Hadoop & HDFS

INF 551

Wensheng Wu

Hadoop

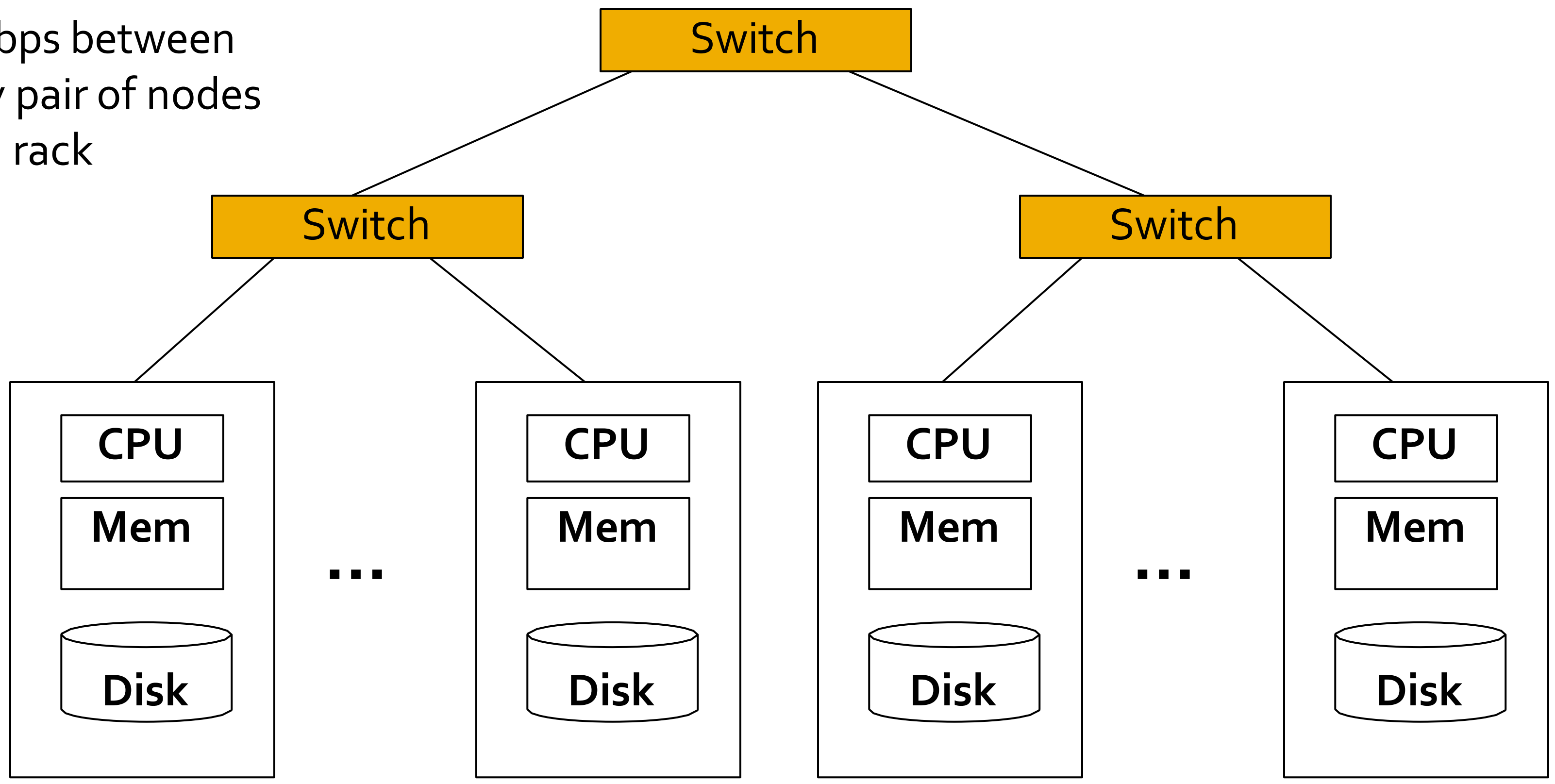
- A large-scale distributed batch-processing infrastructure

- Large-scale:
  - Handle a large amount of data and computation
- Distributed:
  - Distribute data & work across a number of machines
- Batch processing
  - Process a series of jobs without human intervention

Cluster Architecture

2-10 Gbps backbone between racks

1 Gbps between
any pair of nodes
in a rack



Each rack contains 16-64 nodes

In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

History

- 1st version released by Yahoo! in 2006
  - named after an elephant toy

- Originated from Google's work
  - GFS: Google File System (2003)
  - MapReduce (2004)

Roadmap

- Hadoop architecture
  - HDFS
  - MapReduce
- Installing Hadoop & HDFS

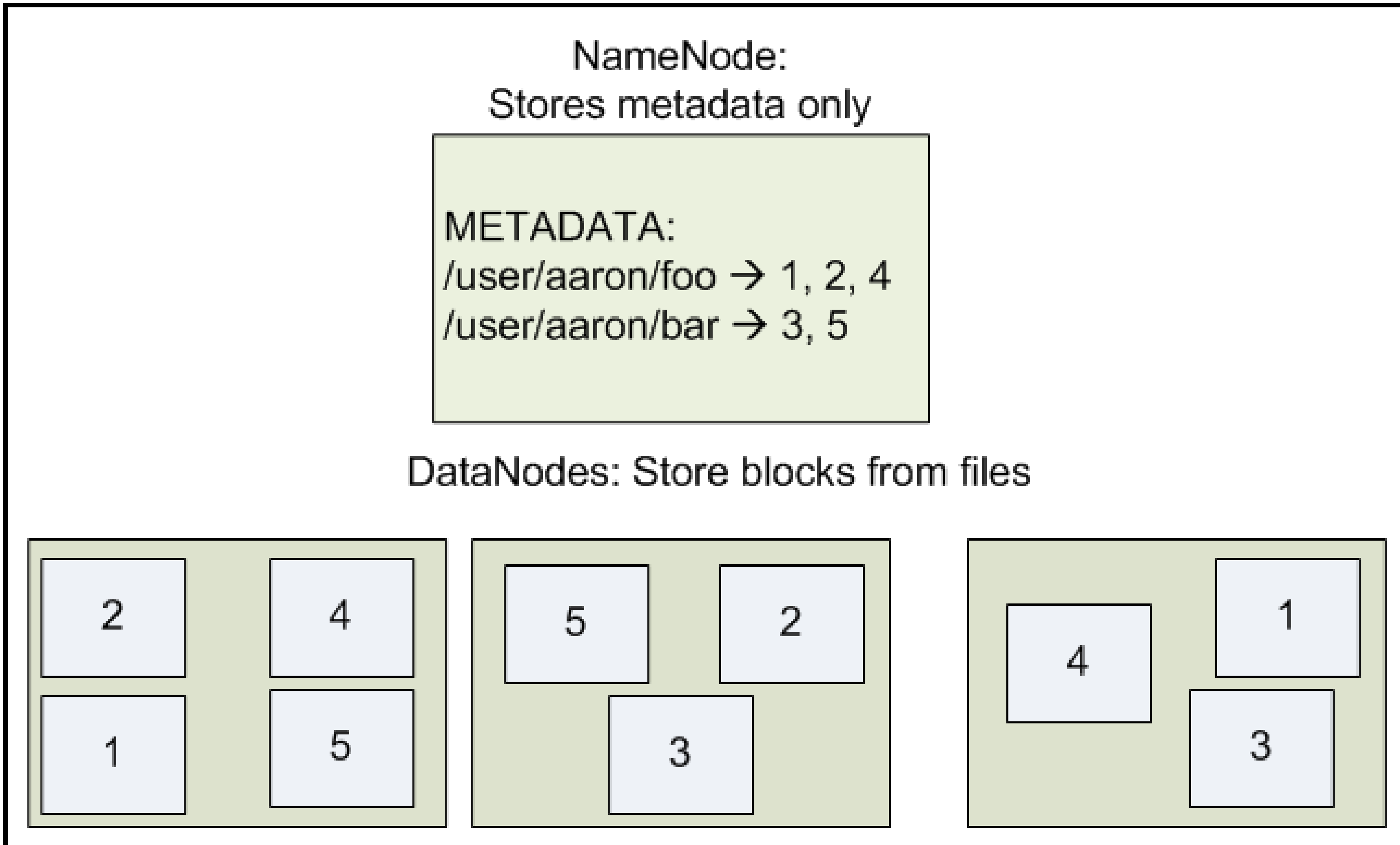
Key components

- HDFS (Hadoop distributed file system)
  - Distributed data storage with high reliability
- MapReduce
  - A parallel, distributed computational paradigm
  - With a simplified programming model

HDFS

- Data are distributed among multiple data nodes
  - Data nodes may be added on demand for more storage space
- Data are replicated to cope with node failure
  - Typically replication factor = 2/3
- Requests can go to any replica
  - Removing the bottleneck (in single file server)

HDFS architecture



HDFS has …

- A single NameNode, storing meta data:
  - A hierarchy of directories and files
  - Attributes of directories and files
  - Mapping of files to blocks on data nodes

- A number of DataNode:
  - Storing contents/blocks of files

HDFS also has …

- A SecondaryNameNode
  - Maintaining checkpoints of NameNode
  - For recovery

- In a single-machine setup
  - all nodes correspond to the same machine

Metadata in NameNode

- NameNode has an inode for each file and dir
- Record attributes of file/dir such as
  - Permission
  - Access time
  - Modification time
- Also record mapping of files to blocks

Mapping information in NameNode

- E.g., file /user/aaron/foo consists of blocks 1, 2, and 4

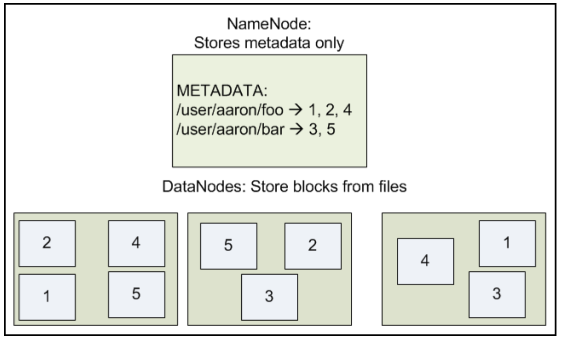
- Block 1 is stored on data nodes 1 and 3
- Block 2 is stored on data nodes 1 and 2
- …

Block size

- HDFS: 64MB
  - Much larger than disk block size (4KB)

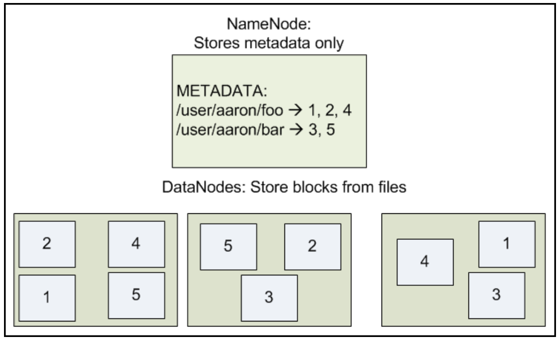
- Why larger size in HDFS?
  - Reduce metadata required per file
  - Fast streaming read of data (since larger amount of data are sequentially laid out on disk)
  - Good for workload with largely sequential read of large file

Reading a file



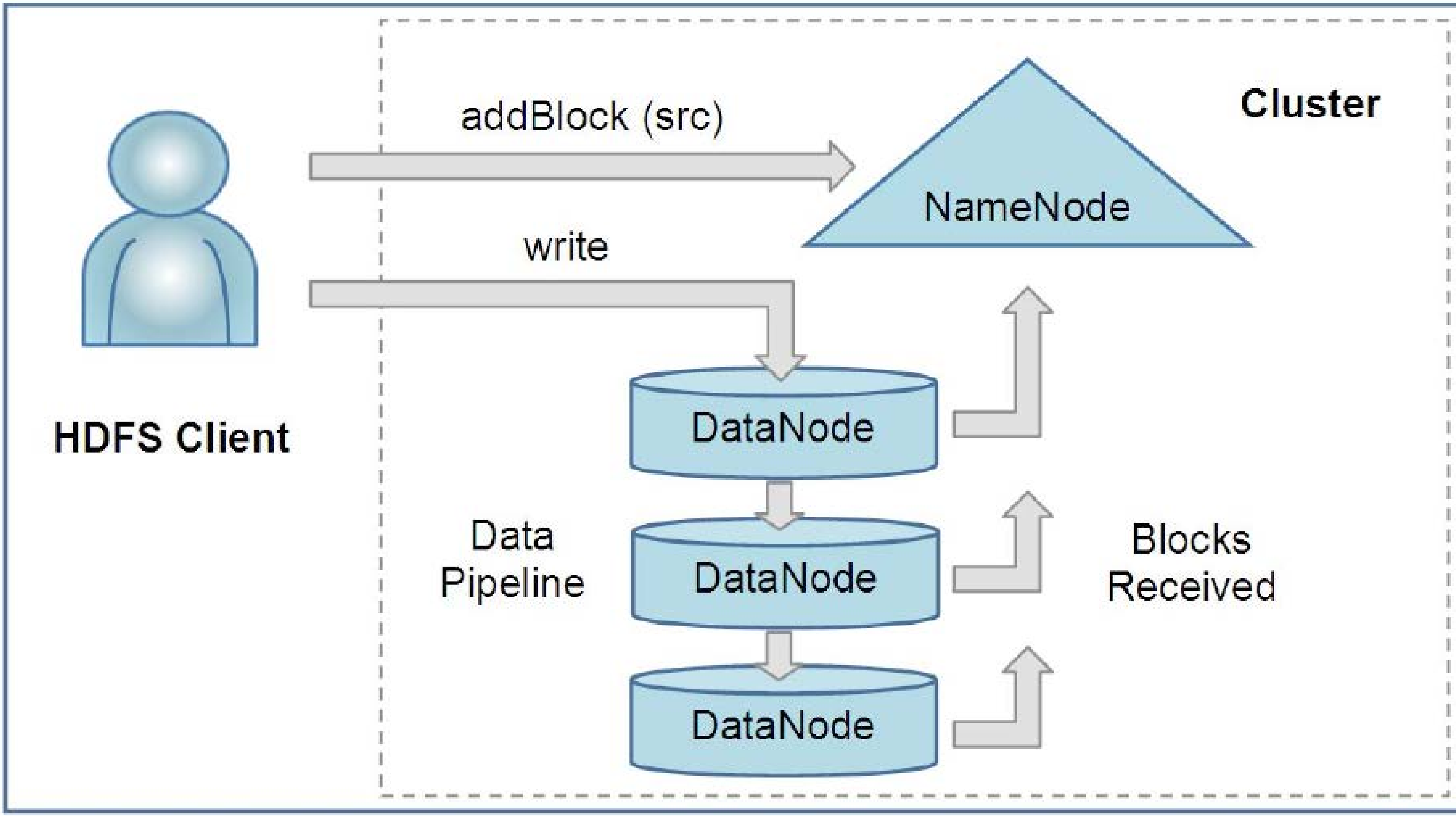
- Client first contacts NameNode
- NameNode informs the client:
  - the closest DataNodes storing blocks of the file
- Client contacts the DataNodes directly
  - For reading the blocks

Writing a file



- Blocks are written one at a time
  - In a pipelined fashion through the data nodes
- For each block:
  - Client asks NameNode to select DataNodes for holding its replica
  - e.g., DataNodes 1 and 3 for the first block of /user/aaron/foo
  - It then forms the pipeline to send the block

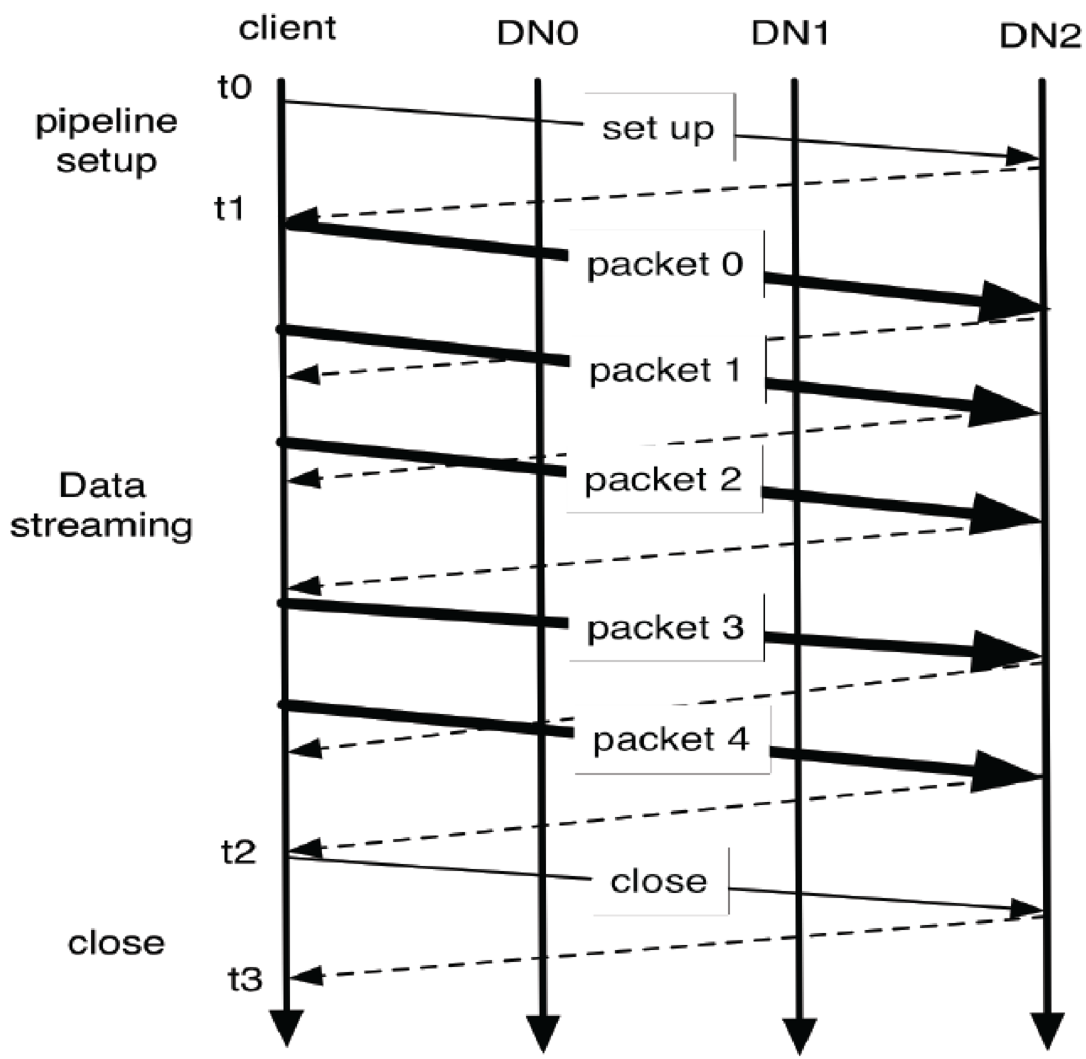
Writing a file



Data pipelining

- Blocks are divided into packets (64KB)
- Packets are sent over the pipeline
- Next packet is sent before previous one is acknowledged

Data pipelining during block writing



Roadmap

- Hadoop architecture
  - HDFS
  - MapReduce

- Installing Hadoop & HDFS

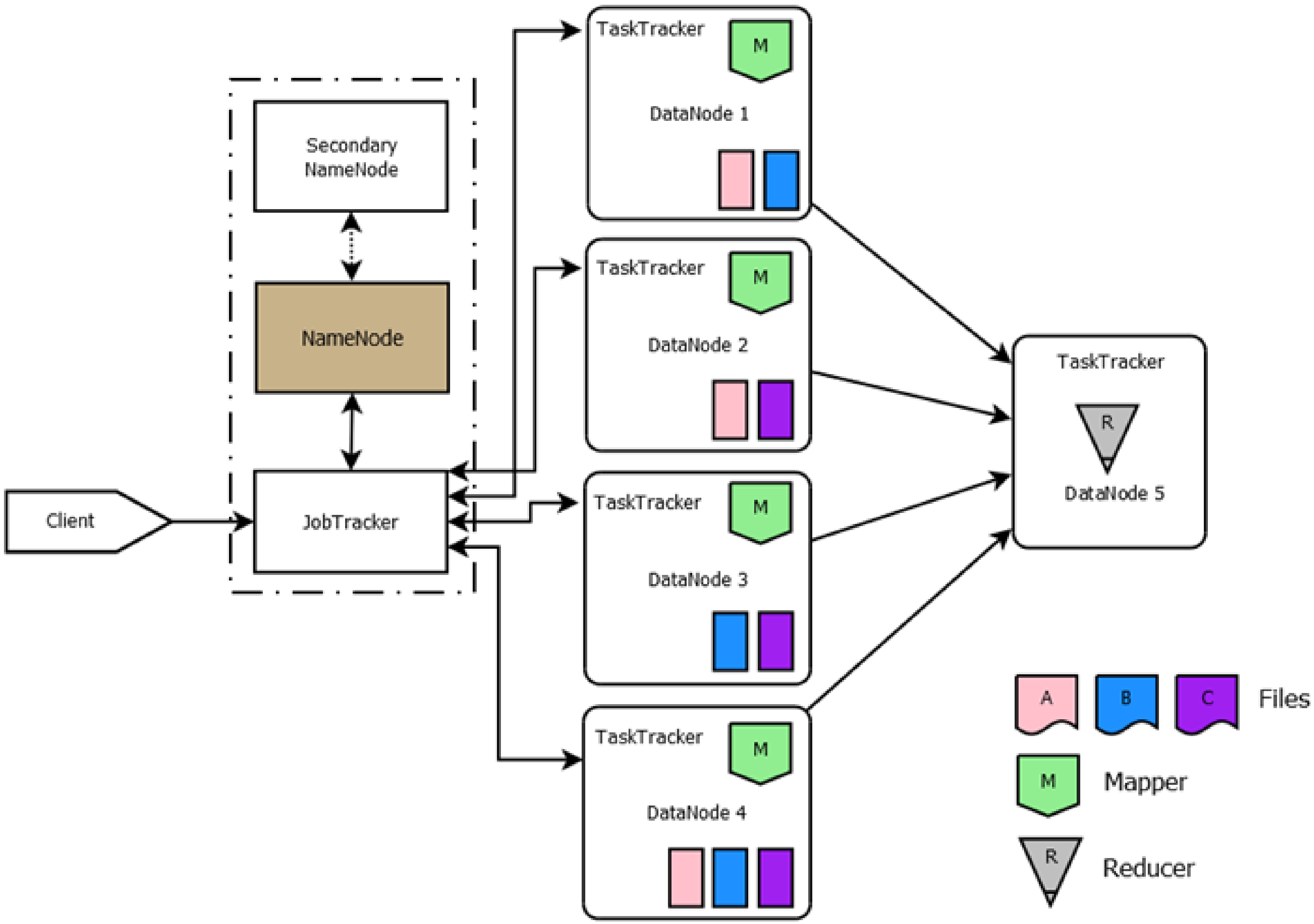
MapReduce job

- A MapReduce job consists of a number of
  - Map tasks (mappers)
  - Reduce tasks (reducers)
  - (Internally) shuffle tasks

Map, reduce, and shuffle tasks

- Map task performs data transformation
- Reduce task combines results of map tasks
- Shuffle task sends output of mappers to right reducers

Hadoop cluster



Job tracker

- Takes requests from clients (MapReduce programs)
- Ask name node for location of data
- Assign tasks to task trackers near the data
- Reassign tasks if failed

Task tracker

- Accept (map, reduce, shuffle) tasks from job trackers

- Send heart beats to job trackers: I am alive

- Monitor status of tasks and notify job tracker

Roots in functional programming

- Functional programming languages:
  - Python, Lisp (list processor), Scheme, Erlang, Haskell

- Two functions:
  - Map: mapping a list => list
  - Reduce: reducing a list => value

- map() and reduce() in Python
  - https://docs.python.org/2/library/functions.html#map

map() and reduce() in Python

- list = [1, 2, 3]
- def sqr(x): return x ** 2
- list1 = map(sqr, list)

What are the value of list1 and z?

- def add(x, y): return x + y
- z = reduce(add, list)

Lambda function

- Anonymous function (not bound to a name)

- list = [1, 2, 3]

- list1 = map(lambda x: x ** 2, list)
- z = reduce(lambda x, y: x + y, list)

How is reduce() in Python evaluated?

- z = reduce(f, list) where f is add function

- Initially, z is set to list[0]
- Next, repeat z = add(z, list[i]) for each i > 0
- Return final z

- Example: z = reduce(add, [1, 2, 3])
  - i = 0, z = 1; i = 1, z = 3; i = 2, z = 6

MapReduce

- Map function:
  - Input: <k, v> pair
  - Output: a list of <k', v'> pairs
- Reduce function:
  - Input: <k', list of v's> (note k's are output by map)
  - Output: a list of <k'', v''> pairs

Roadmap

- Hadoop architecture
  - HDFS
  - MapReduce

- Installing Hadoop & HDFS

Hadoop installation

- Install the Hadoop package
  - Log into your EC2 instance
  - wget http://apache.mirrors.pair.com/hadoop/common/hadoop-2.7.3/hadoop-2.7.3.tar.gz
  - gunzip hadoop-2.7.3.tar.gz
  - tar xvf hadoop-2.7.3.tar

Install java sdk

- sudo yum install java-devel

JAVA_HOME

- Edit etc/hadoop/hadoop-env.sh
  - Comment out the following line:
    - #export JAVA_HOME=${JAVA_HOME}
  - Add this:
    - export JAVA_HOME=/usr/lib/jvm/java

Setup environment variables

- Edit ~/.bashrc by adding the following:
  - export JAVA_HOME=/usr/lib/jvm/java
  - export PATH=${JAVA_HOME}/bin:${PATH}
  - export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar

- Logout and login again to your EC2
  - So that the new variables are in effect

Set up pseudo-distributed mode

- Edit etc/hadoop/core-site.xml by adding this:
  - <configuration>

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

</configuration>

- hdfs://localhost:9000 will be the URI for root of hdfs

Pseudo-distributed mode

- Edit etc/hadoop/hdfs-site.xml, add this:
  - <configuration>

    <property>

    <name>dfs.replication</name>
    <value>1</value>

    </property>
  - </configuration>
- dfs.replication = 1 (replication factor)

Setup passphraseless ssh

- To permit DataNode to access NameNode
  - Need to store public key on DataNode

- DataNode is localhost in our setup
  - All daemons run on localhost

Setup passphraseless ssh

- ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
  - This generates public/private key pairs
  - Id_rsa has the private/ id_rsa.pub has public key
- cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
  - Add public key into the list of authorized keys
- chmod 0600 ~/.ssh/authorized_keys
  - Change the file permissoin properly

Check if it works

- ssh localhost
  - It should login to localhost without asking for password
- exit
  - Make sure you exit from this login

```
[ec2-user@ip-172-31-24-7 hadoop]$ ssh localhost
Last login: Sat Jan 28 19:14:57 2017 from 75-140-79-227.dhcp.mtpk.ca.cha
rter.com

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/
[ec2-user@ip-172-31-24-7 ~]$ |
```

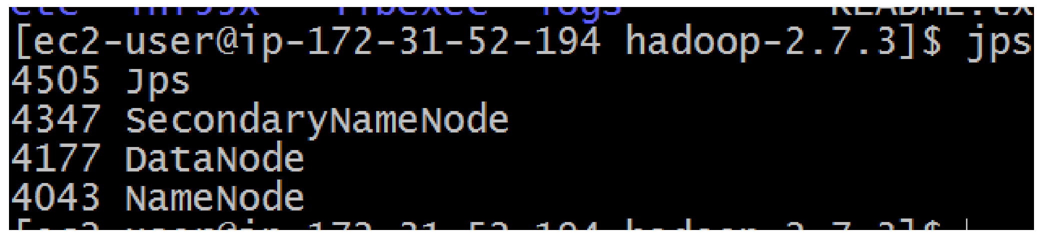
Formatting hdfs & starting hdfs

- bin/hdfs namenode -format
- sbin/start-dfs.sh
  - sbin/stop-dfs.sh to stop it

```
[ec2-user@ip-172-31-52-194 hadoop-2.7.3]$ sbin/start-dfs.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/ec2-user/hadoop-2.7.3/log
s/hadoop-ec2-user-namenode-ip-172-31-52-194.out
localhost: starting datanode, logging to /home/ec2-user/hadoop-2.7.3/log
s/hadoop-ec2-user-datanode-ip-172-31-52-194.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/ec2-user/hadoop-2.
7.3/logs/hadoop-ec2-user-secondarynamenode-ip-172-31-52-194.out
[ec2-user@ip-172-31-52-194 hadoop-2.7.3]$ jps
30298 DataNode
30164 NameNode
30468 SecondaryNameNode
30577 Jps
[ec2-user@ip-172-31-52-194 hadoop-2.7.3]$
```

Verifying HDFS is started properly

- Execute jps, you should see 3 java processes:
  - SecondaryNameNode
  - DataNode
  - NameNode



- If NameNode is not started
  - stop hdfs & reformat namenode (see previous slide)

Where is hdfs located?

- /tmp/hadoop-ec2-user/dfs/

```
[ec2-user@ip-172-31-52-194 data]$ pwd
/tmp/hadoop-ec2-user/dfs/data
[ec2-user@ip-172-31-52-194 data]$ cd ..
[ec2-user@ip-172-31-52-194 dfs]$ ls
data  name  namesecondary
[ec2-user@ip-172-31-52-194 dfs]$ ls data
current  in_use.lock
[ec2-user@ip-172-31-52-194 dfs]$ ls name
current  in_use.lock
[ec2-user@ip-172-31-52-194 dfs]$ ls namesecondary/
current  in_use.lock
[ec2-user@ip-172-31-52-194 dfs]$
```

Working with hdfs

- Setting up home directory
  - bin/hdfs dfs -mkdir /user
  - bin/hdfs dfs -mkdir /user/ec2-user

    (ec2-user is user name of your account)

- Create a directory "input" under home
  - bin/hdfs dfs -mkdir /user/ec2-user/input

Working with hdfs

- Copy data from local file system
  - bin/hdfs dfs -put etc/hadoop/*.xml /user/ec2-user/input

- List the content of directory
  - bin/hdfs dfs -ls /user/ec2-user/input

Working with hdfs

- Copy data from hdfs
  – bin/hdfs dfs -get /user/ec2-user/input input1
  – If input1 does not exist, it will create one
  – If it does, it will create another one under it

- Examine the content of file in hdfs
  – bin/hdfs dfs -cat /user/ec2-user/input/core-site.xml

Working with hdfs

- Remove files
  - bin/hdfs dfs -rm /user/ec2-user/input/core-site.xml
  - bin/hdfs dfs -rm /user/ec2-user/input/*

- Remove directory
  - bin/hdfs dfs -rmdir /user/ec2-user/input
  - Directory "input" needs to be empty first

Reading

- K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, 2010, pp. 1-10.