DISTRIBUTED AND CLOUD COMPUTING

LAB 9: HADOOP DFS & MAP-REDUCE

What is Hadoop?

A framework for distributed **storing** and **processing** of HUGE **datasets** across clusters of computers

Framework: a collection of software libraries, tools and technical documents

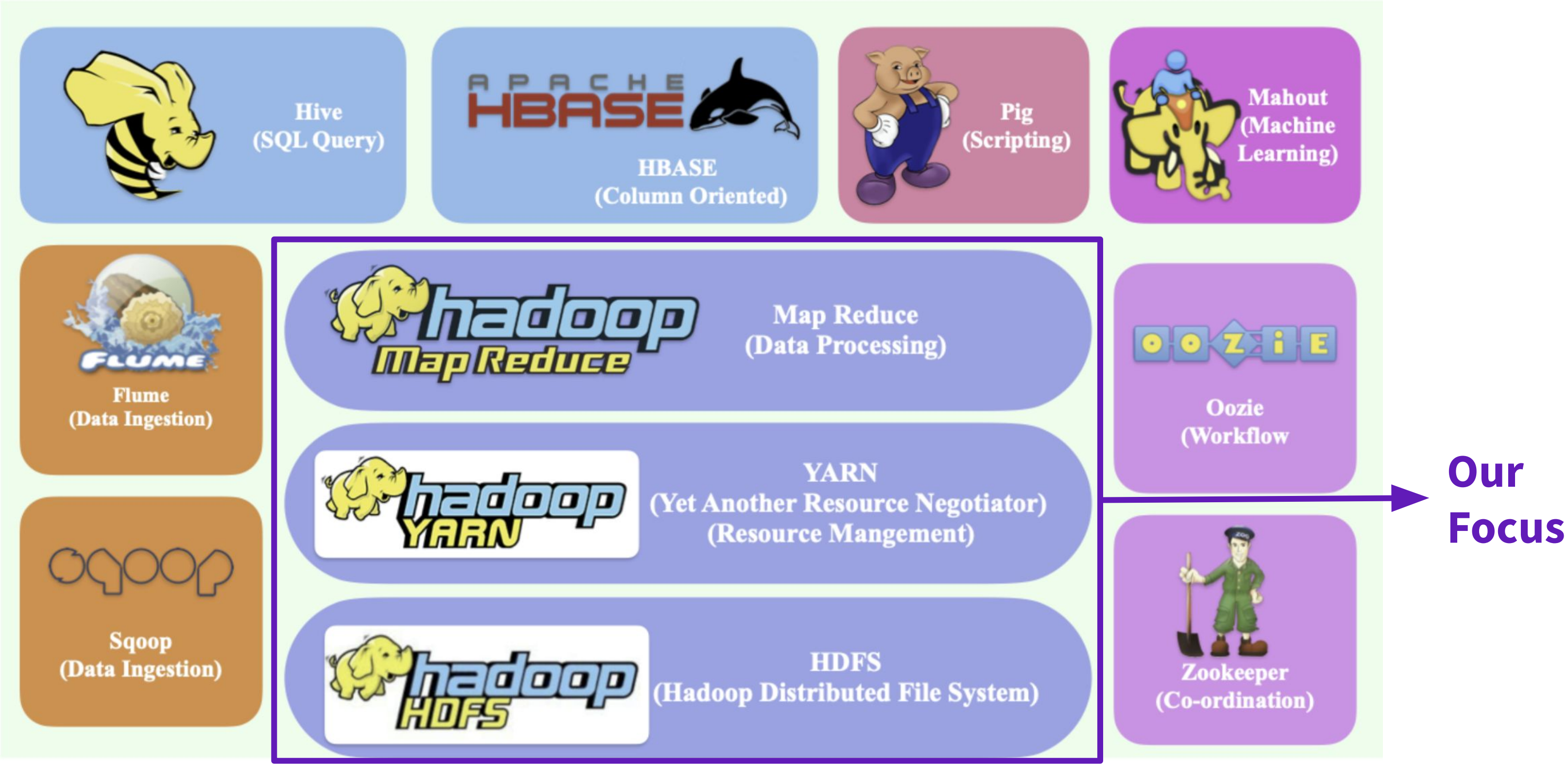
Principles and assumption behind hadoop:

1. Hardware failure is the norm rather than the exception
2. High throughput over low latency of data access
3. Assumes a “write-once-read-many” access model for files
4. Applications that run on HDFS have large data sets (100s of GBs to Terabytes)

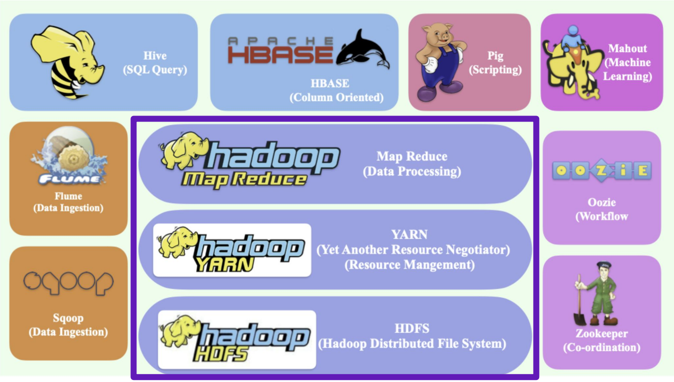
5. Moving Computation is Cheaper than Moving Data

Would you rather **upload a 1MB script** to 10 servers or **download a 1TB dataset** from them?

The Hadoop Ecosystem



A brief explanation...



Hadoop Distributed File System (HDFS)

- Fault tolerant distributed file system

Yet Another Resource Negotiator (YARN)

- Resource scheduler of Hadoop
- Assigns computation resources to jobs (e.g. MapReduce)

MapReduce

- A massively parallel, data processing paradigm based on key-value maps

https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Hadoop Distributed File System (HDFS)

"A distributed, fault-tolerant, write-once-read-many file system"

Distributed: consists of many computers

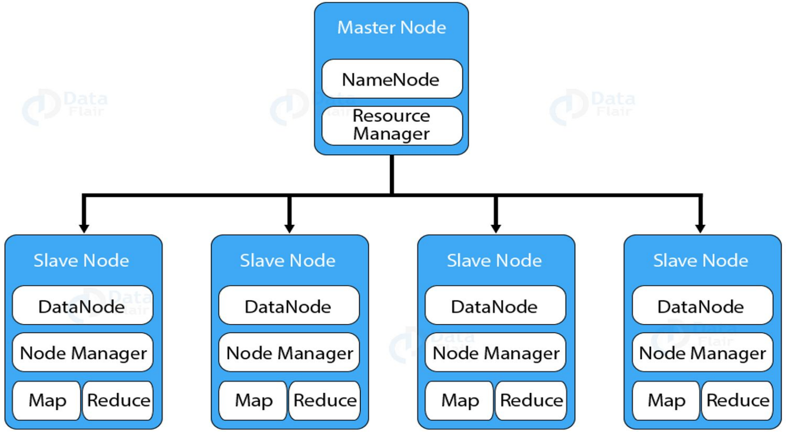
Fault-tolerant: can tolerate **some** computers failing without loss of data

Write-once-read-many: does not support modification of existing data

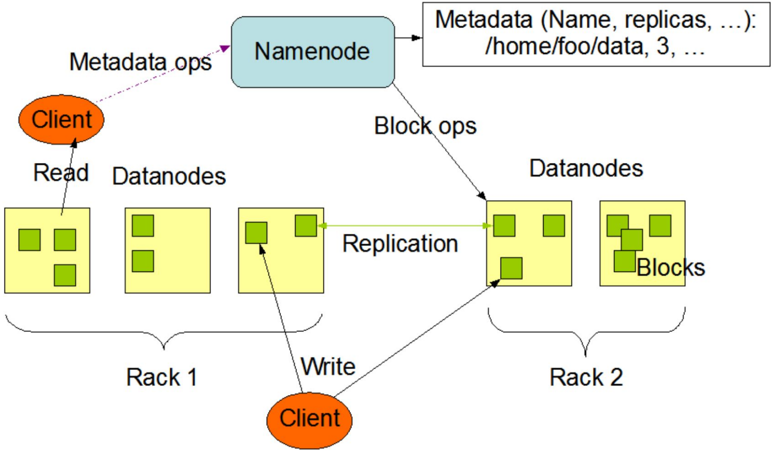
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

HDFS Architecture

- HDFS consists of
  - **1 NameNode** and **MANY DataNodes**
- **NameNode:**
  - Manages how data are distributed among the DataNodes
  - Manages the filesystems Namespace
  - Executes client requests (open, close, rename, etc..)
- **DataNodes:**
  - Store the data
  - Respond to client read and write requests after permission is given from the **NameNode**
- The Namespace of HDFS follows a traditional hierarchical file organisation
  - e.g. /users/name/dataset1



HDFS Architecture



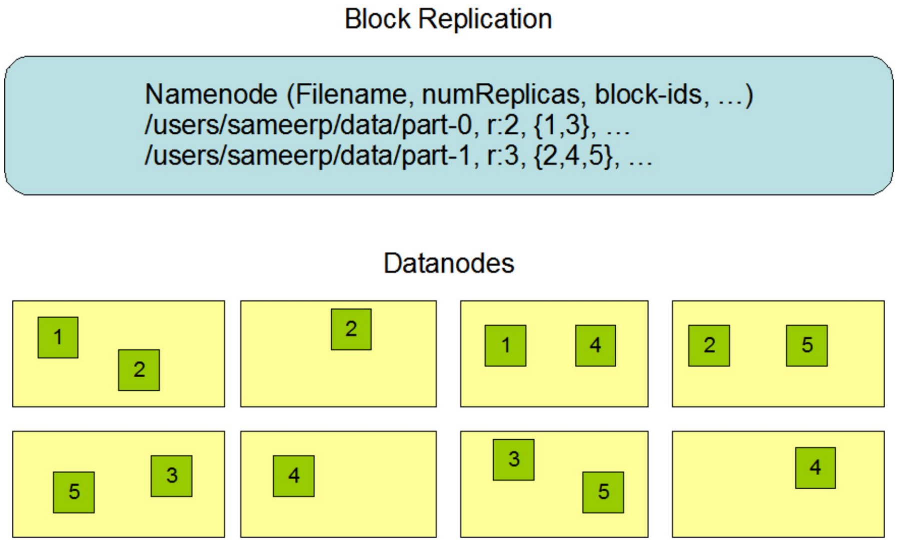
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

How HDFS store files

- **HDFS** files are split into blocks
  - Default value is 128MB
- These blocks are stored by the DataNodes as **normal files** in their local file system
- Each file has a replication factor **(r)** associated with it
  - The default replication factor is 3

The NameNode ensures that **ALL** blocks of a file are stored in at least **r** DataNodes

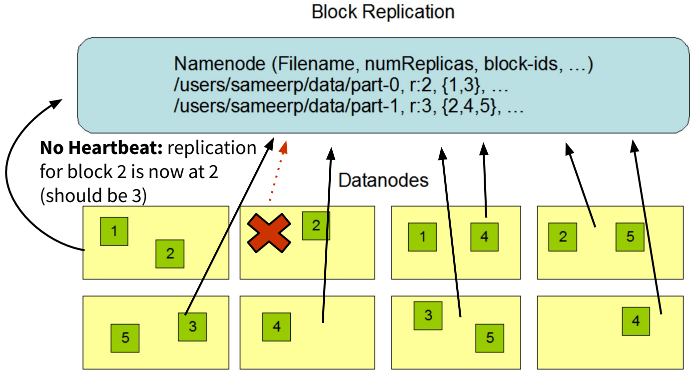
A for a single file we can tolerate **r-1** node failing **at once** before any data is lost!

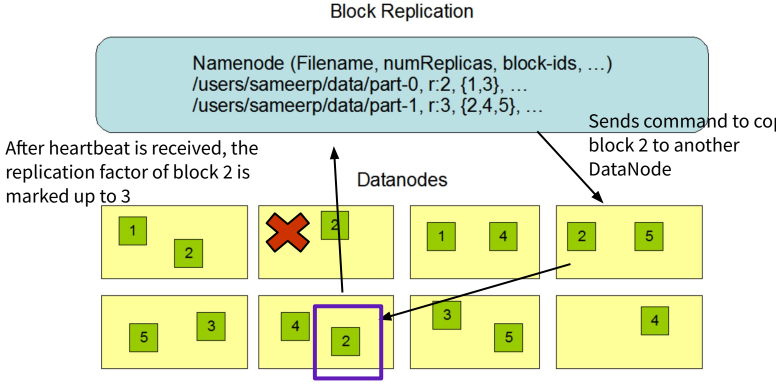


https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

How HDFS achieves fault-tolerance

- DataNodes send heartbeat messages periodically to the NameNode
    - These contain all data blocks stored in that node
- Trough heartbeats the NameNode maintains a picture of:
    - where each file (its blocks) is stored
    - The current replication of each block
- If the NameNode stops receiving heartbeats for a DataNode it marks it as failed
- When a node fails the NameNode calculate the replication of all blocks stored in that were stored in that DataNode
    - If any blocks fall under their replication factor the NameNode issues a command to create new copies in new DataNode to achieve replication





https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

TASK 1: Install Hadoop and create a HDFS file

Installation process of Hadoop is described here:
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html

This assumes you have executed:
$ wget **https://dlcdn.apache.org/hadoop/common/hadoop-3.4.0/hadoop-3.4.0.tar.gz**
$ tar -xzf hadoop-3.4.0.tar.gz
You need to have Java and the **SSH server running**

The JAVA_HOME does not need the "bin/java" part
Example: **export JAVA_HOME=/usr/**

```
(base) george@DESKTOP-E24BUDU:~/hadoop$ which java
/usr/bin/java
```

This line has to be added to:
<path_to_hadoop>/etc/hadoop/hadoop-env.sh

PLEASE FOLLOW THE INSTRUCTION FOR PSEUDO-DISTRIBUTED OPERATION

After installation run:
$ bin/hdfs namenode -format
$ sbin/start-dfs.sh

Open http://localhost:9870/ on your browser to see your node!
- Go to utilities->browse the file system to explore your HDFS

Try creating some files and directories using the following:
$ bin/hdfs dfs -command <options>

Pseudo-Distributed Operation

Hadoop can also be run on a single-node in a pseudo-distributed mode where each Hadoop daemon runs in a separate Java process.

Configuration

Use the following:

etc/hadoop/core-site.xml:

The <configuration> <\configuration> part is already there be careful not to add it twice

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

etc/hadoop/hdfs-site.xml:

The <configuration> <\configuration> part is already there be careful not to add it twice

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

Setup passphraseless ssh

Now check that you can ssh to the localhost without a passphrase:

```
$ ssh localhost
```

If you cannot ssh to localhost without a passphrase, execute the following commands:

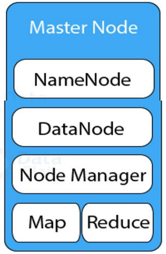
```
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

HDFS list of commands: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html

Hadoop deployments

Local mode: everything runs in a single thread!

- Great for debugging
- Stores data in the **local fs** since the DataNode, NameNode and everything else are in the same place



Pseudo-Distributed: Runs the DataNode and NameNode in separate processes

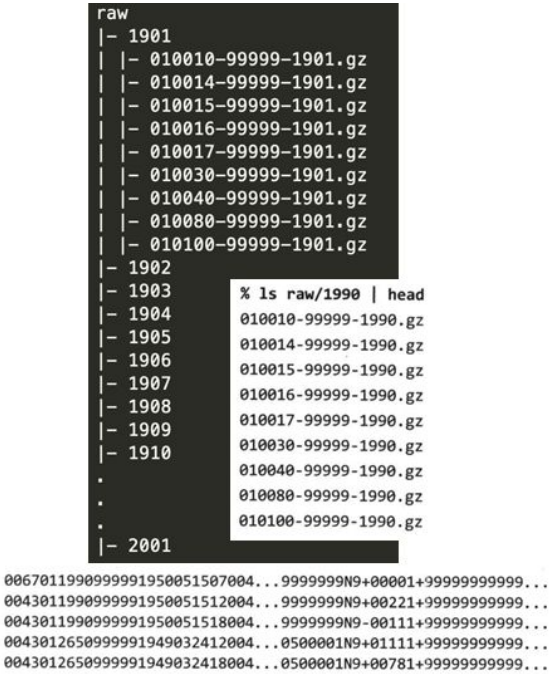
- Much closer to real deployment of hadoop

Best to use Pseudo-Distributed as the workflow is the same as if working in a huge cluster

MapReduce: Calculating max temperature

- Assume you have to process a weather dataset to find the highest temperature each year
    - Simple! Just loop over all entries while updating a few variable to keep track of highest value we have seen so far for each year
- What if that dataset has data from over 100 years…
- What about using MPI to distribute the search?
    - The root would have to send **many GB's** of data to other processes!

Solution: store everything in an HDFS cluster and utilise MapReduce!

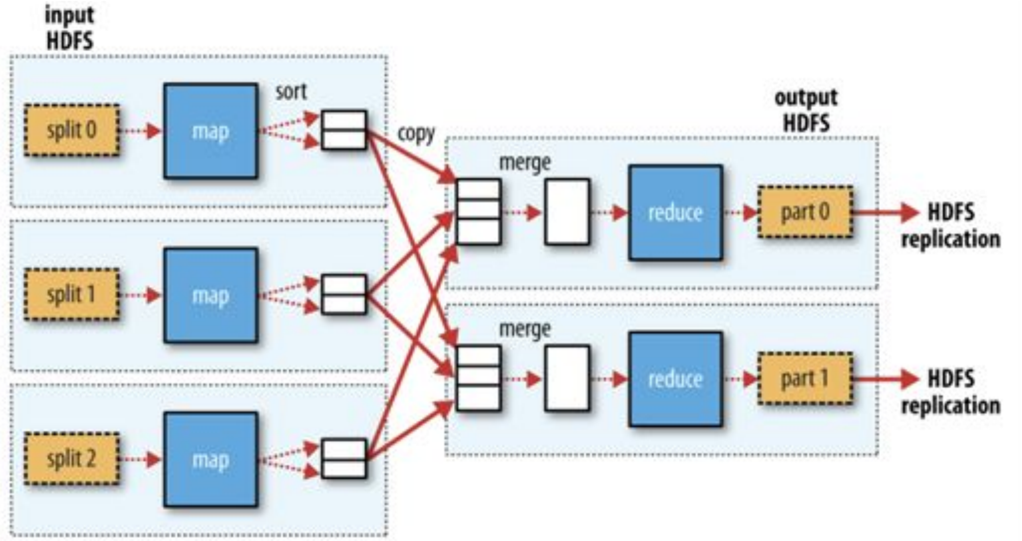


But what is MapReduce?

Moving computation is easier than moving data!

- The MapReduce paradigm only contains two steps!
  - Map and Reduce
- Assumes data are in the form of Key-Value pairs
- Map:
  - Split input to smaller blocks
  - Apply an operation
- Reduce:
  - Group results from Map by key

Takes full advantage of the distributed nature of HDFS



MapReduce: Calculating max temperature (cont.)

The Map function: extracts the year and the air temperature

The Reduce function: picks out the maximum reading of each year

REDUCES THE AMOUNT OF DATA THAT MOVES!

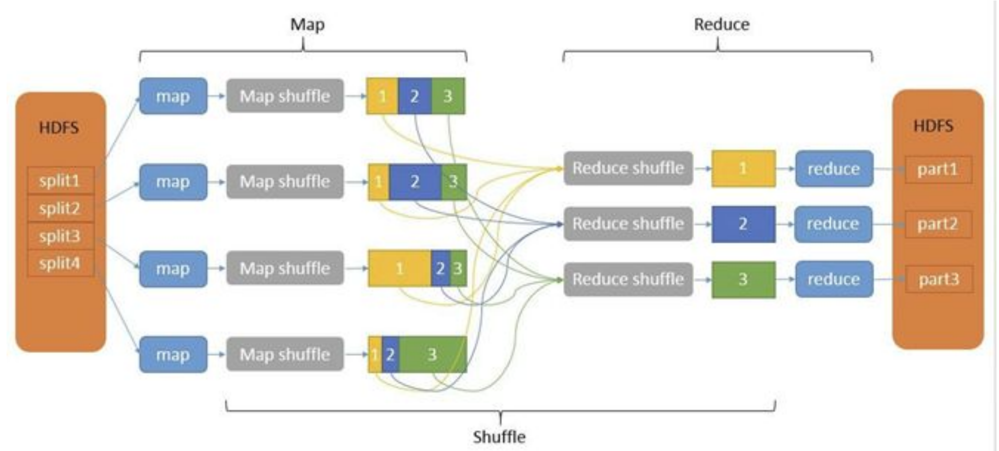
However:

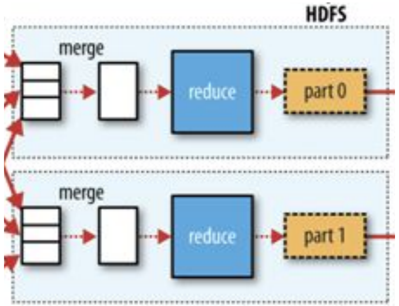
- **requires formulating the problem as operations on <key, value> items**
- **requires keys to be comparable (why?)**

MapReduce: Shuffle / Sort

Doesn't that violate the “move computation not data” idea?
I guess a bit but… map HEAVILY compresses data

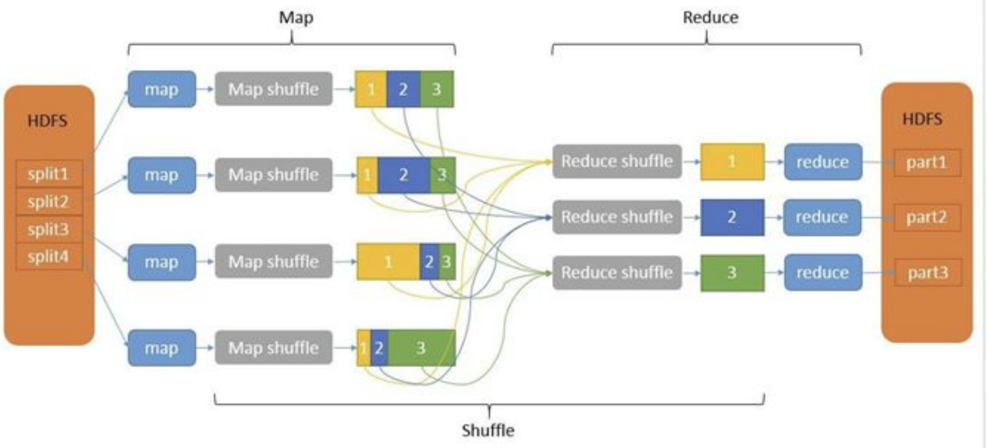
- The reduction operation ‘**reduces**’ all values associated with a key
    - List of temperature -> max temperature
- We need to give the reducers all <key,value> pairs for their associated key
- This is the goal of the sort/shuffle part and the reason keys NEED to be comparable
- The nodes that execute the reduce are
    - The same that execute the mapp
    - The same that store the original data
- **MapReduce produces as many result files (parts) as the reducers**

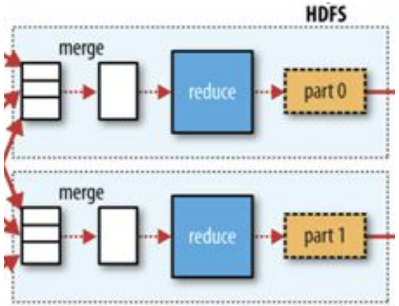




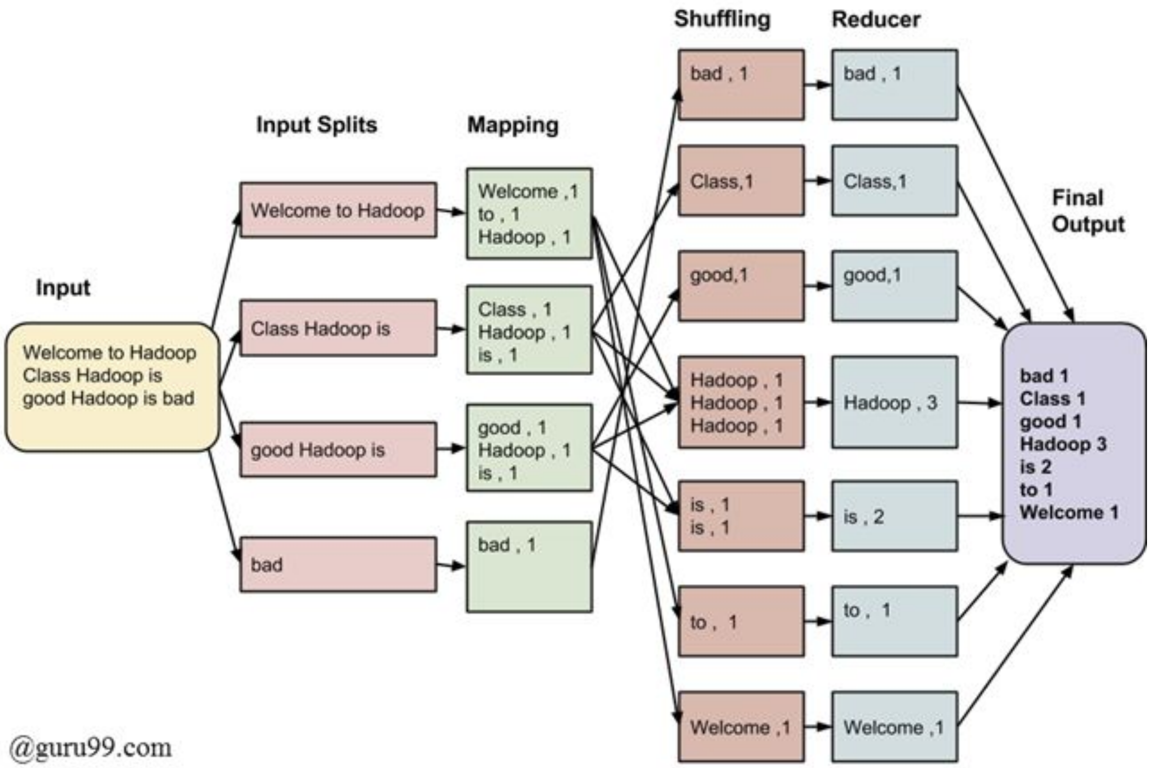
MapReduce: other nice things

- MapReduce is executed as a **job** which managed by the framework!
- The fault tolerance of HDFS extends to MapReduce!
  - If a map tasks fails on a node it will be rescheduled automatically!





A more complete example: WordCount



TASK: Implement a Word Count MapReduce programme

If you managed to set up hadoop for the previous task this tutorial has a step by step guide for WordCount!
https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Notes:

In the command $ bin/hadoop jar wc.jar WordCount /user/joe/wordcount/input /user/joe/wordcount/output

- Are HDFS directories created with $ bin/hdfs dfs …
- Hadoop **requires** its input data to be in /user/<your_user_name>/…

You need to put some data in HDFS by creating it locally and using $ bin/hdfs dfs -put <local_src> <hdfs_dest>

MapReduce CANNOT overwrite the output! If you want to re-run delete output/ or use a different name

ADDITIONAL CHALLENGES: Edit the source code to

- Only display words that appear 2 or more times
- Only count specific words (e.g ‘hello’ and ‘goodbye’)

TASK: Implement a Word Count MapReduce programme

If you cannot set up hadoop you can try using the docker image with a **LOCAL NODE**

- This uses the local file system so no need to use **/bin/hdfs dfs** etc…

$ docker run -it --rm apache/hadoop:3.4 /bin/bash # this should launch an container with a local node of hadoop

This container does not have javac!
You have to use the prebuilt example jar that implement a “grep” MapReduce

```
$ mkdir input
$ cp etc/hadoop/*.xml input
$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.4.1.jar grep input output 'dfs[a-z.]+'
$ cat output/*
```

Try changing the above commands so that they count the words in a file you have created (hint: *+)

Try counting specific words only!

MapReduce CANNOT overwrite the output! If you want to re-run delete output/ or use a different name **(still valid)**

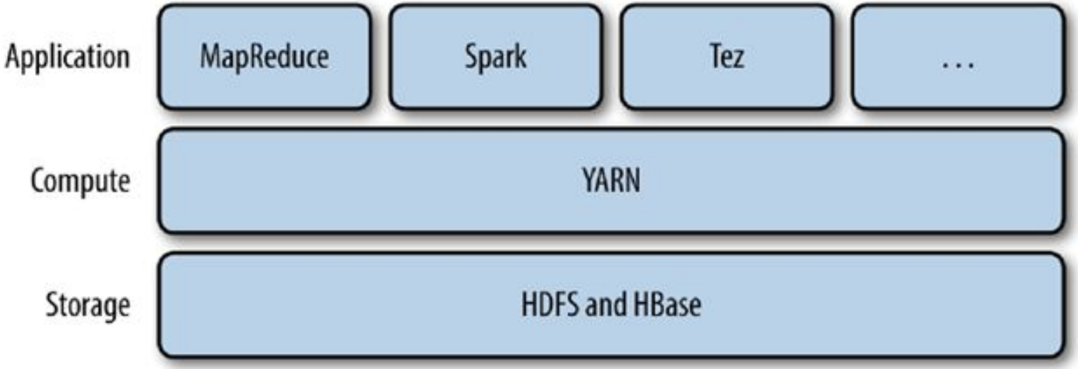
Code from “standalone” section:
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html

YARN: Brief introduction

YARN hides resource management details from the users

Some distributed computing frameworks (MapReduce, Spark, and so on) run as YARN jobs - letting YARN negotiate and provide resources for their execution from the cluster.

Allows many users to share the cluster without worrying about managing their resources



YARN: Brief introduction

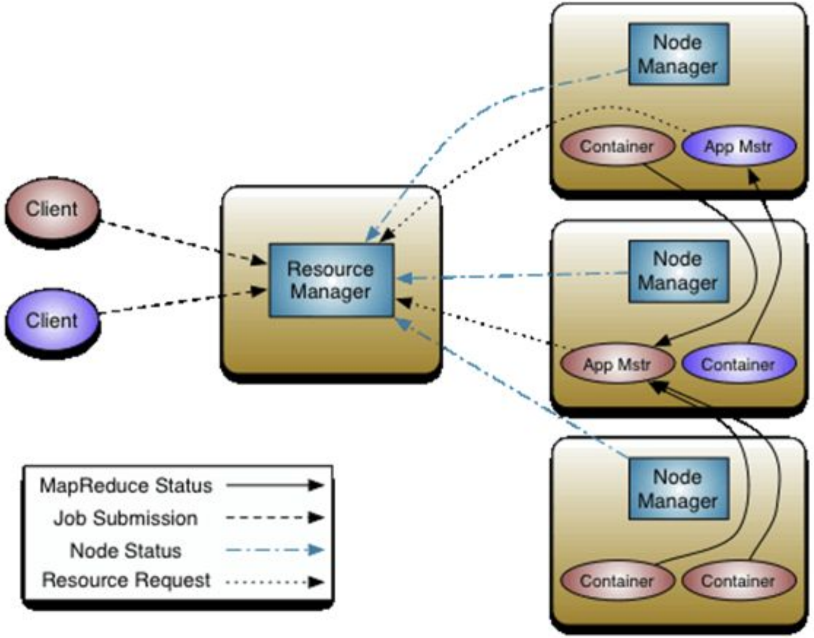
YARN CONSISTS OF

ResourceManager: One per cluster

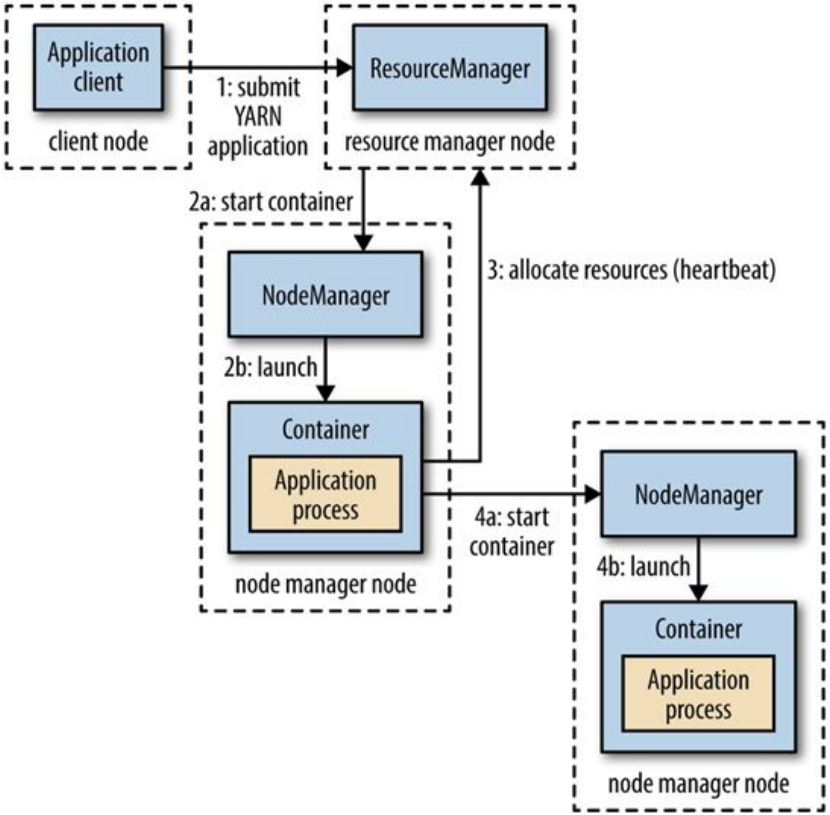
- The central controller, allocate resources, submitting & managing the jobs

NodeManager: per-machine agent

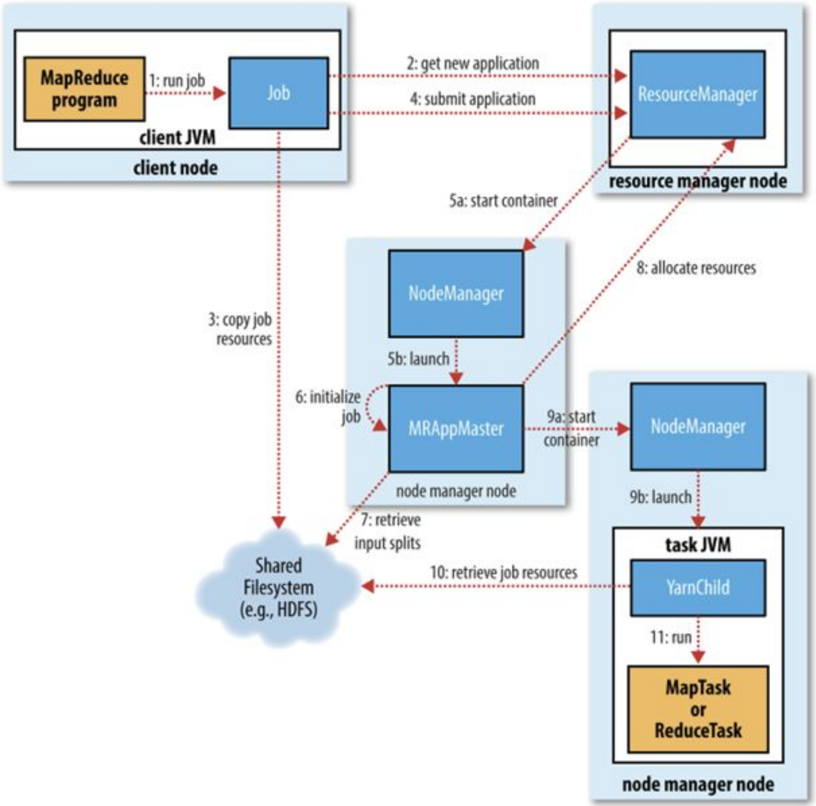
- Monitors containers that execute the application-specific process with a constrained set of resources (CPU, memory, disk, network).
- Reports the resource usage to the ResourceManager/Scheduler.



YARN in detail: Architecture and MapReduce example



YARN High Level Architecture



How Hadoop runs MapReduce