BUSS6002 Data Science in Business

Week 9: Computing and Programming Techniques for Big Data - II
MapReduce and Hadoop

Dr. Jie Yin
Discipline of Business Analytics
University of Sydney Business School

Learning Objectives

- In Week 7, we learnt how to select the optimal model based on model evaluation and selection techniques.
- In Week 8, we learnt statistical and computational techniques for dealing with Big Data.
- This week we will focus on MapReduce – an off-the-shelf programming technique for dealing with Big Data, and then present examples of MapReduce.
- Then we will look at the most popular way of using these tools, known as Hadoop.
- You will learn essentials about the Core Hadoop
  - Hadoop Distributed File System (HDFS)
  - Yet Another Resource Negotiator (YARN)
  - MapReduce/Spark
- You will learn about the Hadoop Ecosystem

MapReduce for Big Data

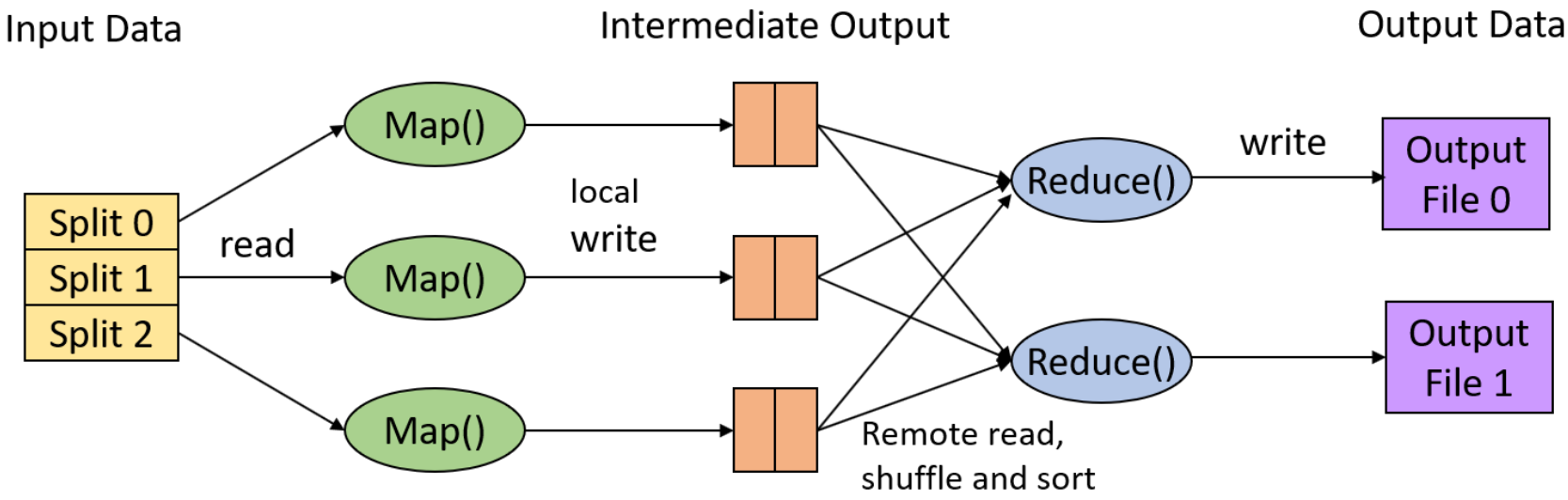
What is MapReduce?

- MapReduce
  - Invented by Jeffrey Dean and Sanjay Ghemawat from Google in 2004.
  - A programming technique for dealing with large datasets that do not fit in the computer memory.
  - Allows to process huge amounts of data (terabytes and petabytes) by connecting many commodity computers to work in parallel.
- Key idea: Divide the dataset into smaller subsets (chunks) so that each chunk can be loaded into the computer memory. Each chunk is loaded into RAM and analysed separately. The results are then combined to form the final result.
- MapReduce fits perfectly well to the divide-and-conquer and subsampling approaches as they both analyse data subsets separately.

Typical problems solved by MapReduce

- Read a lot of data splits (chunks).
- Map: extract something you care about from each chunk.
- Shuffle and sort.
- Reduce: aggregate, summarise, filter, or transform.
- Write the final results.

MapReduce workflow



Map

extract something you care about from each data split

Reduce

aggregate, summarize, filter, or transform

Mappers and Reducers

- In MapReduce, chunks are processed in isolation by tasks called Mappers.
- The outputs from the mappers are denoted as intermediate outputs and are fed into a second set of tasks called Reducer.
- The process of bringing together intermediate outputs into a set of Reducers is known as shuffling process.
- The Reducers produce the final outputs.
- Overall, MapReduce workflow can be broken into two phases: map phase and reduce phase.

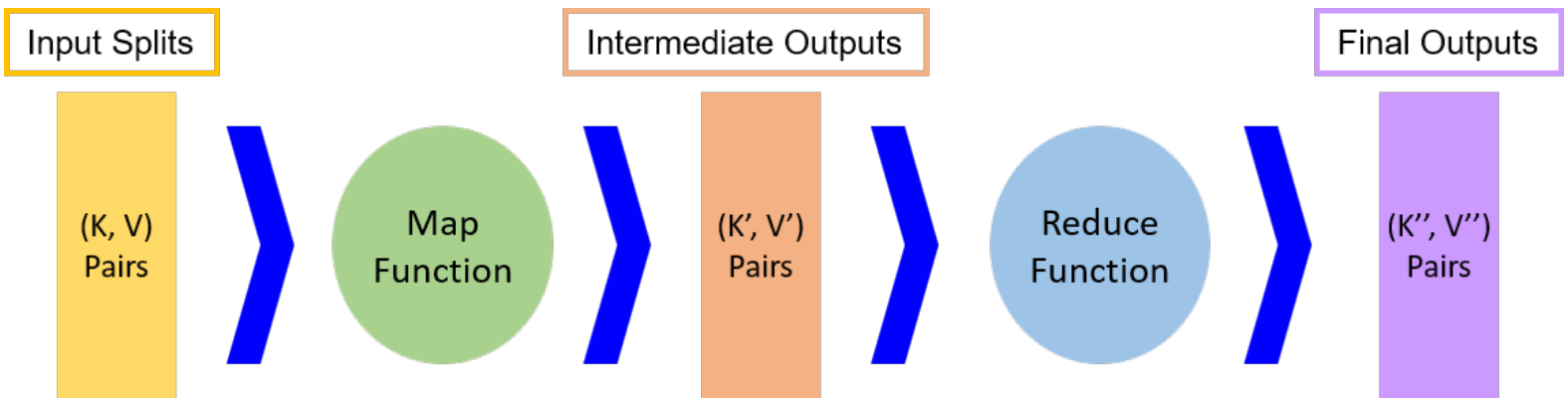
Data execution phases

In MapReduce, the data goes through the following four phases of execution:

- Input Splits: An input to a MapReduce job is divided into fixed-size chunks called input splits.
- Mapping: Data in each input split is passed to a mapping function to produce intermediate outputs.
- Shuffling: This phase consolidates the relevant records from intermediate outputs.
- Reducing: Outputs from the shuffling phase are aggregated to return a single output value.

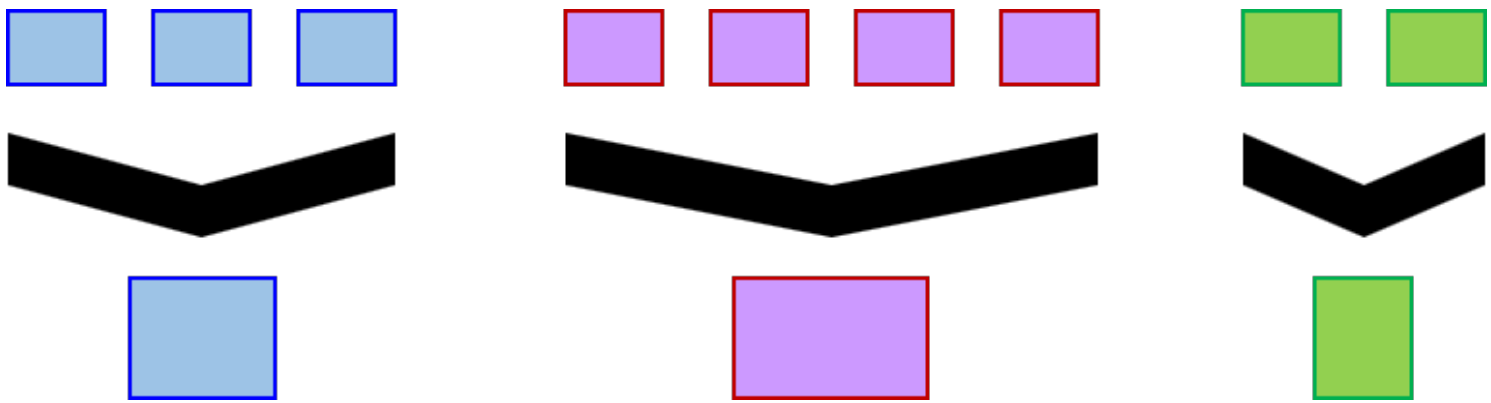
Keys and Values

- The programmer in MapReduce needs to specify two functions, map function and reduce function that implement the Mapper and the Reducer in a MapReduce program.
- In MapReduce, data elements are always structured as key-value pairs, i.e., (K,V) pairs.
- Map and reduce functions receive and emit (K,V) pairs.



Partitions

- In MapReduce, intermediate output values are not usually reduced together.
- All values with the same key are passed to a single Reducer all together.
- More specifically, a different subset of intermediate key space is assigned to each Reducer.
- These subsets are called partitions.



Different colors represent different keys (potentially) from different Mappers and partitions are the input to Reducers.

Example 1: word count with MapReduce

Let us consider an example of counting word frequency in a corpus.

Suppose that you are given a large corpus of text

```
Welcome to Hadoop
Class Hadoop is
good Hadoop is bad
```

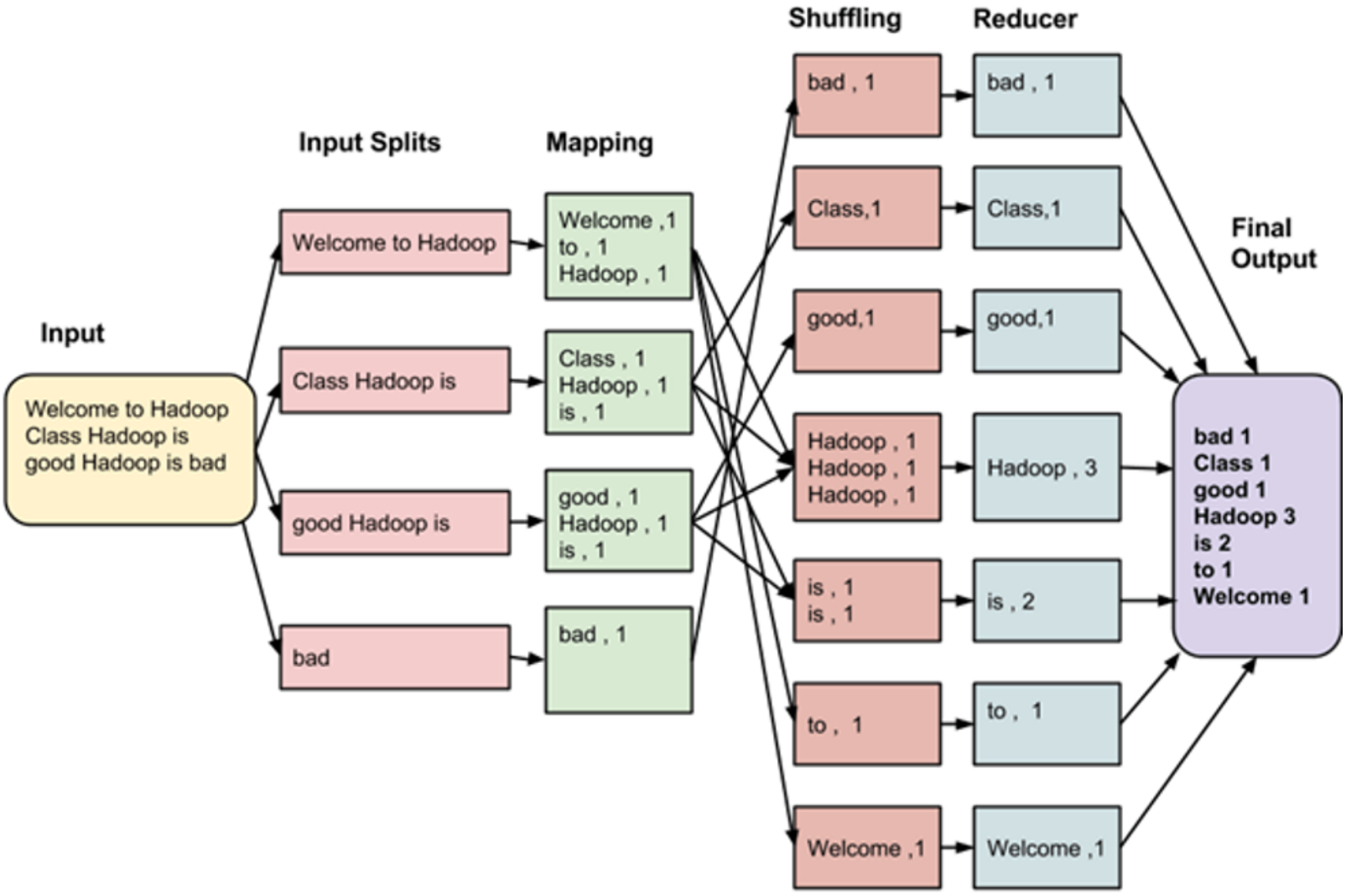
This is not a large corpus but we use it to illustrate the idea.

Example 1: word count with MapReduce

We need to define the map and reduce functions.

```
# Import necessary modules
import re
from mockr import run_stream_job

# This regular expression matches words
WORD_RE = re.compile(r"[\w']+")

def map_fn(chunk):
    # Use the regex to find all words in each chunk, The chunk is a line of text because we are
    # using run_stream_job
    for word in WORD_RE.findall(chunk):
        # Emit a result using the word as the key and the number of times it occured as the value.
        # We emit once for each word so this value is 1.
        yield (word.lower(), 1)

def reduce_fn(key, values):
    # Recieves all the values for each key (unique word) then sums them together for the total count
    yield (key, sum(values))

# run_stream_job expects an input string, map function and reduce function and
# returns a list of results
input_file = open("Hadoop.txt", 'r')
input_str = input_file.read()
results = run_stream_job(input_str, map_fn, reduce_fn)
```

MapReduce architecture for word count



Source: `https://www.guru99.com/`

Linear regression with MapReduce

Consider the linear regression model

$$\boldsymbol{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{i1} & \cdots & x_{ip} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{bmatrix}, \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_i \\ \vdots \\ \epsilon_n \end{bmatrix}.$$

The solution is

$$\widehat{\boldsymbol{\beta}}_{ls} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y},$$

Computing this is infeasible in Big Data as $\boldsymbol{X}$ is a huge matrix.

Linear regression with MapReduce

Let's divide $\boldsymbol{X}$ and $\boldsymbol{y}$ into blocks

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_1 \\ \vdots \\ \boldsymbol{X}_k \\ \vdots \\ \boldsymbol{X}_K \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_k \\ \vdots \\ \boldsymbol{y}_K \end{bmatrix}.$$

Then

$$\widehat{\beta}_{ls} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y} = \left(\sum_{k=1}^{K} \boldsymbol{X}_k^T \boldsymbol{X}_k \right)^{-1} \left(\sum_{k=1}^{K} \boldsymbol{X}_k^T \boldsymbol{y}_k \right)$$

Each pair $\boldsymbol{X}_k^T \boldsymbol{X}_k$ and $\boldsymbol{X}_k^T \boldsymbol{y}_k$ can be computed for each data chunk separately.

Example 2: linear regression with MapReduce

Consider the dataset "BatonRouge.xls" for house price prediction, we need to define the map and reduce functions.

```
def map_linear_fn(chunk):
# Get the dependant variable y
y = chunk['Price'].values

# Get the independant/feature variables
X_vals = chunk[chunk.columns.difference(['Price'])].values

# Get the number of data points
m = chunk.shape[0]

# Insert a column of "1"s for the intercept term
X = np.column_stack((np.ones(m), X_vals))

# Convert to matrix to make multiplication easier
X = np.asmatrix(X)

# Calculate required multiplications
XtX = X.T*X
Xty = X.T * y.reshape(m,1)

# Yield the result
yield("result", [XtX, Xty])
```

Example 2: linear regression with MapReduce

```
def reduce_linear_fn(key, values):

# Create lists to accumulate the matrices/vectors in
XtX_list = []
Xty_list = []

for result_list in values:
    XtX_list.append(result_list[0])
    Xty_list.append(result_list[1])

# Sum up all the XtX matrices
XtX = np.asmatrix(sum(XtX_list))

# Sum up all the Xty vectors
Xty = sum(Xty_list)

# Solve the linear regression objective
betas = np.linalg.inv(XtX) * Xty

yield (key, betas)
```

Example 2: linear regression with MapReduce

```
from mockr import run_pandas_job

df = pd.read_excel("BatonRouge.xls")

results = run_pandas_job(df, map_linear_fn, reduce_linear_fn, n_chunks = 4)

# Simplify the result to make it more readable
params_mr = np.array(results[0][1]).ravel()

print(params_mr)
```

Estimated parameters from the Divide-and-Conquer approach:

```
[-4.91975489e+04 -4.40598830e+02  3.98305235e+04 -2.56557205e+04
-2.13140882e+01 -2.94581106e+03  7.81085610e+03 -1.63093297e+03
8.54451540e+01  1.06961744e+03  5.62042730e+04]
```

Example 2: linear regression with full data set

```
from sklearn.linear_model import LinearRegression

lr_obj = LinearRegression()

features = df[df.columns.difference(['Price'])]
target = df['Price']

lr_obj.fit(features, target)

params_sk = np.append(np.array(lr_obj.intercept_), lr_obj.coef_)

print(params_sk)
```

Estimated parameters from the Divide-and-Conquer approach:

```
[-4.91975489e+04 -4.40598830e+02  3.98305235e+04 -2.56557205e+04
-2.13140882e+01 -2.94581106e+03  7.81085610e+03 -1.63093297e+03
8.54451540e+01  1.06961744e+03  5.62042730e+04]
```

Recall from Week 8 that, the divide-and-conquer solution for linear regression is exactly equal to the full-data solution.

Logistic regression for big data

Now look at a real big dataset:
`http://stat-computing.org/dataexpo/2009/the-data.html`

All available data on all US airline flights for over a decade (departures, arrivals, destrinations, delays, etc.)

Details at:
`https://www.transtats.bts.gov/Fields.asp?Table_ID=236`

Volume: hundreds of GB

Goal: a predictive model for the probability of a > 20 minutes delay as a function of flight distance

Logistic regression with MapReduce

- A small subset `Airline.csv` is processed as in Canvas with target variable `Delayed` (flight arrival delay more than 20 minutes (1) or not (0)) and predictor `Distance` (total flight distance in thousands of miles)
- Let's solve this classification problem with logistic regression

$$P(\texttt{Delayed}|\texttt{Distance}) = \frac{\exp(\beta_0 + \beta_1 \times \texttt{Distance})}{1 + \exp(\beta_0 + \beta_1 \times \texttt{Distance})}$$

- How to estimate the coefficient vector $\widehat{\boldsymbol{\beta}}$ with MapReduce?

Example 3: logistic regression with MapReduce

Basic idea:

- Generate multiple random subsample sets of the original data.
- Each mapper receives a random subsample set and fit a logistic regression to estimate the parameters.
- The reducer simply takes the average of the estimated parameters from each subset to form the final solution.

Try to implement the MapReduce for logistic regression by yourself, based on linear regression MapReduce example.

Example 3: logistic regression with MapReduce

```
from mockr import run_pandas_job

# Load the data set
df= pd.read_csv("airline.csv")

# Generate random subsets of the data
sub_sample_size = 5000

# Set number of replications
n_replications = 5

subsamples = []
for k in range(n_replications):
    # Generate a random subset of size "sub_sample_size"
    subsamples.append(data.sample(n = sub_sample_size))

samples = pd.concat(subsamples, axis = 0)

results = run_pandas_job(samples, mapper, reducer, n_chunks = n_replications)

print(results)
```

Estimated parameters from subsampling approach:

```
[-1.67324077,  0.1374882]
```

Example 3: logistic regression with full data set

```
from sklearn.linear_model import LogisticRegression

df= pd.read_csv("airline.csv")
x = df['Distance'].values.reshape(-1,1)
y = df['Delayed']

log_reg = LogisticRegression(C = 1e32)
log_reg.fit(x, y)

# Output the estimated coefficients
params = np.append(np.array(log_reg.intercept_), log_reg.coef_)

print(params)
```

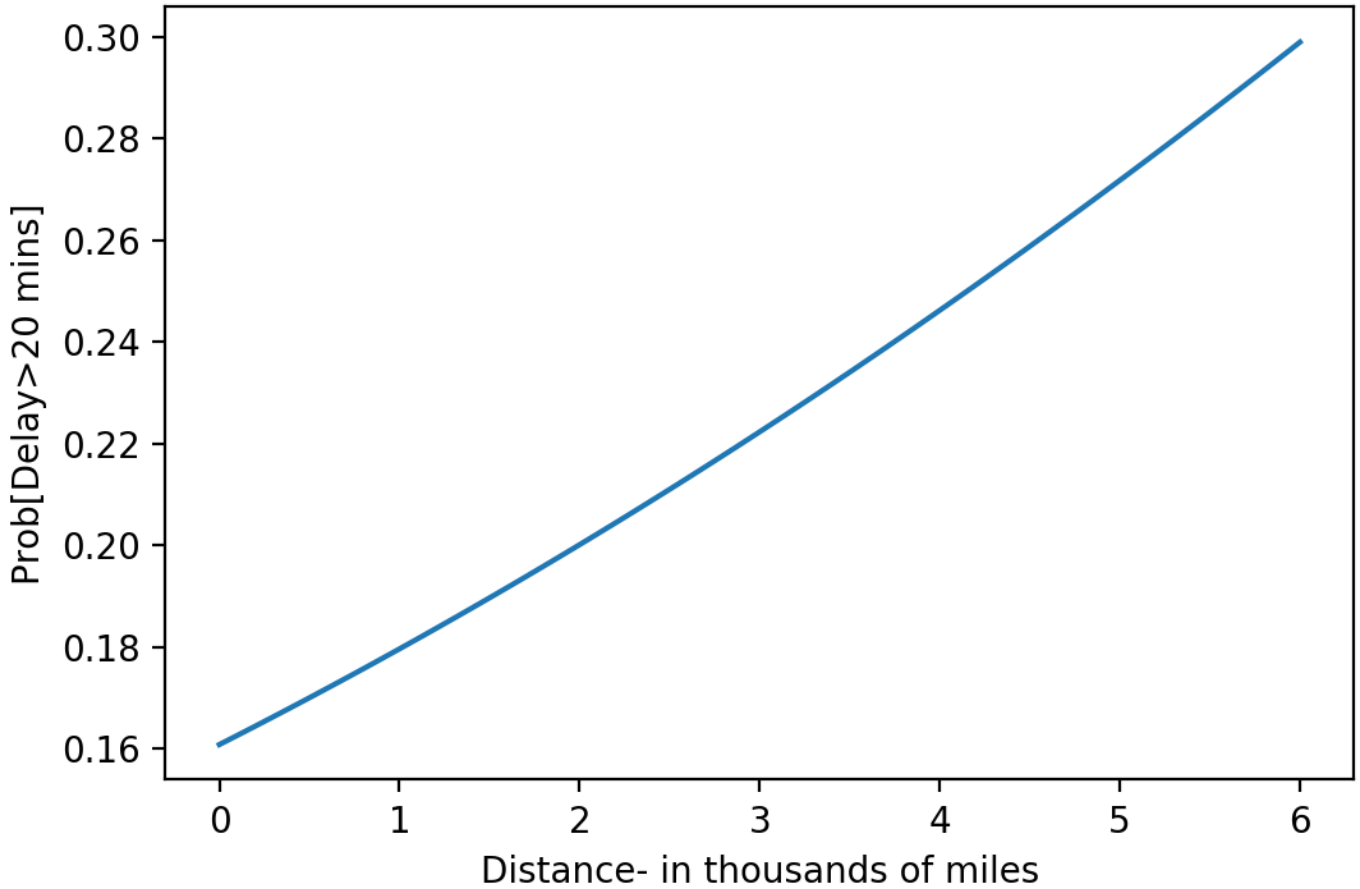
Estimated parameters from full data set:

```
[-1.67053828  0.14150803]
```

As discussed in Week 8, the subsampling estimated parameters are very close to the full data estimated parameters, but the subsampling can be much faster than using the full data. The difference is due to randomness, mathematically guaranteed to go to zero when the number of replications increases.

Example 3: logistic regression with MapReduce

Predictive model for flight delay with parameters estimated from sub-sampling approach



What makes MapReduce unique?

- Allows the user to quickly write and test.
- Works in all environments, from a laptop to Hadoop clusters, with minimal code change.
- Handles efficient and automatic distribution of data and workload across machines.
- Permits major efficiency gains due to parallelisation.
- Enables a flat scalability curve. E.g., after a MapReduce program is written and functioning on 10 nodes, very little work is required for making the same program run on 1,000 nodes.

Hadoop Introduction

What is Hadoop?

- An open-source implementation of MapReduce for storing and analysing Big Data.
  - distributed
  - scalable
  - fault-tolerant
- Developed by Apache Software Foundation, a decentralised community of developers.

Short History of Hadoop

- Ghemawat, S.; Gobioff, H.; Leung, S. T. (2003). "The Google file system". *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles - SOSP '03.* `http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf`
- 2004: MapReduce
- 2006: Hadoop 0.1.0 released (named after Doug Cutting's son's yellow plush toy)
- 2009: Yahoo runs Hadoop on 17 clusters with 24,000 machines
- 2010: Facebook runs 2,300 clusters/40 petabytes (1PB = 1 ml GB)
- 2010: Apache created
- 2013: Hadoop Summit 3000+ people
- 2017: Apache Hadoop 2.8

Current state

- 90 company-committers with 20 core committers: Cloudera, Yahoo!, LinkedIn, Intel, Facebook, etc
- Contributers: anybody
- Hadoop Ecosystem: Hadoop core and Hadoop-related tools and projects

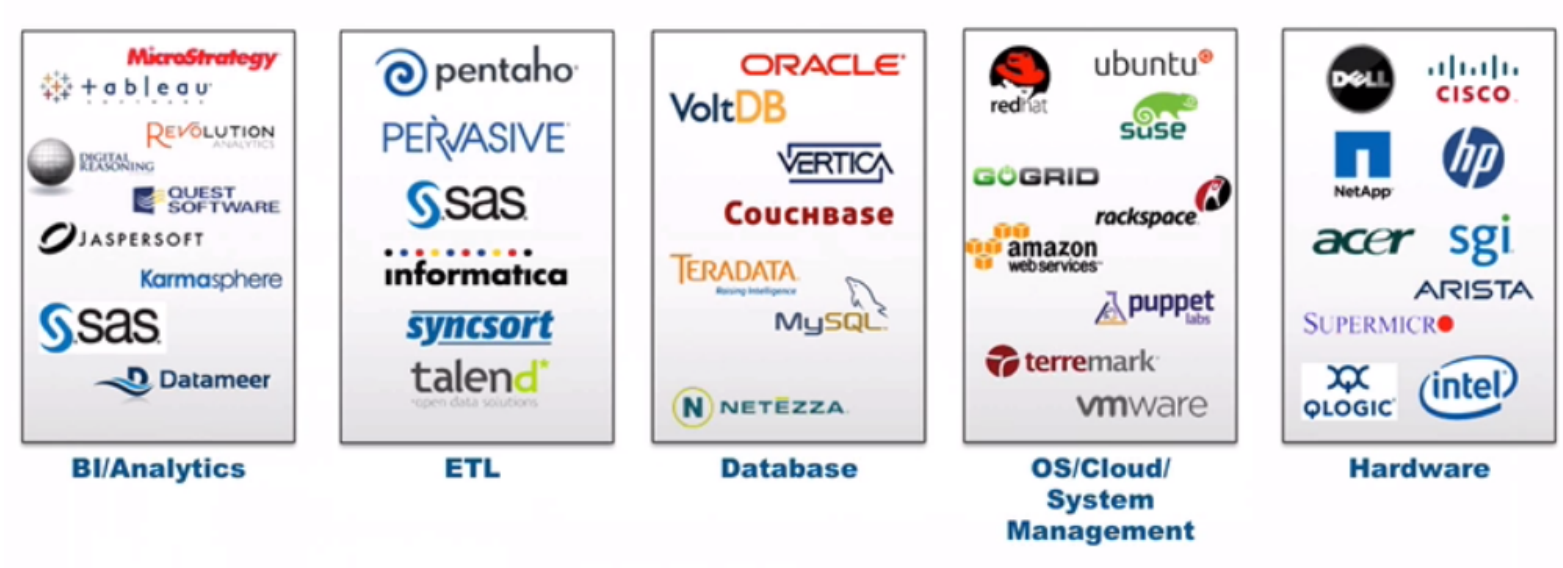
Hadoop Ecosystem

Hadoop Ecosystem

Project	What does it do?
Spark	In-memory and streaming processing framework
HBase	NoSQL database built on HDFS
Hive	SQL processing engine designed for batch workloads
Impala	SQL query engine designed for BI workloads
Parquet	Very efficient columnar data storage format
Sqoop	Data movement to/from RDBMSs
Flume, Kafka	Streaming data ingestion
Solr	Powerful text search functionality
Hue	Web-based user interface for Hadoop
Sentry	Authorization tool, providing security for Hadoop

Source: Cloudera

Commercial products Hadoop works with



Source: Cloudera

Hadoop users

Source: Cloudera

Examples of Hadoop use

- Financial Sector:
  - JP Morgan Stanley: fraud detection, money laundering
  - Mastercard: credit card default, fraud detection
- Insurance:
  - Allstate: premium pricing, fraud detection
  - Marketstudy: price discrimination, market segmentation
- Telecom:
  - Nokia: traffic modelling, user experience modelling
  - Telkomsel: data mining

Many stop throwing away the data as Hadoop solutions became available

Core Hadoop

Recall: Divide-and-conquer and subsampling methods use chunks of the original data

Hadoop makes use of this approach

- Distribute the data into chunks when it is loaded into the system
- Run computations where the data is stored
- Capacity is added by adding chunks (computers) to a cluster, not by increasing each computer's capacity (scaling out vs scaling up)

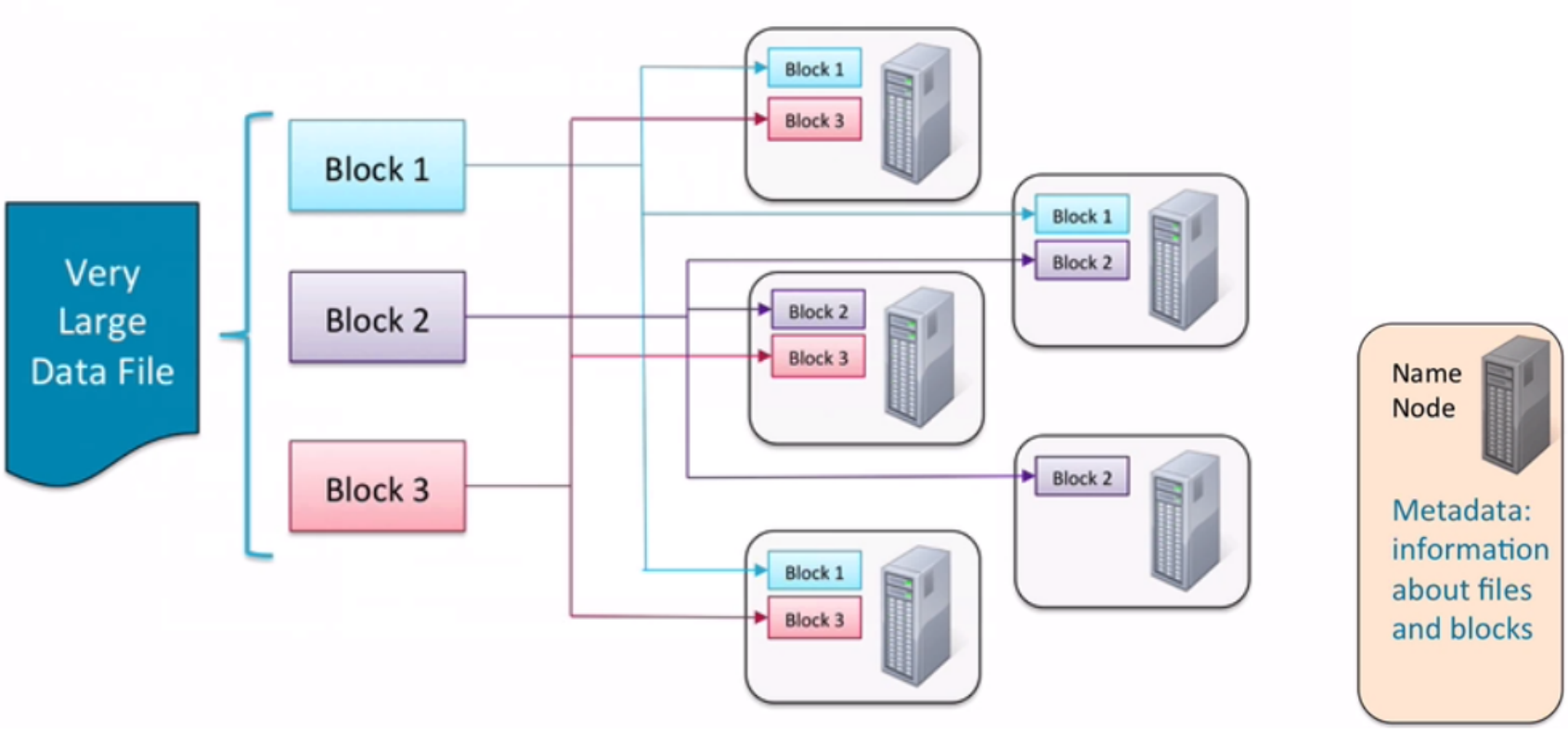
Core Hadoop includes:

- HDFS: stores chunks of data on a cluster (of computers)
- MapReduce: analyses chunks of data on the cluster
- YARN: schedules work on the cluster

HDFS

Data are split into chunks (128 MB each) and distributed to nodes (computers in a cluster)

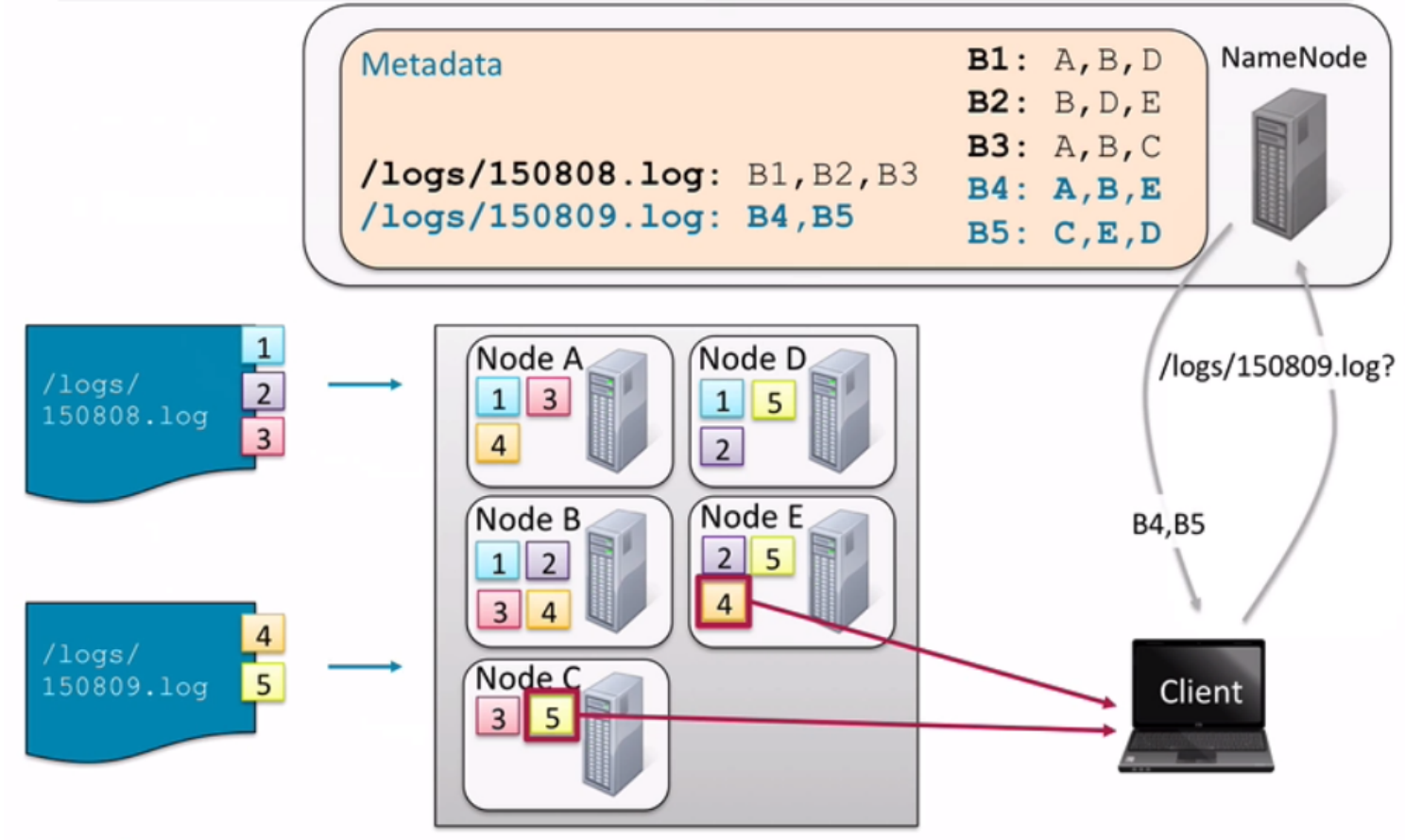
Key difference: each chunk is replicated on more than one node (usually 3x); information about where, is stored on namenode



Note: if a node fails, nothing is lost and the failed node is automatically re-replicated on yet another node

Example: online store data processing

Point: clients get the locations (of web log files) from NameNode but goes directly to data nodes for data/analytics



Source: Cloudera

Getting data in and out of HDFS

- "manual": using the command line
- "automated": using Ecosystem projects:
  - Flume, Kafka:
    - collect data from networks (websites, system logs, streaming data from Twitter, sensor data, network traffic, emails messages, GPS tracking info, etc)
    - allows for Big Data analytics as that data is being generated, on-the-fly (e.g, threat and fraud detection, self-driven cars, twitter sentiment)
  - Sqoop (Sequel to Hadoop):
    - collects data from RDBMS (Relational Database Management Systems) such as SQL Servers, MySQL.
    - allows for the use of massive tables from data warehouses (e.g., stores, production lines, back-ends of a web-server).
    - custom "connectors" between RDBMS and HDFS such as Teradata and Oracle.

MapReduce on Hadoop

MapReduce allows to use the CPUs and RAM of individual computers on a cluster to obtain estimates based on divide-and-conquer and subsampling methods, and then combine them to build a learner – see Week 8.

YARN and Spark/Hive/Impala

Spark is a newer version of the MapRaduce framework, better adapted to iterative methods (e.g., MLE); usually works faster than original MapReduce

Hive is a framework that implements MapReduce/Spark using an SQL type language; needs no developer (to write a code in Matlab or Python or R); SQL is an easy language to learn, used in data warehousing/retrieval

YARN (Yet Another Resource Negotiator) allocates computing resources between all these frameworks (MapReduce, Spark, Hive, Impala, etc) that are used by Hadoop.

Impala

Is a new word in parallel computing on Hadoop;

It uses an SQL-type language like Hive;

Does not use MapReduce; uses Dremel, a way of optimally querying massive data sets;

Seem more like a optimal query engine to me;

Originates from Google Research:

Melnik, S.; Gubarev, A.; Long, J.-J.; Romer, G.; Shivakumar, Sh.; Tolton, M.; Vassilakis, Th. "Dremel: Interactive Analysis of Web-Scale Datasets". *Proceedings of the 36th Int'l Conference on Very Large Data Bases*
`https://research.google.com/pubs/pub36632.html`

Concluding remarks

- Hadoop is the industry standard in Big Data analytics.
- Hadoop allows to make use of parallel computing to implement the divide-and-conquer and subsampling methods on a large scale.
- Hadoop has a large and quickly growing Ecosystem covering problems of storage, on-the-fly analytics, search, security and access, etc.

Recommended reading

- Cloudera tutorial:
  `https://www.cloudera.com/more/training/library/hadoop-essentials.html`
- Chapter 2 of *Mining of Massive Datasets*, by Leskovic, J, Rajaraman, A and Ullman, J.
  `http://www.mmds.org/`