# Data 603 – Big Data Platforms

UMBC

Lecture 3
MapReduce & Cloud Computing

## What we will cover today

- Git/Github Review
- MapReduce
- MapReduce Demo
- Cloud Computing
- Cloud Computing Demo

## Git/Github Review

## MapReduce

## MapReduce Definition 1

"Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner."

[Source: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html](https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html)

## MapReduce Definition 2

"*MapReduce* is a computing model that decomposes large data manipulation *jobs* into individual *tasks* that can be executed in parallel across a cluster of servers. The results of the tasks can be joined together to compute the final results."

Source: https://www.oreilly.com/library/view/programming-hive/9781449326944/ch01.html

## MapReduce in a Nutshell

- MapReduce is a programming model for processing **large data sets** and generating results developed at Google.
- Runs on a large cluster of **commodity hardware** and is highly scalable
- Processes many **terabytes of data** on thousands of machines
- Uses the **Master–Worker** model
  - Widely used on distributed and loosely coupled systems

## MapReduce

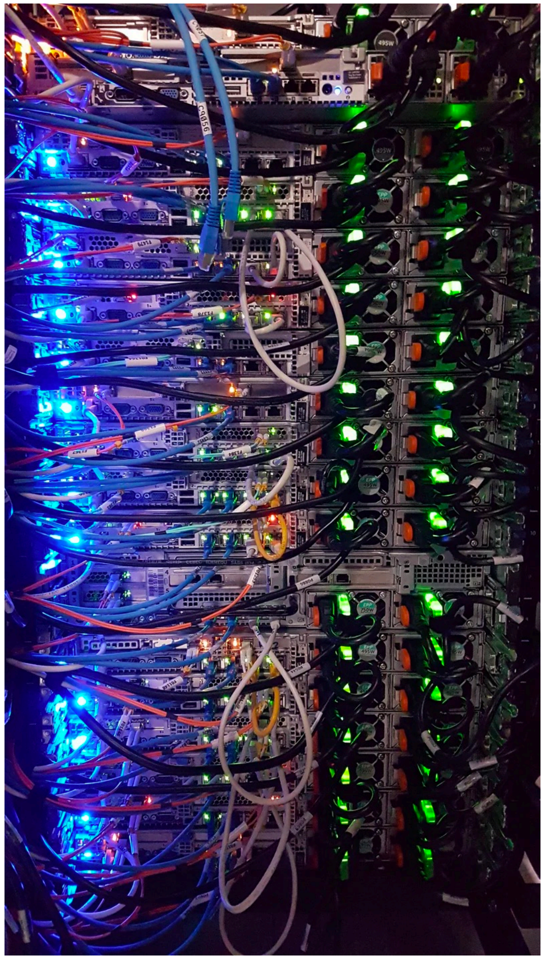
- The goal: Abstracting away the complexity of parallel computing from the users
  - Easy utilization of distributed parallel computing

… but HOW?

## MapReduce

- The run time (e.g. Hadoop) takes care of
  - The details of the partitioning of input data
  - Scheduling the program's execution across nodes
  - Managing inter-node communications
  - Handling of node failures

You only need to focus on the code logic!



## MapReduce

MapReduce = Map + Reduce

- **Map:**
  - Accepts key/value pair
  - Emits *intermediate* key/value pair
- **Reduce :**
  - Accepts *intermediate* key/value pair
  - Emits *output* key/value pair

- We will review some examples of the MapReduce application

## MapReduce

- Processing is broken into two phases: Mapping phase followed by the reduce phase
- For each phase a key-value pair is passed as the input
- Each phase returns a key-value pair
- The types of key and value can be chosen by the programmer

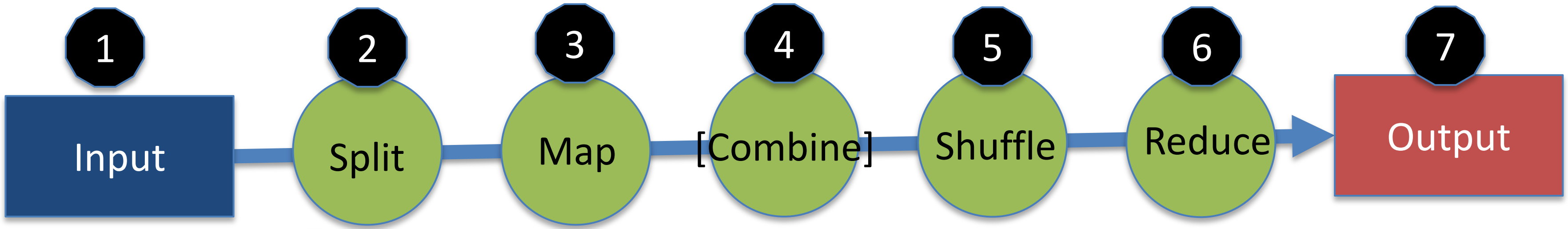
## Map Phase

- Map phase is done by mappers
- Mappers take unsorted key/value pairs
- Mapper returns zero, one or multiple key/value pairs for each input key/value pair.
- Input and output keys might be different.
- Input and output values might be different.

## Reduce Phase

- Reduce phase is done by reducers
- Reducer takes key/value_list pairs sorted by the key.
  - The sorting is done by the framework (e.g. Hadoop)
- Value_list contains all values with the same key produced by mappers.
- Returns zero, one or many key/value pairs for each input key/value_list pair
- Goal: Convert value_list to a value (e.g. sum, average, etc.)

## There is more to MapReduce than map and reduce



1 Input Data/File

2 Input files are split by Hadoop into smaller 'splits'

3 Mapper task

4 Combiner (optional) – used to improve the performance by reducing the amount of data transferred across the wire via aggregation of map result

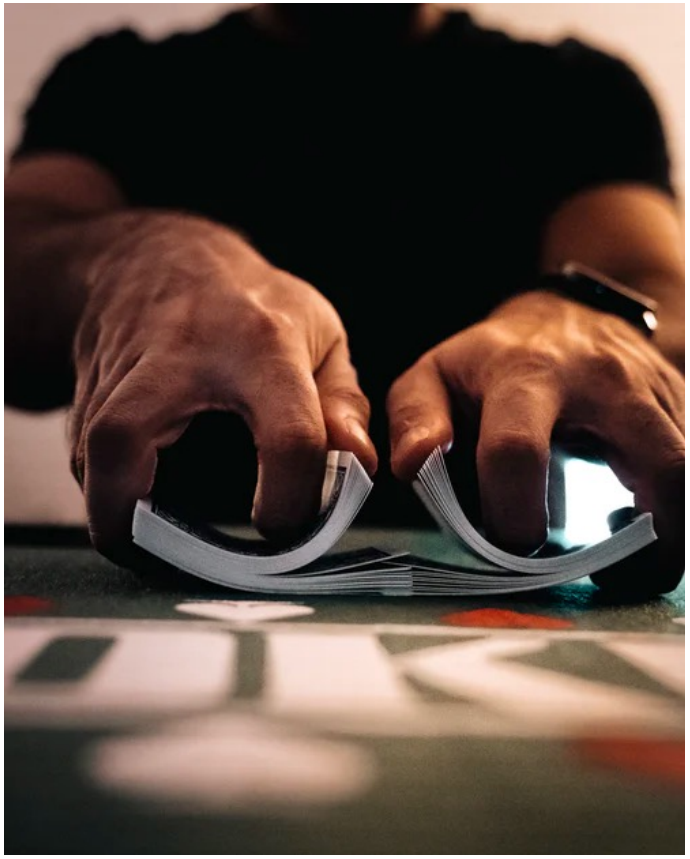
5 Outputs of mapper are sorted and shuffled

6 Reducer task

7 Output file

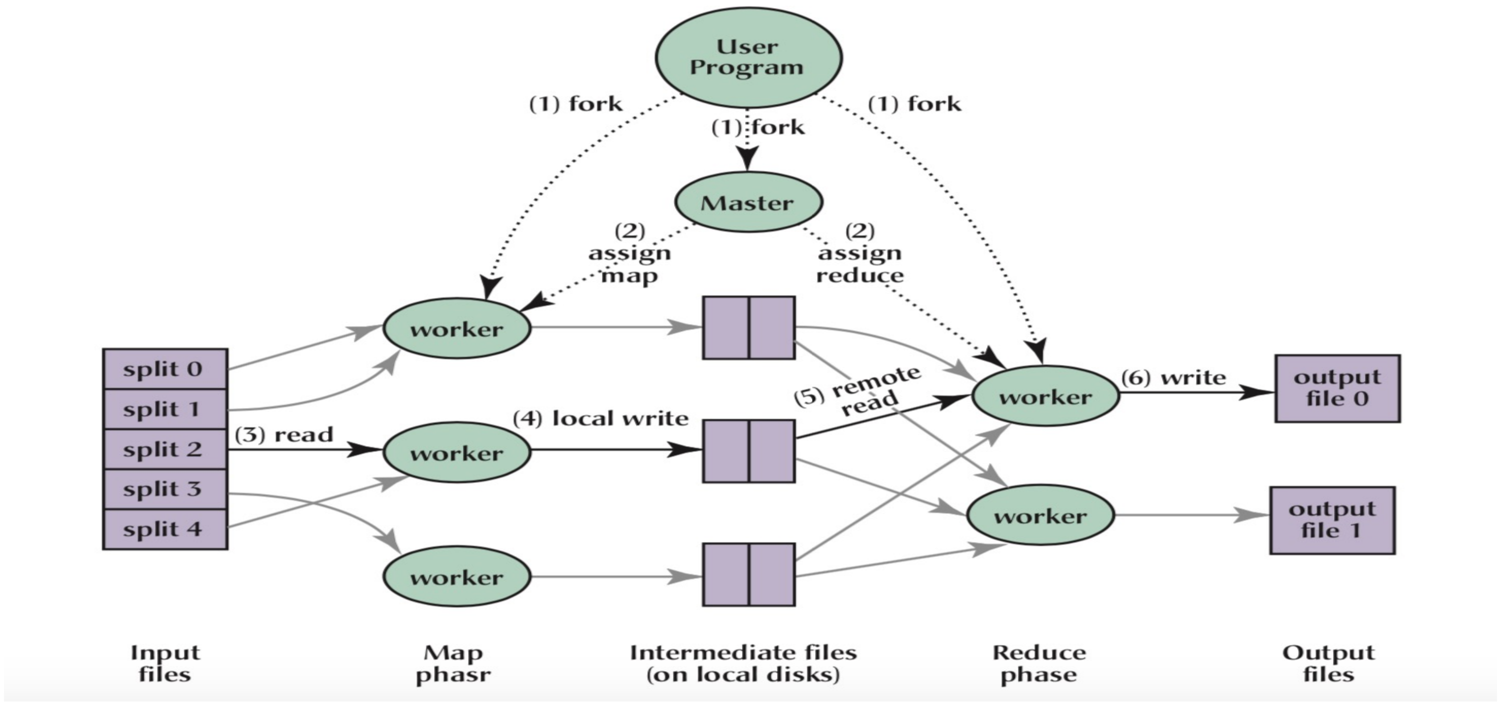
Adapted from
https://www.mssqltips.com/sqlservertip/3222/big-data-basics--part-5--introduction-to-mapreduce/

## Shuffle Time!



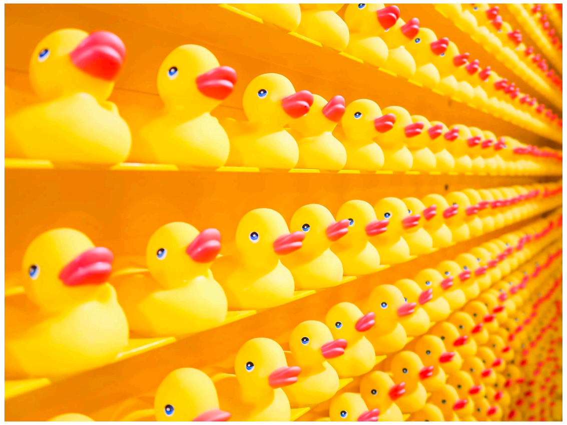
## Shuffle Time!

Hadoop sorts the key-value pairs by key and it “shuffles” all pairs with the same key to the same *Reducer*.



## Hadoop's Role

- Makes sure that MapReduce jobs run successfully
- Breaks down submitted jobs into map and reduce tasks
- Schedules running of tasks based on available resources
- Decides where to send tasks in the cluster (closer to the data as much as possible to reduce the network latency)
- Monitors each task for success/failure
- Restarts failed tasks



## What's in a MapReduce Job?

- A unit of work to be performed
- A job consists of input data, MapReduce program and configuration information
- Hadoop divides a job into a number of tasks
  - Map tasks
  - Reduce tasks
- Tasks are scheduled using YARN and run on nodes in the cluster.
  - A failed task will be automatically rescheduled to run on a different node

## Wait … What is YARN?

## Execution overview

1. MapReduce Library splits the input files into M pieces of 16 MB to 64 MB
2. The master picks idle workers and assigns each one a map task or a reduce task
3. Worker-Parses key/value pairs out of input and passes to user defined Map function
4. The intermediate key/value pairs produced by the Map function are buffered in memory
5. Buffered pairs are written to local disk, partitioned into R regions by the partitioning function
6. Sorts intermediate data by the intermediate keys so that all occurrences of the same key are grouped together
7. Output of the Reduce function is appended to a final output file
8. MapReduce call in the user program returns back to the user code

# Yet Another Resource Negotiator (YARN)

- Also called MR v2
- Goal: Splitting up of the functionalities into separate daemons.
  - Resource management (global resource manager)
  - Job scheduling/monitoring (per-application Application Manager)
- ResourceManager (RM): Arbitrates resources among all the applications in the system. Replaces the Job Tracker in MR v1
- NodeManager: Per-machine framework agent. Replaces Task Tracker in MR v1
  - Responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.
- Per-application ApplicationMaster (AM): A framework specific library tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

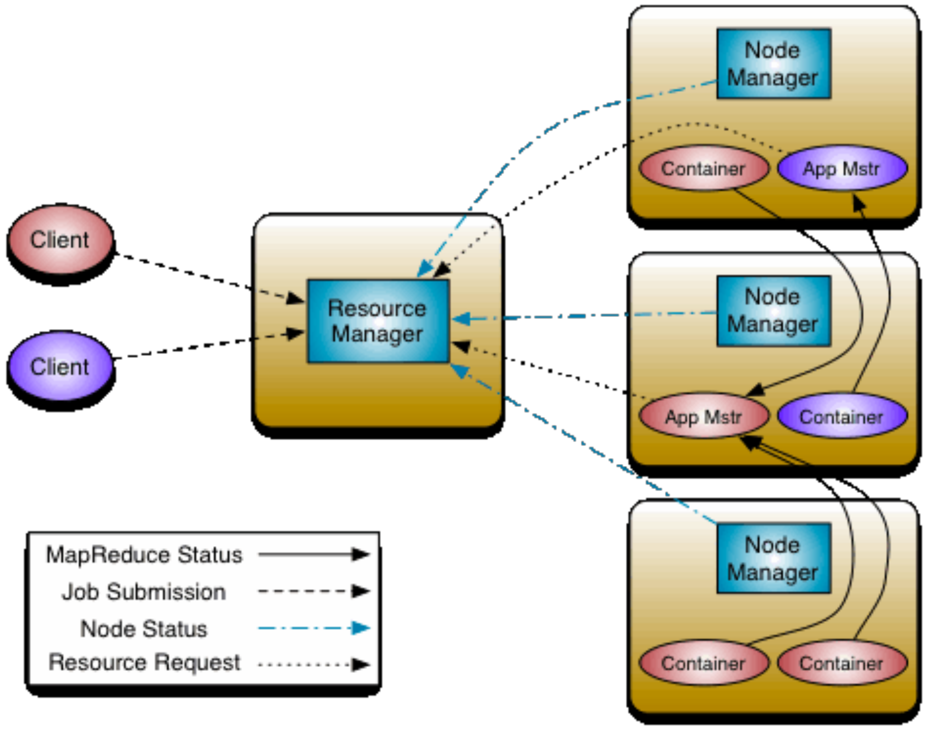
## Yet Another Resource Negotiator (YARN)

ResourceManger has two main components:

- Scheduler: Responsible for allocating resources to the running applications subject to constraints of capacities, queues etc.
  - It performs no monitoring or tracking of status for the application.
  - It offers no guarantees about restarting failed tasks either due to application failure or hardware failures.
  - The Scheduler performs its scheduling function based on the resource requirements of the applications.
- Applications Manager: Responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure.

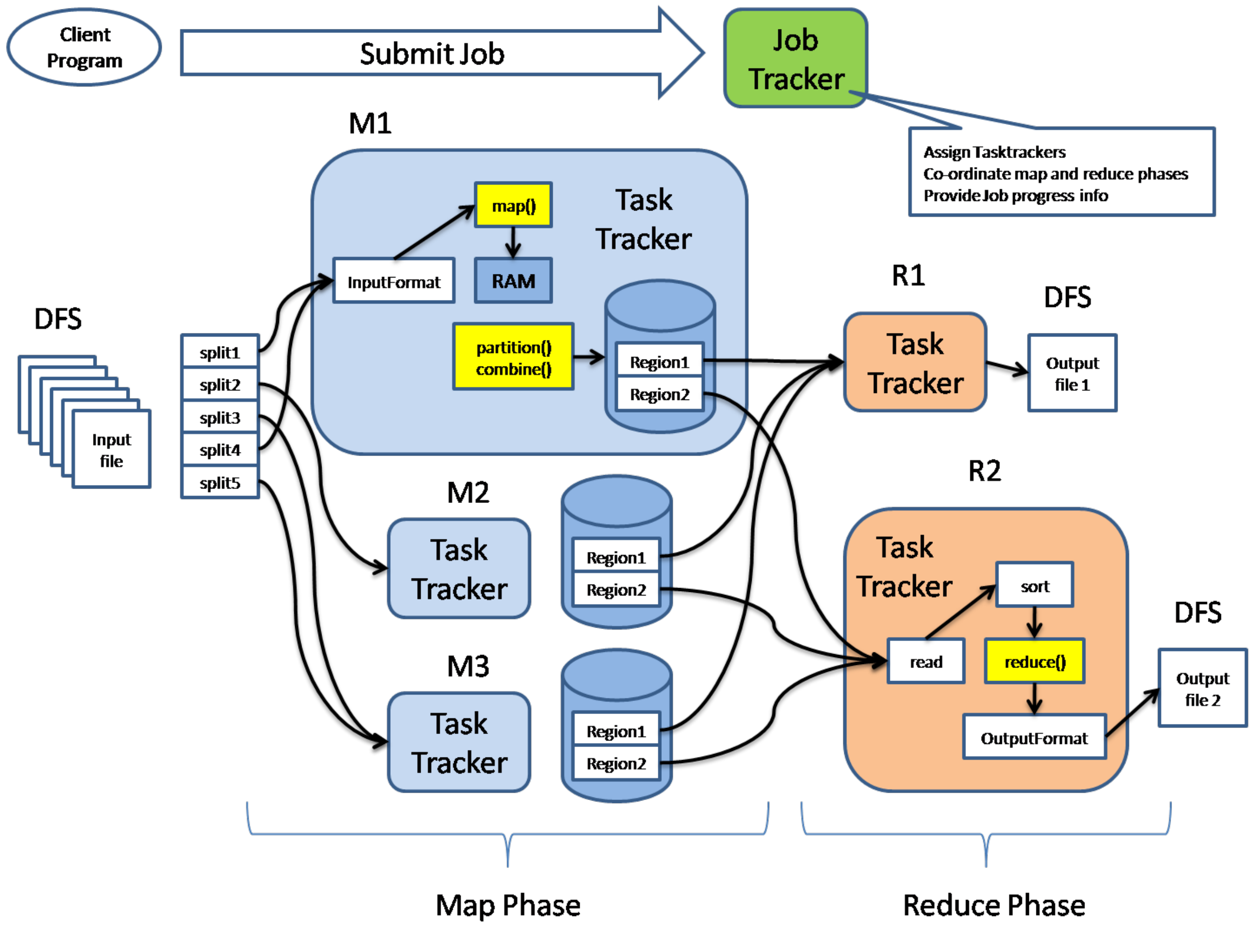
https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

## Yet Another Resource Negotiator (YARN)

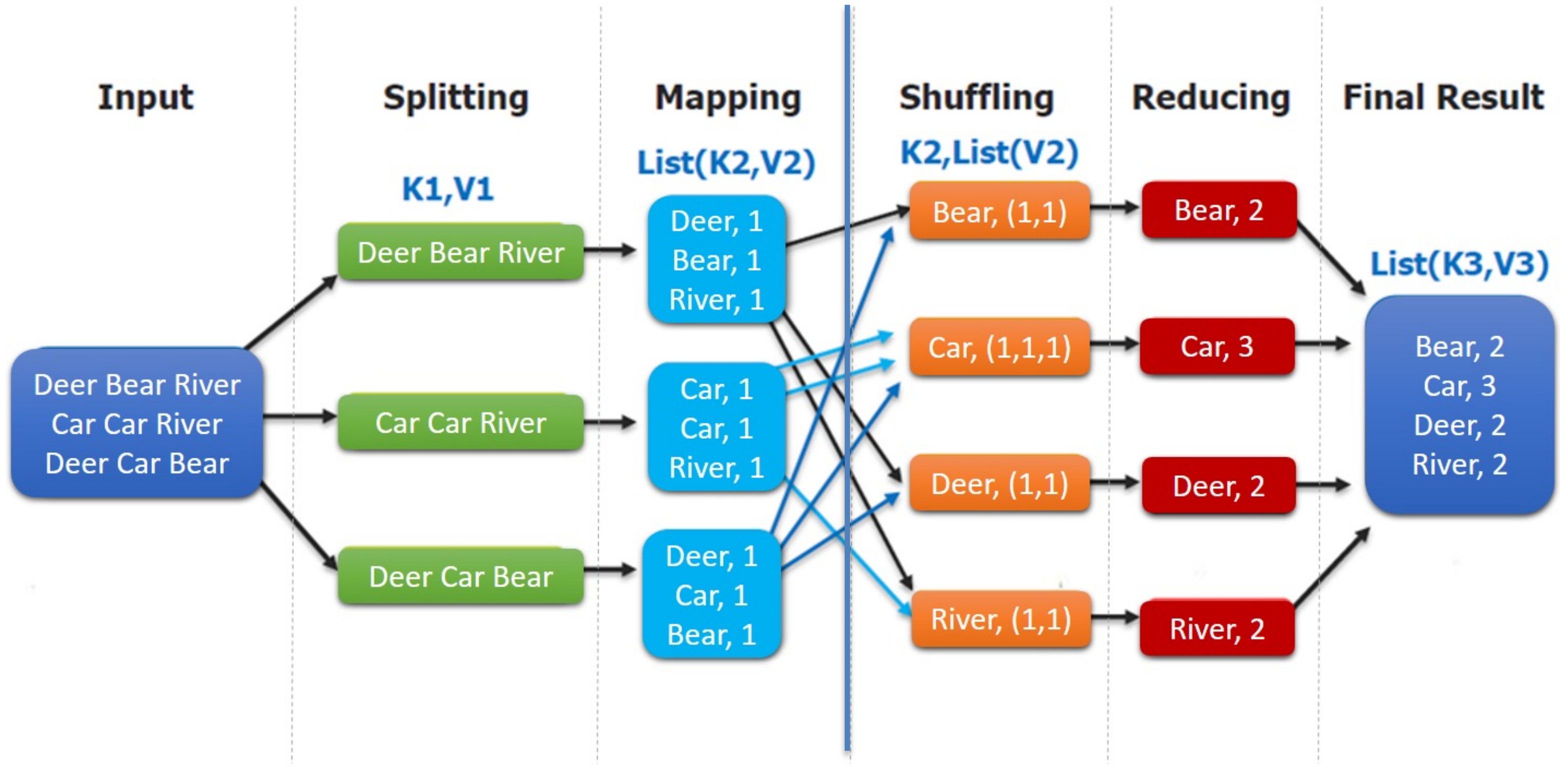


https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

# MapReduce Processing Details



# Example: Word Count



MapReduce Word Count Process

## **Problem of counting the number of occurrences of each word in a large collection of documents** : (Example)

```
map(String key, String value):
// key: document name
 // value: document contents
 for each word w in value:
 EmitIntermediate(w, "1");

reduce(String key, Iterator values):
 // key: a word
// values: a list of counts
 int result = 0;
for each v in values:
result += ParseInt(v);
Emit(AsString(result));
```

The user writes code to fill in a MapReduce specification object with the names of the input and output files, and optional tuning parameters. The user then invokes the MapReduce function, passing it the specification object. The user's code is linked together with the MapReduce library

The map function emits each word plus an associated count of occurrences (just '1' in this simple example). The reduce function sums together all counts emitted for a particular word.

## Example: X-Ref Indexing

- For the cross-reference/indexing example:
  - The Map function parses each document and emits a sequence of <word, document ID>
  - The Reduce function accepts all pairs for a given work, sorts the corresponding document ID and emits a <word, list(document ID)>
  - The set of all output pairs forms the simple answer

## Example: Distributed Grep

- Grep is very popular command in Unix and Linux
- Definition:
  - The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression
  - Grep stands for Globally search for Regular Expression and Print out

## Example: Distributed Grep (Cont.)

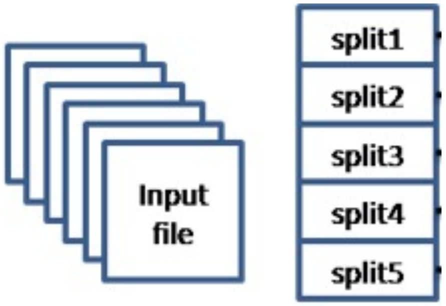
- Distributed Grep
  - The Map function emits <word, line_number> if word matches search criteria
  - The reduce function is the identity function that copies the supplied intermediate data to the output

## More Examples

- **Count of URL Access Frequency**
  - The **map function** processes logs of web page requests and outputs ⟨URL, 1⟩
  - The **reduce function** adds together all values for the same URL and emits a ⟨URL, total count⟩ pair.
- **Reverse Web-Link Graph**
  - The **map function** outputs ⟨target, source⟩ pairs for each link to a target URL found in a page named source
  - The **reduce function** concatenates the list of all source URLs associated with a given target URL and emits the pair: ⟨target, list(source)⟩

## Refinements

- Task Granularity (Splitting)
  - Mapping the task into large set of smaller tasks allows us to benefit as follows:
    - Minimizes time for fault recovery
    - Load balancing
    - Local execution for debugging/testing
    - Compression of intermediate data



## Input Splits

- An input for a MapReduce job is divided into fixed-size pieces called *input splits*.
- A map task is created for each split
  - Runs user-defined map function for each record in the split.
- Divide and Conquer
  - Processing each split is faster than processing the whole input
  - Splits are processed in parallel

## Input Splits

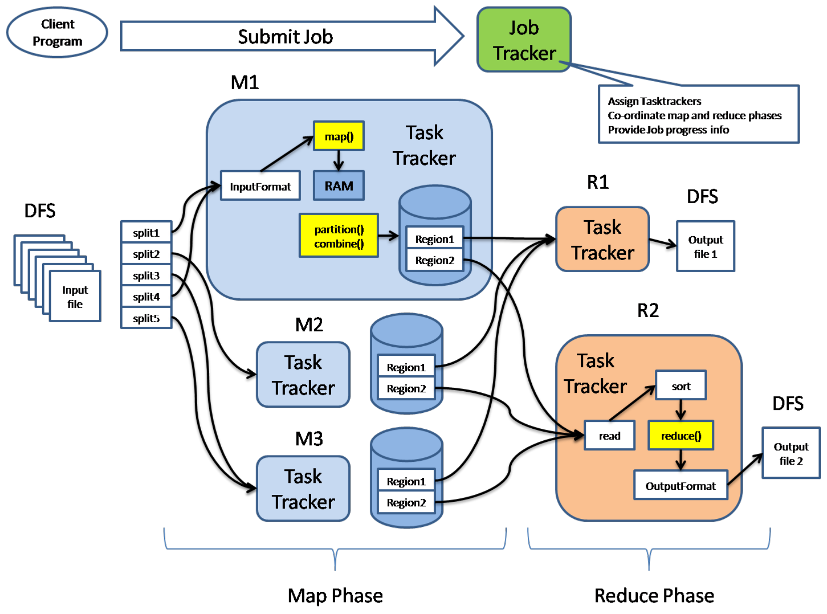
- Size Matters
  - Smaller split size better?
    - Pro
      - Better load balance – faster machines will be able to pick up more splits
    - Con
      - Overhead of managing the splits and time to create map task
  - Rule of thumb
    - Use the size of an HDFS block (128 MB by default)

## Fault Tolerance

- Worker failure
  - Workers are periodically pinged by master
    - NO response = failed worker
  - If the processor of a worker fails, the tasks of that worker are reassigned to another worker (the Master is in charge)
- Master failure
  - Master writes periodic checkpoints
  - Another master can be started from the last checkpointed state
  - If eventually the master dies, the job will be aborted

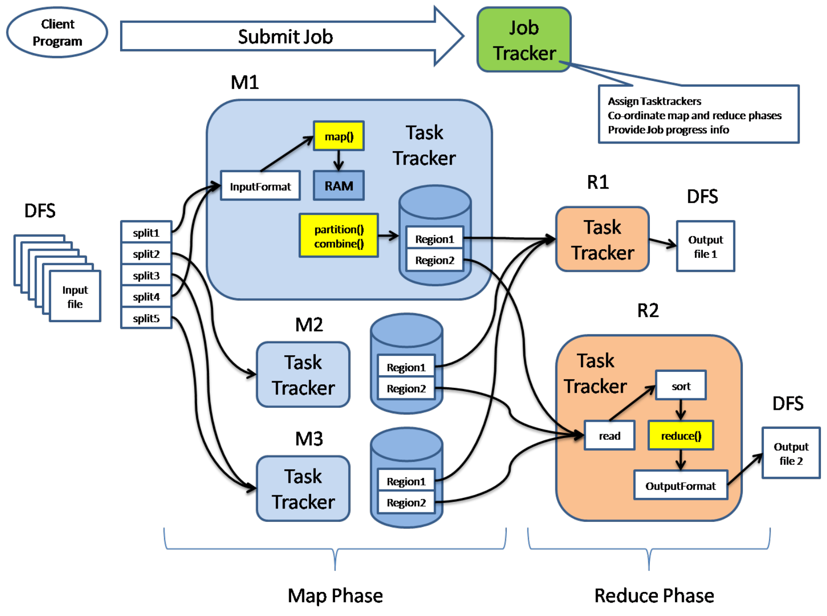
## MapReduce Characteristics

- Master-Worker model is used
- Sometimes, Master is referred to as **Job Tracker** and Worker is referred to as **Task Tracker**
- No *reduce* can begin until *map* is complete
- Master must communicate locations of intermediate files
- Tasks scheduled based on location of data



## MapReduce Fault Tolerance

- Failures:
  - If Worker fails, Master assigns new Worker
    - Max number of failures is configurable
    - Applies to both Map workers and Reduce workers
  - If *map* worker fails any time before *reduce* finishes (losing the intermediate results), the Reduce task must completely rerun



# MapReduce

## More Readings:

- MapReduce: Simplified Data Processing on Large Clusters: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/16cb30b4b92fd4989b8619a61752a2387c6dd474.pdf
- Wikipedia: https://en.wikipedia.org/wiki/MapReduce
- Hadoop MapReduce Tutorial: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- https://www.javacodegeeks.com/2012/05/mapreduce-questions-and-answers-part-1.html
- https://www.mssqltips.com/sqlservertip/3222/big-data-basics--part-5--introduction-to-mapreduce/
- An Overview of Hadoop and MapReduce : https://www.oreilly.com/library/view/programming-hive/9781449326944/ch01.html
- YARN Federation: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/Federation.html

## Python MapReduce Packages

- MrJob: https://mrjob.readthedocs.io/en/latest/
- Pydoop: http://crs4.github.io/pydoop/
- And more …

## Using mrjob

**mrjob** is a Python package that allows you to write MapReduce jobs in Python and run them on several platforms. It allows:

- Write multi-step MapReduce jobs in pure Python
- Test on your local machine
- Run on a Hadoop cluster
- Run in the cloud using Amazon Elastic MapReduce (EMR)
- Run in the cloud using Google Cloud Dataproc (Dataproc)
- Easily run *Spark* jobs on EMR or your own Hadoop cluster

Source: https://mrjob.readthedocs.io/en/latest/

## Using mrjob

- Create a Python class that inherits from MRJob.
- The new class contains methods that define the steps (map/reduce) of the MapReduce job
- Mapper method
  - mapper()
  - Takes a key and a value as parameters
  - Yields many key-value pairs
- Reducer method
  - reducer ()
  - Takes a key and an iterator of values
  - Yields many key-value pairs

## MapReduce Exercise

Write a MapReduce job that prints the number of occurrences of each word within a body of text

```
from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"[\w']+")


class MRWordFreqCount(MRJob):

    def mapper(self, _, line):
    # Mapper definition here

    def reducer(self, word, counts):
    # Reducer definition here

    if __name__ == '__main__':
MRWordFreqCount.run()
```

## Q: Who invented MapReduce?

## Q: Who invented MapReduce?

**A:** https://www.fiware.org/2014/05/30/mapreduce-was-really-invented-by-julius-caesar/

## Cloud Computing

## What is Cloud Computing

- Renting resources
  - Compute Power
  - Storage
  - Networking
  - Analytics
- Only pay for what you use
- Cloud Providers (Azure, AWS, GCP) manage the physical hardware and the infrastructure

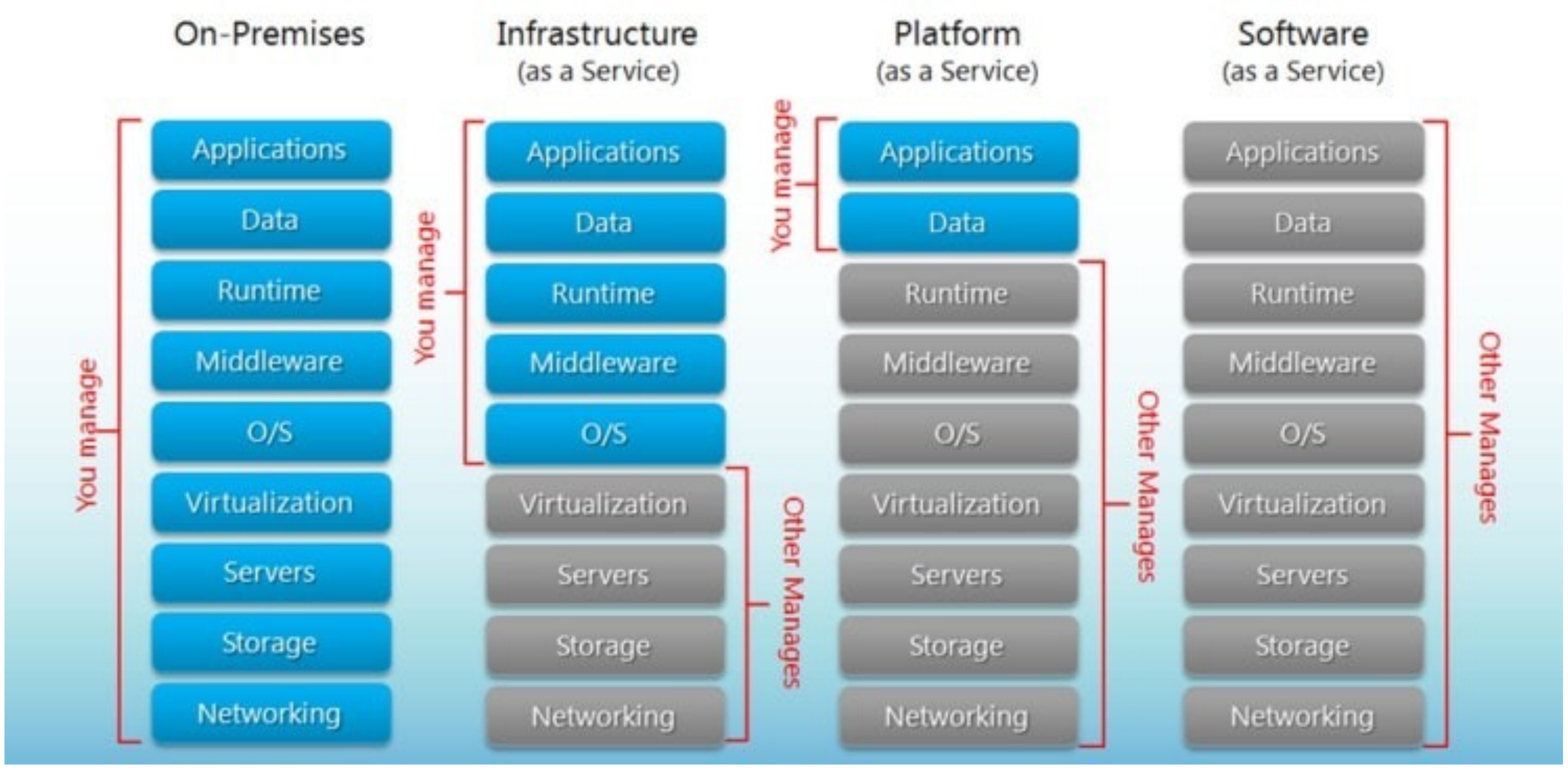
## Benefits of Cloud Computing

- Cost-effective – only pay for what you use
  - No infrastructure cost
  - No need to maintain own servers, network, storage.
- Scalable
  - Scaling out/Horizontal Scaling
- Elasticity
  - Automatically adding or removing resources based on the workload

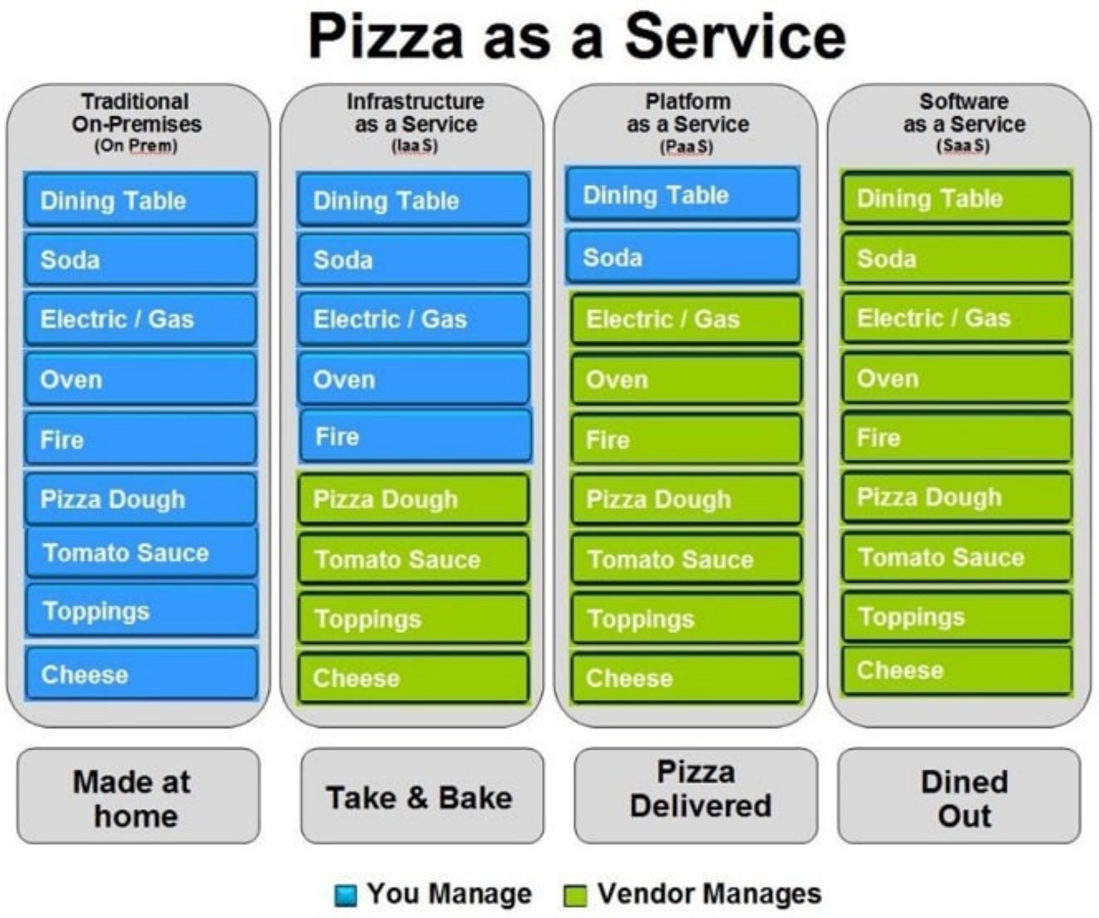
## Benefits of Cloud Computing

- Reliability
  - Data backup, disaster recovery, data replication
- Security
  - Shared responsibility

# Cloud Service Categories



## Cloud Service Categories



Pizza as a Service

| Traditional On-Premises (On Prem) | Infrastructure as a Service (Iaa S) | Platform as a Service (Paa S) | Software as a Service (Saa S) |
|---|---|---|---|
| 🟦 Dining Table | 🟦 Dining Table | 🟦 Dining Table | 🟩 Dining Table |
| 🟦 Soda | 🟦 Soda | 🟦 Soda | 🟩 Soda |
| 🟦 Electric / Gas | 🟦 Electric / Gas | 🟩 Electric / Gas | 🟩 Electric / Gas |
| 🟦 Oven | 🟦 Oven | 🟩 Oven | 🟩 Oven |
| 🟦 Fire | 🟦 Fire | 🟩 Fire | 🟩 Fire |
| 🟦 Pizza Dough | 🟩 Pizza Dough | 🟩 Pizza Dough | 🟩 Pizza Dough |
| 🟦 Tomato Sauce | 🟩 Tomato Sauce | 🟩 Tomato Sauce | 🟩 Tomato Sauce |
| 🟦 Toppings | 🟩 Toppings | 🟩 Toppings | 🟩 Toppings |
| 🟦 Cheese | 🟩 Cheese | 🟩 Cheese | 🟩 Cheese |
| Made at home | Take & Bake | Pizza Delivered | Dined Out |

🟦 You Manage 🟩 Vendor Manages

## Cloud Data Analytics Services

- Data Warehouse (Azure Synapse, AWS Redshift)
- Advanced Analytics (Databricks, HDInsight, AWS EMR)
- Data Integration Service (Azure Data Factory)
- Machine Learning (Azure ML, AWS Sage Maker)
- Storage (Azure Data Lake, AWS S3)

## Data Lake

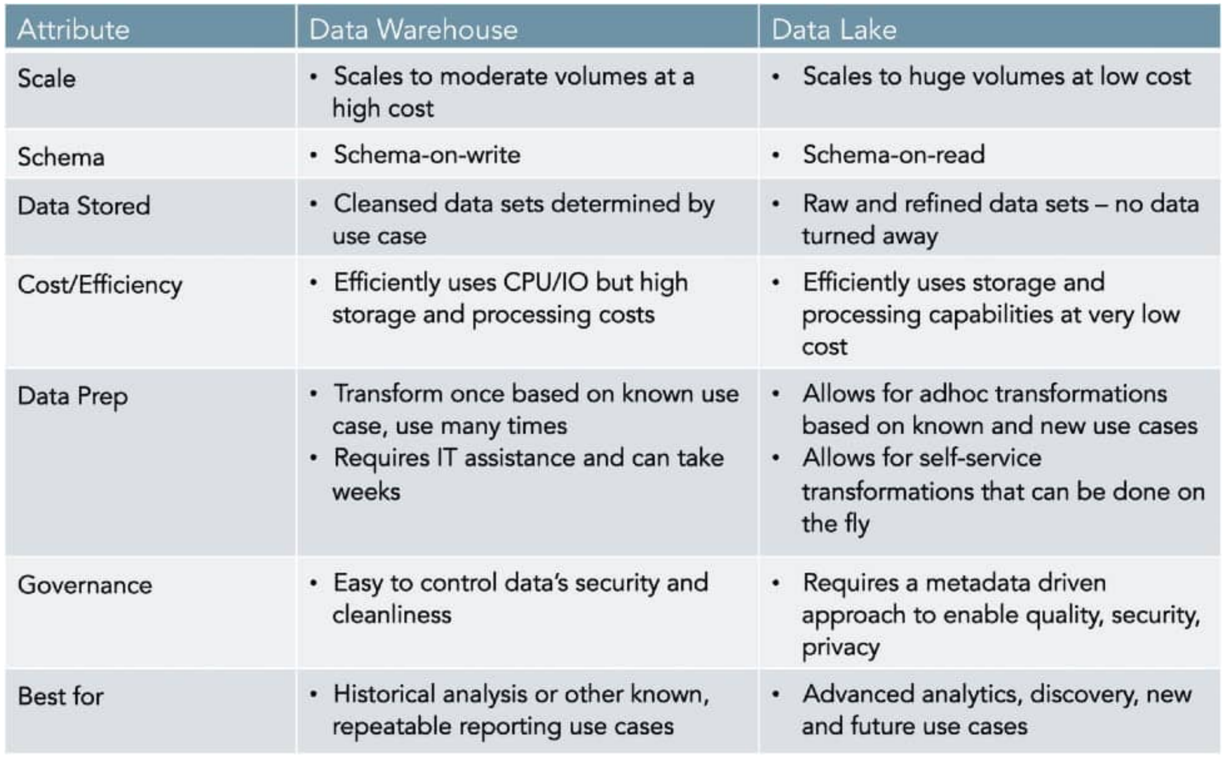
# Data Lake

A **data lake** is a system or repository of data stored in its natural/raw format,[1] usually object blobs or files. A data lake is usually a single store of data including raw copies of source system data, sensor data, social data etc.,[2] and transformed data used for tasks such as reporting, visualization, advanced analytics and machine learning. A data lake can include structured data from relational databases (rows and columns), semi-structured data (CSV, logs, XML, JSON), unstructured data (emails, documents, PDFs) and binary data (images, audio, video).[3] A data lake can be established "on premises" (within an organization's data centers) or "in the cloud" (using cloud services from vendors such as Amazon, Microsoft, or Google).

A **data swamp** is a deteriorated and unmanaged data lake that is either inaccessible to its intended users or is providing little value.[4]

https://en.wikipedia.org/wiki/Data_lake

## Data Lake vs. Data Warehouse



| Attribute | Data Warehouse | Data Lake |
|---|---|---|
| Scale | • Scales to moderate volumes at a high cost | • Scales to huge volumes at low cost |
| Schema | • Schema-on-write | • Schema-on-read |
| Data Stored | • Cleansed data sets determined by use case | • Raw and refined data sets – no data turned away |
| Cost/Efficiency | • Efficiently uses CPU/IO but high storage and processing costs | • Efficiently uses storage and processing capabilities at very low cost |
| Data Prep | • Transform once based on known use case, use many times<br>• Requires IT assistance and can take weeks | • Allows for adhoc transformations based on known and new use cases<br>• Allows for self-service transformations that can be done on the fly |
| Governance | • Easy to control data's security and cleanliness | • Requires a metadata driven approach to enable quality, security, privacy |
| Best for | • Historical analysis or other known, repeatable reporting use cases | • Advanced analytics, discovery, new and future use cases |

https://www.zaloni.com/resources/blog/why-smart-companies-are-complementing-their-data-warehouses-with-data-lakes/

## Homework

- THIS HOMEWORK IS HARDER THAN PREVIOUS ONES. Start working early and ask questions often
- Download **yelp.csv** from Kaggle

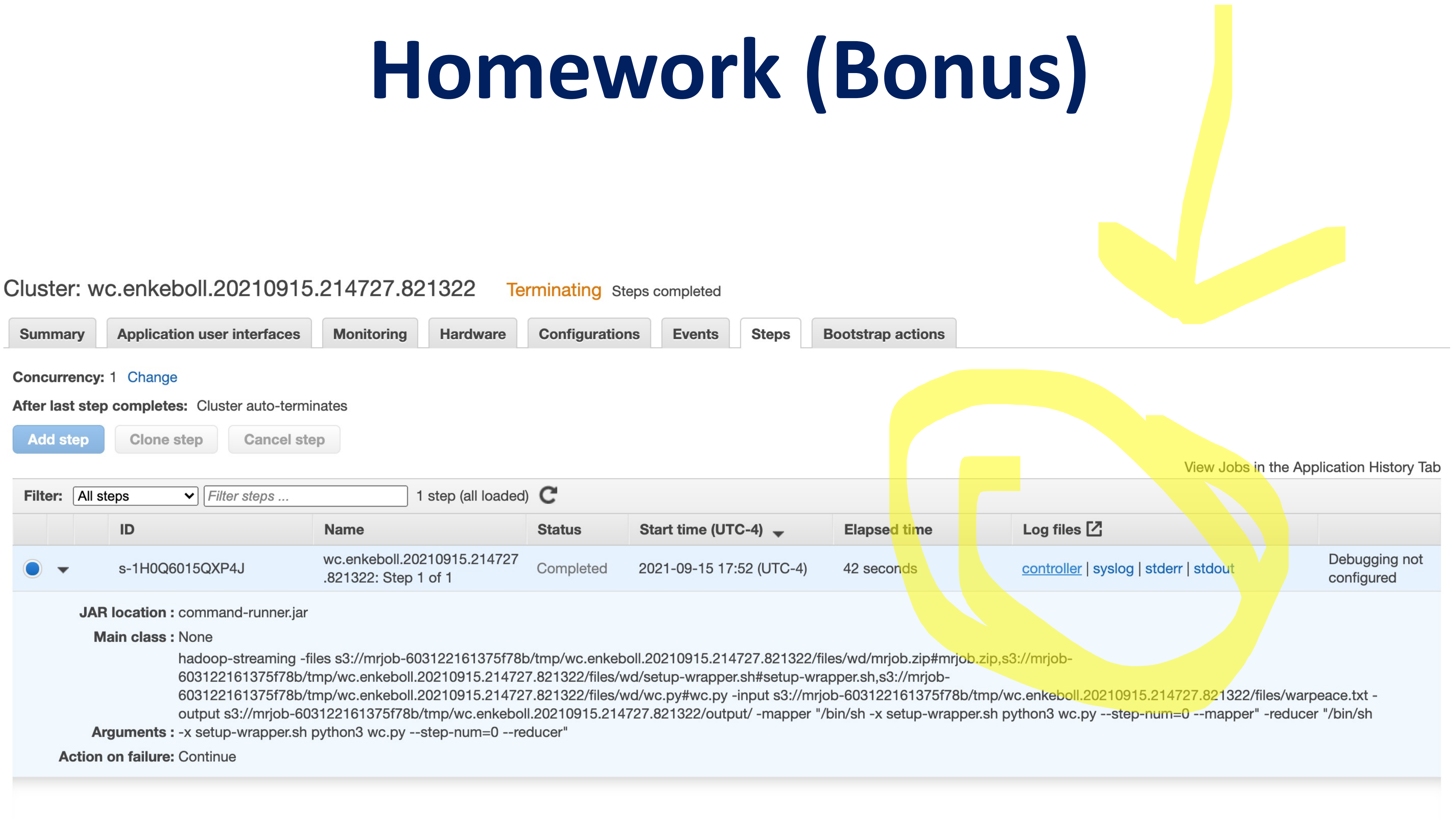
## Homework

- Write three Python scripts using **mrjob** that tell me:
  1. Average number of words in each review (define "words" however you like but be explicit about it)
  2. Count of reviews by year-month (eg "**2021-09**")
  3. Average rating of any review marked "cool" (eg where cool != 0)

## Homework

- Bonus (2 points): run the homework on Amazon EMR
- Submission: same as before
- Filenames should be **homework/hw03-[1|2|3].py**
- Bonus should be the file **controller** from EMR Console

# Homework (Bonus)


Cluster: wc.enkeboll.20210915.214727.821322 Terminating Steps completed

Summary | Application user interfaces | Monitoring | Hardware | Configurations | Events | Steps | Bootstrap actions

**Concurrency:** 1 Change

**After last step completes:** Cluster auto-terminates

Add step | Clone step | Cancel step

View Jobs in the Application History Tab

Filter: All steps | Filter steps ... | 1 step (all loaded)

| ID | Name | Status | Start time (UTC-4) | Elapsed time | Log files | |
|---|---|---|---|---|---|---|
| s-1H0Q6015QXP4J | wc.enkeboll.20210915.214727.821322: Step 1 of 1 | Completed | 2021-09-15 17:52 (UTC-4) | 42 seconds | controller \| syslog \| stderr \| stdout | Debugging not configured |

**JAR location :** command-runner.jar

**Main class :** None

**Arguments :**
```
hadoop-streaming -files s3://mrjob-603122161375f78b/tmp/wc.enkeboll.20210915.214727.821322/files/wd/mrjob.zip#mrjob.zip,s3://mrjob-603122161375f78b/tmp/wc.enkeboll.20210915.214727.821322/files/wd/setup-wrapper.sh#setup-wrapper.sh,s3://mrjob-603122161375f78b/tmp/wc.enkeboll.20210915.214727.821322/files/wd/wc.py#wc.py -input s3://mrjob-603122161375f78b/tmp/wc.enkeboll.20210915.214727.821322/files/warpeace.txt -output s3://mrjob-603122161375f78b/tmp/wc.enkeboll.20210915.214727.821322/output/ -mapper "/bin/sh -x setup-wrapper.sh python3 wc.py --step-num=0 --mapper" -reducer "/bin/sh -x setup-wrapper.sh python3 wc.py --step-num=0 --reducer"
```

**Action on failure:** Continue

## Questions