CMSC 691
High Performance Distributed Systems

Apache Hadoop

Dr. Alberto Cano
Assistant Professor
Department of Computer Science
acano@vcu.edu

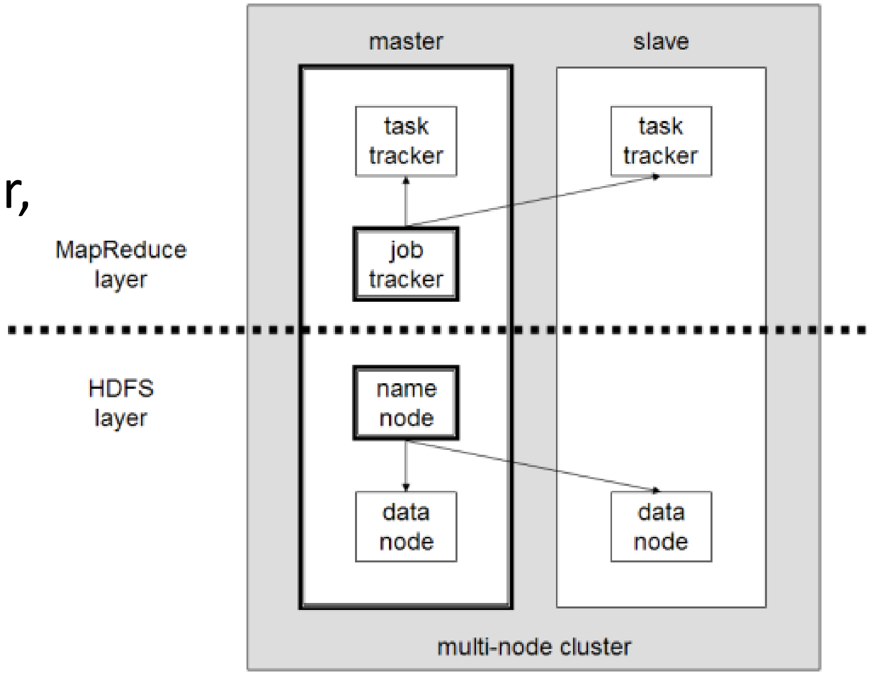
Apache Hadoop

- Open-source framework for distributed storage and computing of big data sets on clusters
- Hadoop Distributed File System (HDFS): fault-tolerant, high-bandwidth, high availability distributed storage
- MapReduce: distributed big data processing infrastructure (abstract/paradigm, fault-tolerant, schedule, execution)
- Data locality: processing data local to each compute node i.e. “don't move data to workers, move workers to the data”
- Assumptions: commodity hardware is inexpensive but fail all the time, moderate number of huge write-once-read-only data files

Apache Hadoop architecture

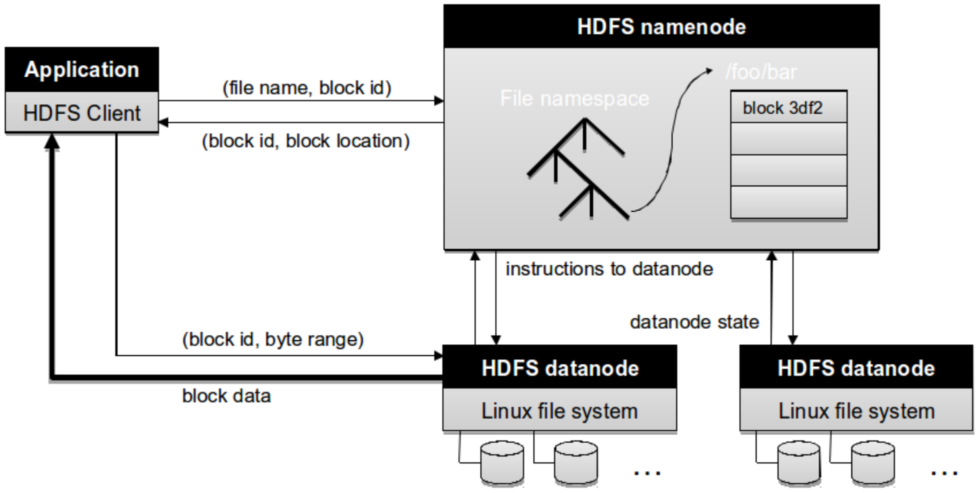
- Hadoop Common: OS level abstractions
- Hadoop Distributed File System (HDFS)
- MapReduce engine

- Master: Job Tracker, Task Tracker, NameNode, and DataNode
- Slave/Workers: DataNode and Task Tracker



HDFS

- Single namespace for entire cluster
- NameNode: Maps a file to a file-id and list of DataNodes
- DataNode: Maps a block-id to a physical location on disk
- Files are broken up into blocks of 64 MB
- Data coherency (write-once-read-many)
- Data replication 3



HDFS NameNode

- Managing the file system namespace:
  - Holds file/directory structure, metadata, file-to-block mapping, access permissions, etc.
- Coordinating file operations:
  - Directs clients to DataNode for reads and writes
  - No data is moved through the NameNode
- Maintaining overall health:
  - Periodic communication with the DataNode
  - Block re-replication and rebalancing

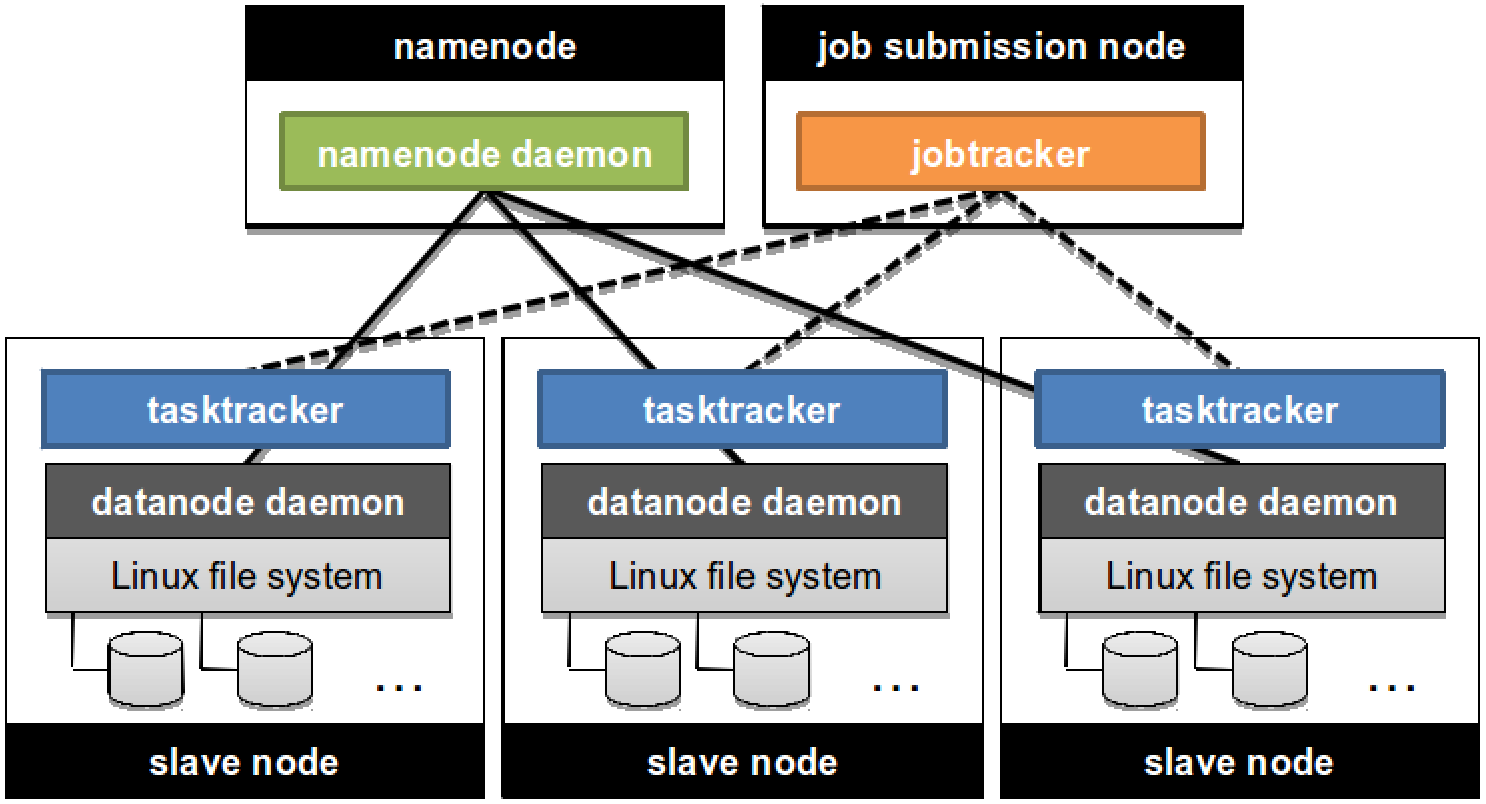
HDFS DataNode

- A Block Server
  - Stores data in the local file system
  - Stores meta-data of a block (checksums)
  - Serves data and meta-data to clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode
  - Facilitates Pipelining of Data
- Forwards data to other specified DataNodes

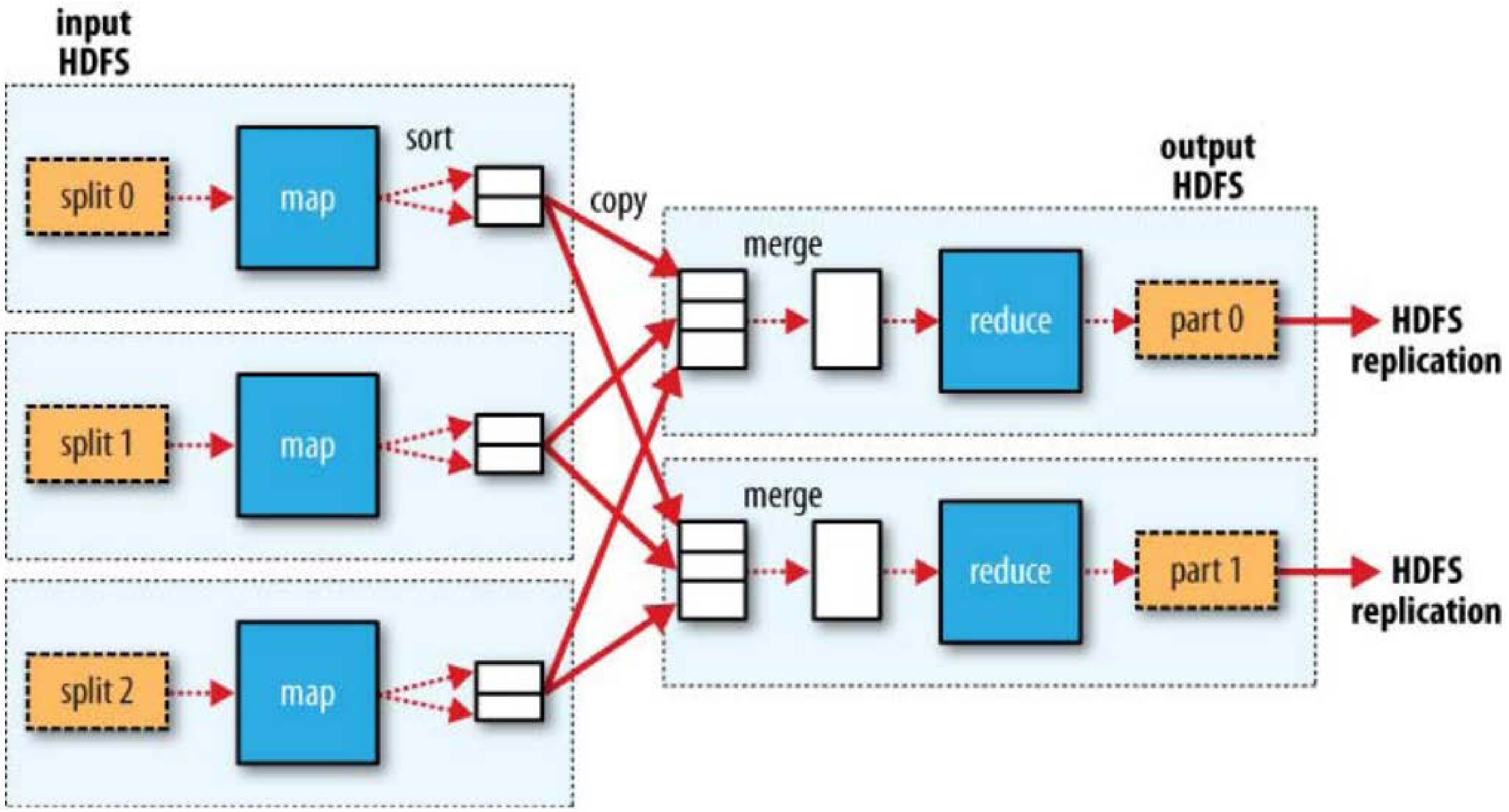
HDFS block placement

- Current strategy: default replication 3
    - One replica on one node in the local rack
    - Second replica on one node in a remote rack
    - Third replica on another node in the same remote rack
    - Additional replicas are randomly placed
- Clients read from nearest replica
- Data correctness validated through checksums
- Fault-tolerance: NameNode is a single point of failure!

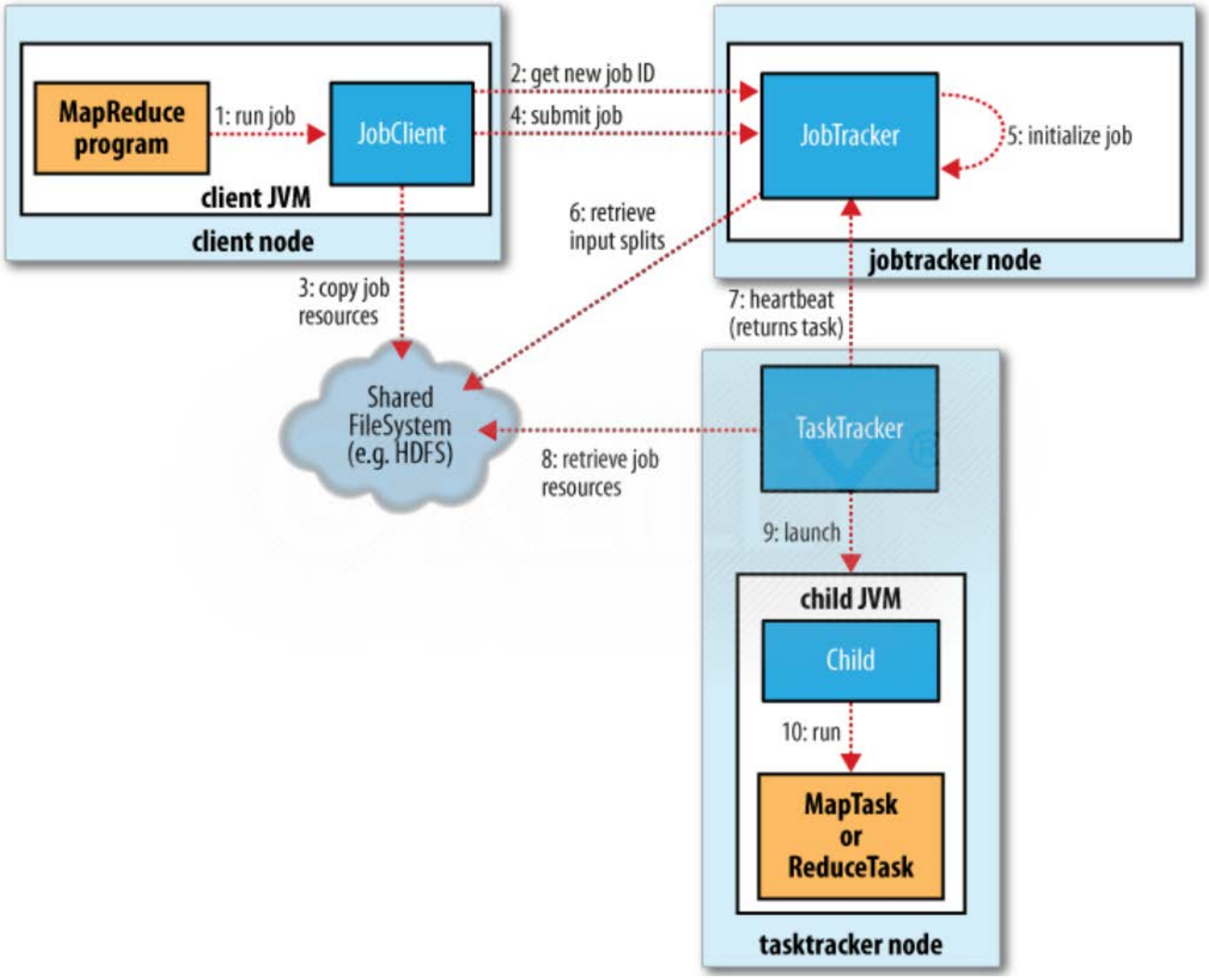
Apache Hadoop architecture



Apache Hadoop MapReduce data flow



Apache Hadoop MapReduce data flow



Apache Hadoop Installation

- Install software dependencies

```
$ sudo apt-get install maven git ssh rsync default-jdk default-jre openssh-server
```

- Download and install Eclipse Neon from https://eclipse.org/downloads/
- Download Apache Hadoop 2.7.3 from http://hadoop.apache.org/releases.html (binary) and extract in a folder e.g. /home/user/hadoop-2.7.3
- Edit line 25 of the file /home/user/hadoop-2.7.3/etc/hadoop/hadoop-env.sh

```
export JAVA_HOME=/usr/lib/jvm/default-java
```

- Edit your /home/user/.bashrc and append to the end:

```
export PATH=$PATH:/home/user/hadoop-2.7.3/bin
```

- Close the terminal and reopen it

Set up a new Eclipse project for Hadoop using Maven

- New -> Project -> Maven -> Maven Project
- Click on Next (default) two times
- Write a Group Id and Artifact Id, e.g. Hadoop and MapReduce
- Edit the pom.xml (right click, open with Text Editor) and add a new dependency within the dependencies section

```
<dependency>
		<groupId>org.apache.hadoop</groupId>
		<artifactId>hadoop-core</artifactId>
		<version>1.2.1</version>
</dependency>
```

- Delete the App.java and copy & paste the WordCount.java in the src/main/java as a baseline for your Hadoop MapReduce project

Compile and run a Eclipse project for Hadoop using Maven

- Right click on the project name -> Run As -> Maven install
- This will download dependencies, compile the code and package a jar e.g. in target/MapReduce-0.0.1-SNAPSHOT.jar
- Run Hadoop e.g. using the WordCount as:

```
hadoop jar target/MapReduce-0.0.1-SNAPSHOT.jar Hadoop.MapReduce.WordCount
example/textfile.txt output
```

which states for:

```
hadoop jar jarfile.jar mainclass inputdata outputdata
```

- Analyze the files in the output
- Check out the online courses available in blackboard!

CMSC 691
High Performance Distributed Systems

Apache Hadoop

Dr. Alberto Cano
Assistant Professor
Department of Computer Science
acano@vcu.edu