BDM1213 Data Encoding Principles

Week 04: Word Count Program and YARN

Dr. James Hong

Since MapReduce framework is based on Java, you might be wondering how a developer can work on it if he/ she does not have experience in Java. Well, developers can write mapper/Reducer application using their preferred language and without having much knowledge of Java, using *Hadoop Streaming* rather than switching to new tools or technologies like Pig and Hive.

What is Hadoop Streaming?

Hadoop Streaming is a utility that comes with the Hadoop distribution. It can be used to execute programs for big data analysis. Hadoop streaming can be performed using languages like Python, Java, PHP, Scala, Perl, UNIX, and many more. The utility allows us to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer.

Hadoop word count in Python

```python
#!/usr/bin/env python
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

We will simply use Python's sys.stdin to read input data and print our own output to sys.stdout.

```python
#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Test

```
# Test mapper.py and reducer.py locally first

# very basic test
hduser@ubuntu:~$ echo "foo foo quux labs foo bar quux" | /home/hduser/mapper.py
foo     1
foo     1
quux    1
labs    1
foo     1
bar     1
quux    1

hduser@ubuntu:~$ echo "foo foo quux labs foo bar quux" | /home/hduser/mapper.py | sort -
bar     1
foo     3
labs    1
quux    2

# using one of the ebooks as example input
# (see below on where to get the ebooks)
hduser@ubuntu:~$ cat /tmp/gutenberg/20417-8.txt | /home/hduser/mapper.py
 The     1
 Project 1
 Gutenberg       1
 EBook   1
 of      1
 [...]
 (you get the idea)
```

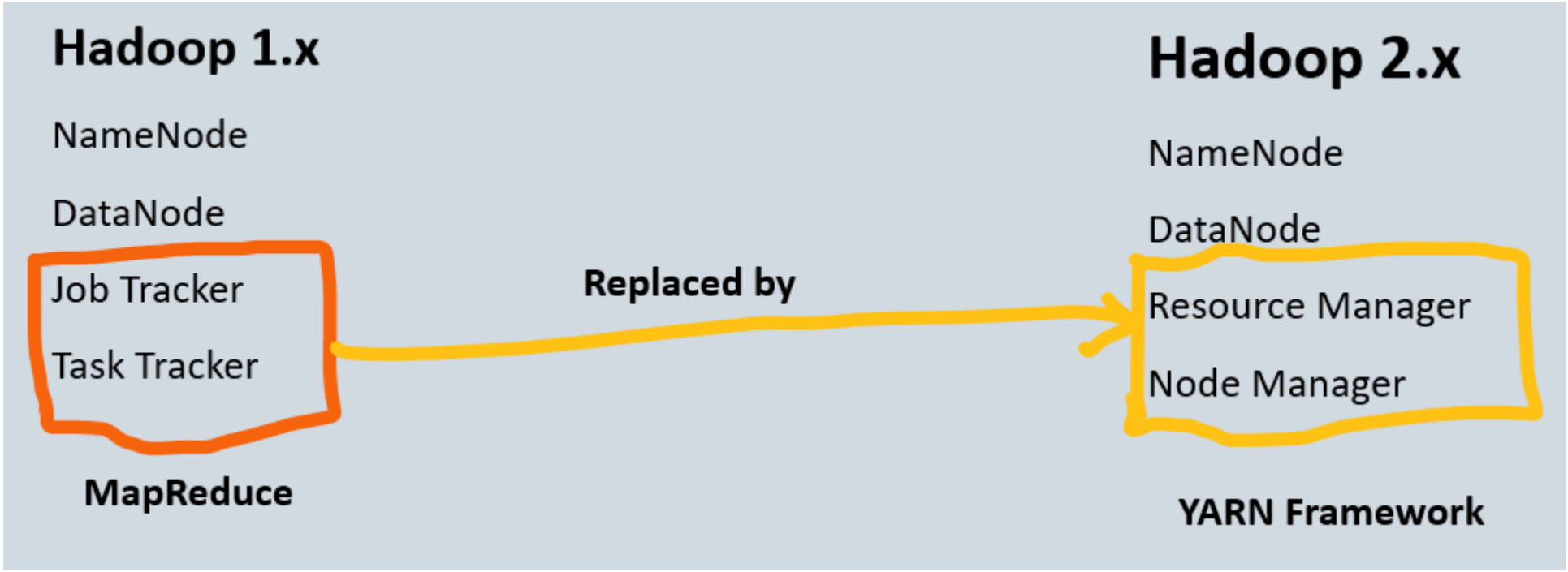
Review of HDFS

- Storing Big Data was a problem due to it's massive volume.
- The solution was to use HDFS, storing Big Data was possible using HDFS. In the traditional approach, all the data was stored in a single central database.
- With the rise of big data, a single database was not enough for storage.
- The solution was to use a distributed approach to store the massive amount of data. Data was divided and distributed amongst many individual databases.
- Hadoop Distributed File System (HDFS) is a specially designed file system for storing huge dataset in commodity hardware.
- HDFS splits massive files into small chunks, these chunks are known as data blocks. The default size of one data block is 128 MB. This was HDFS. Hadoop 1.0 was also known as MapReduce Version 1.

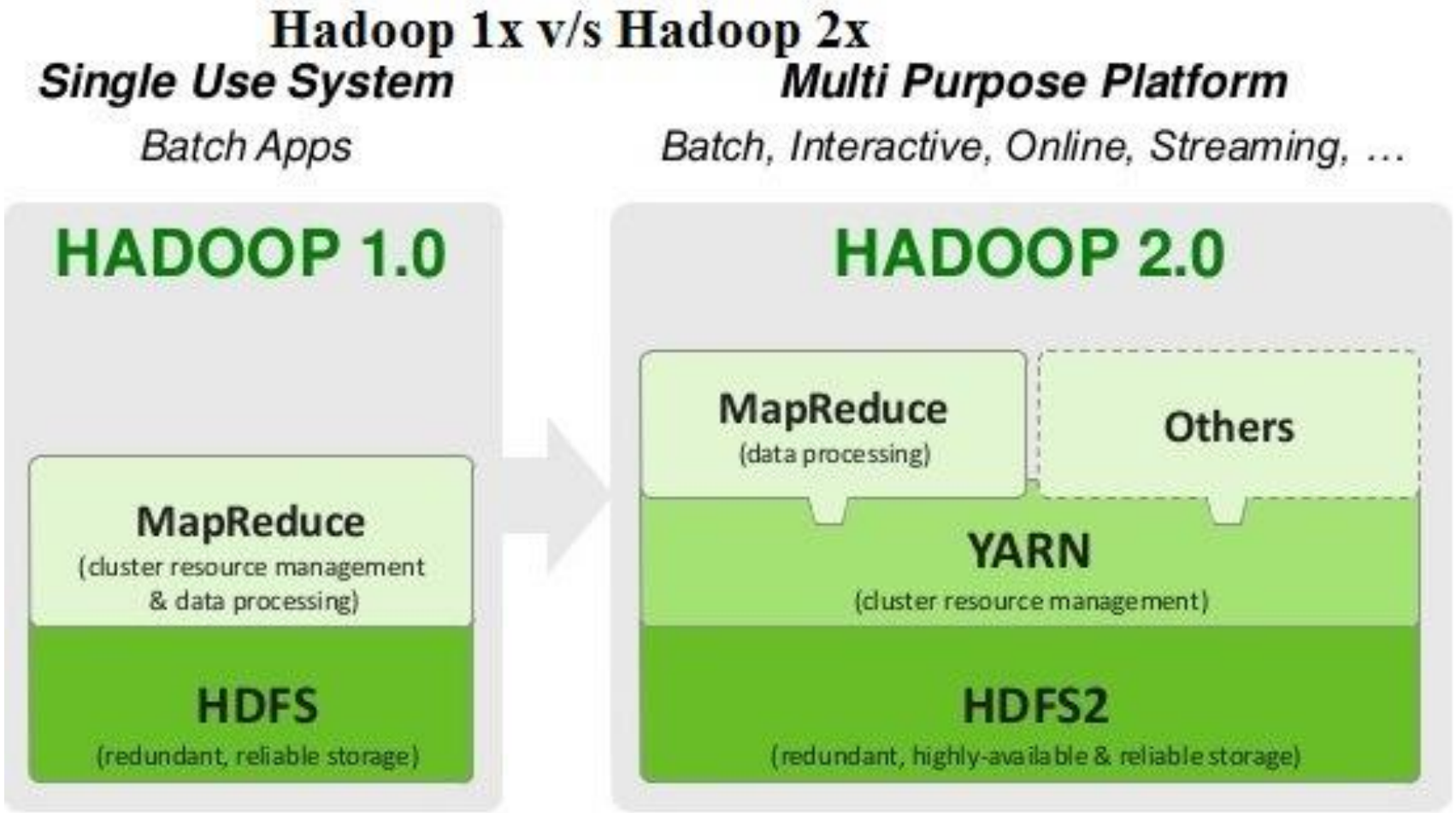
Bottleneck of Hadoop v1

- The disadvantage with this version was that the Job tracker did both the processing of data and resource allocation.
- As a result, Job tracker was overburdened due to handling, job scheduling and resource management.
- To overcome this issue, Hadoop 2 introduced YARN as the processing layer.
- Yet Another Resource Negotiator (YARN) acts as the resource management unit of Hadoop.
- Few features of YARN are, Job scheduling, Multitenancy, and Scalability.

Hadoop v1 vs v2



Hadoop v1 vs v2



YARN components

YARN enabled the users to perform operations as per requirement by using a variety of tools like ***Spark*** for real-time processing, ***Hive*** for SQL, ***HBase*** for NoSQL and others.

Apart from Resource Management, YARN also performs Job Scheduling. YARN performs all your processing activities by allocating resources and scheduling tasks. Apache Hadoop YARN Architecture consists of the following main components :

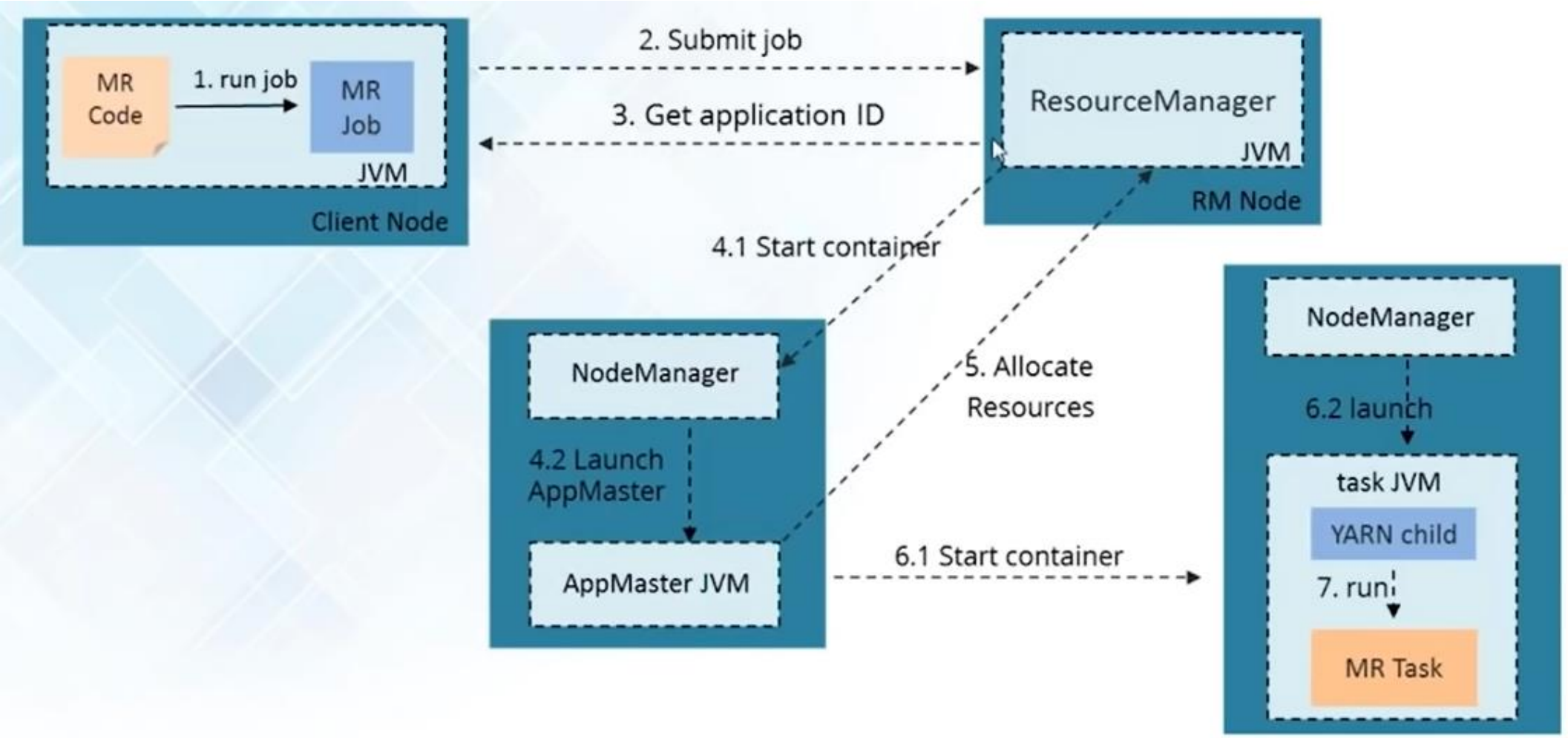
1.Resource Manager: Runs on a master daemon and manages the resource allocation in the cluster.

2.Node Manager: They run on the slave daemons and are responsible for the execution of a task on every single Data Node.

3.Application Master: Manages the user job lifecycle and resource needs of individual applications. It works along with the Node Manager and monitors the execution of tasks.

4.Container: Package of resources including RAM, CPU, Network, HDD etc on a single node.

Hadoop 2 Process



Install Hadoop!