TextLoader
2. PigStorage
3. JsonLoader & JsonStorage
4. BinStorage
5. HBaseStorage
6. OrcStorage
7. MongoStorage

Operator: LOAD

- To load data from storage system

lines=LOAD ‘myfile’ AS (line: chararray);

books = LOAD ‘/data/pig/books.csv’ as (line: chararray)

Operator: DUMP

- Print the data on console

DUMP RelationName;

DUMP sample_books;

Operator: LOAD cont…

- Load data without schema

relXYZ = LOAD ‘yourfile.csv' USING PigStorage(',');

books = LOAD ‘/data/pig/books.csv' USING PigStorage(',');

- Load data with schema

relXYZ = LOAD ‘yourfile.csv' USING PigStorage(',') as (col1:datatype, col2:datatype,…);

books = LOAD ‘/data/pig/books.csv' USING PigStorage(',') as (id:int, author:chararray, name:chararray, year:int);

Operator: LIMIT

- Take sample records

New_Rel = LIMIT RelationName <Sample Count>;

Sample_books = LIMIT books 5;

Operator: FOREACH

- Select specific columns

New_Rel = FOREACH RelationName GENERATE col1, col2, col3….;

book_no_author = FOREACH books GENERATE id, name, year;

Operator: JOIN

- Joins two relations/datasets

```
join_data = JOIN  relation1 BY (column1), relation2
BY (column1);
```

```
book_review = JOIN books BY (id), reviews BY (id)
```

Operator: SORT

- Sort a relation based on key

New_rel = ORDER RelationName BY ColumnName asc;

books_sorted_by_year = ORDER books BY year asc;

Operator: FILTER

- Filter the dataset

New_rel = FILTER RelationName BY (Condition);

books_before_2000 = FILTER books BY (year < 2000)

Operator: DISTINCT

- Remove duplicates

New_rel = DISTINCT RelationName;

bedupe = DISTINCT books_before_2000;

Aggregate

- Aggregate based on a key

GroupRel = GROUP RelName BY columnName;

```
group_review = group book_review by books::id;
```

AggRel = FOREACH GroupRel GENERATE group , AVG(columnName)

```
avg_rating = foreach group_review generate group as id,
AVG($1.reviews::rating)
```

Operator: STORE

- Store the output

STORE relationName INTO 'output_directory' USING PigStorage(',');

STORE dedupe INTO '/data/pig/dedupe' USING PigStorage(',');

PigServer API


```
import java.io.IOException;
import org.apache.pig.PigServer;
public class idlocal{
public static void main(String[] args) {
        try {
            PigServer pigServer = new PigServer("local");
            runIdQuery(pigServer, "passwd");
            }
            catch(Exception e) {}
            }
        public static void runIdQuery(PigServer pigServer, String inputFile) throws IOException {
            pigServer.registerQuery("A = load '" + inputFile + "' using PigStorage(':');");
            pigServer.registerQuery("B = foreach A generate $0 as id;");
            pigServer.store("B", "id.out");
 }}
```

UDF

- Prepare a Jar file
- Register the Jar
- Define alias
- Use it

Reference

- Hadoop standalone vs pseudo-distributed

https://stackoverflow.com/questions/23435333/what-is-the-difference-between-single-node-pseudo-distributed-mode-in-hadoop

- Hadoop installation differences

https://medium.com/@nidhinmahesh/getting-started-hadoop-mapreduce-hdfs-and-yarn-configuration-and-sample-program-febb1415f945

- Download Ubuntu

https://www.ubuntu.com/download/desktop

- Hadoop installation step by step guide

http://www.bogotobogo.com/Hadoop/BigData_hadoop_Install_on_ubuntu_16_04_single_node_cluster.php

- Map Reduce job chaining:

https://mapr.com/blog/how-to-launching-mapreduce-jobs/

- Pig inbuilt functions

https://pig.apache.org/docs/latest/func.html

- Mapreduce job optimization

https://blog.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/

- Pig UDF

https://www.tutorialspoint.com/apache_pig/apache_pig_user_defined_functions.htm