

Hadoop

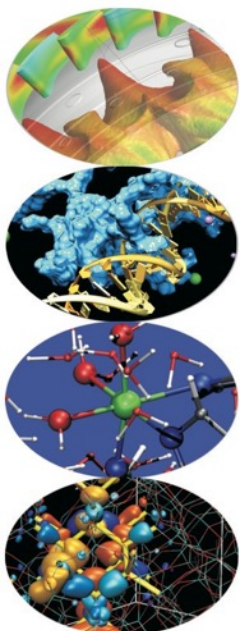
Lab

Paolo D'Onorio De Meo – p.donoriodemeo@cineca.it

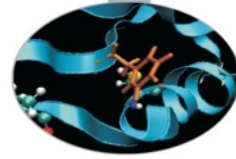
SCAI

SuperComputing Applications and Innovation Department

December 15, 2014



Goals



- **Docker**

- Build your development environment

- Hadoop Java

- Word count example

- **Hadoop streaming**

- Bash pipes

- MapReduce with any language

- **Mrjob**

- Use same code for local or hadoop tests

- Streaming/Mrjob *Use case*

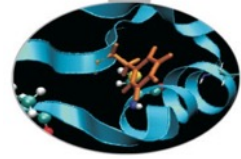
- A scientific real problem

- Breaking the problem into pieces

~ 1h

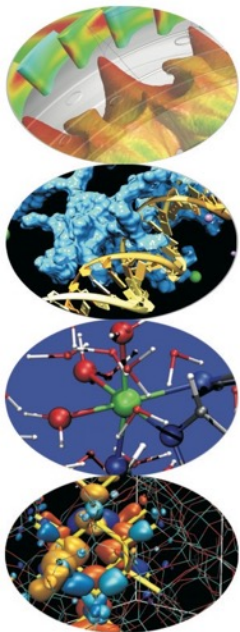
~ 1h

~ 1h

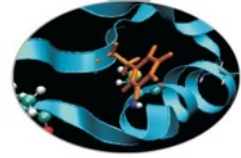


Docker

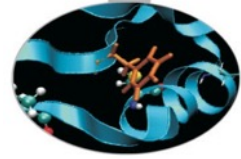
Development & production environment Virtualization



Docker - our schedule

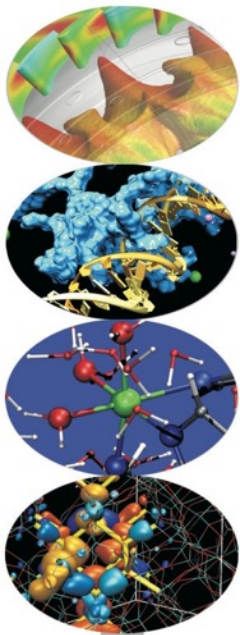


- Concepts
 - From Virtual environment to Docker
- How does it work?
 - Docker images
 - Docker registry
 - Docker containers
- Tutorial
 - Build and image
 - Docker file commands
 - Run a container
- Final thoughts

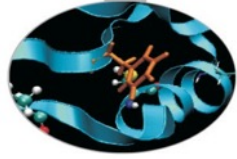


Docker

Concepts

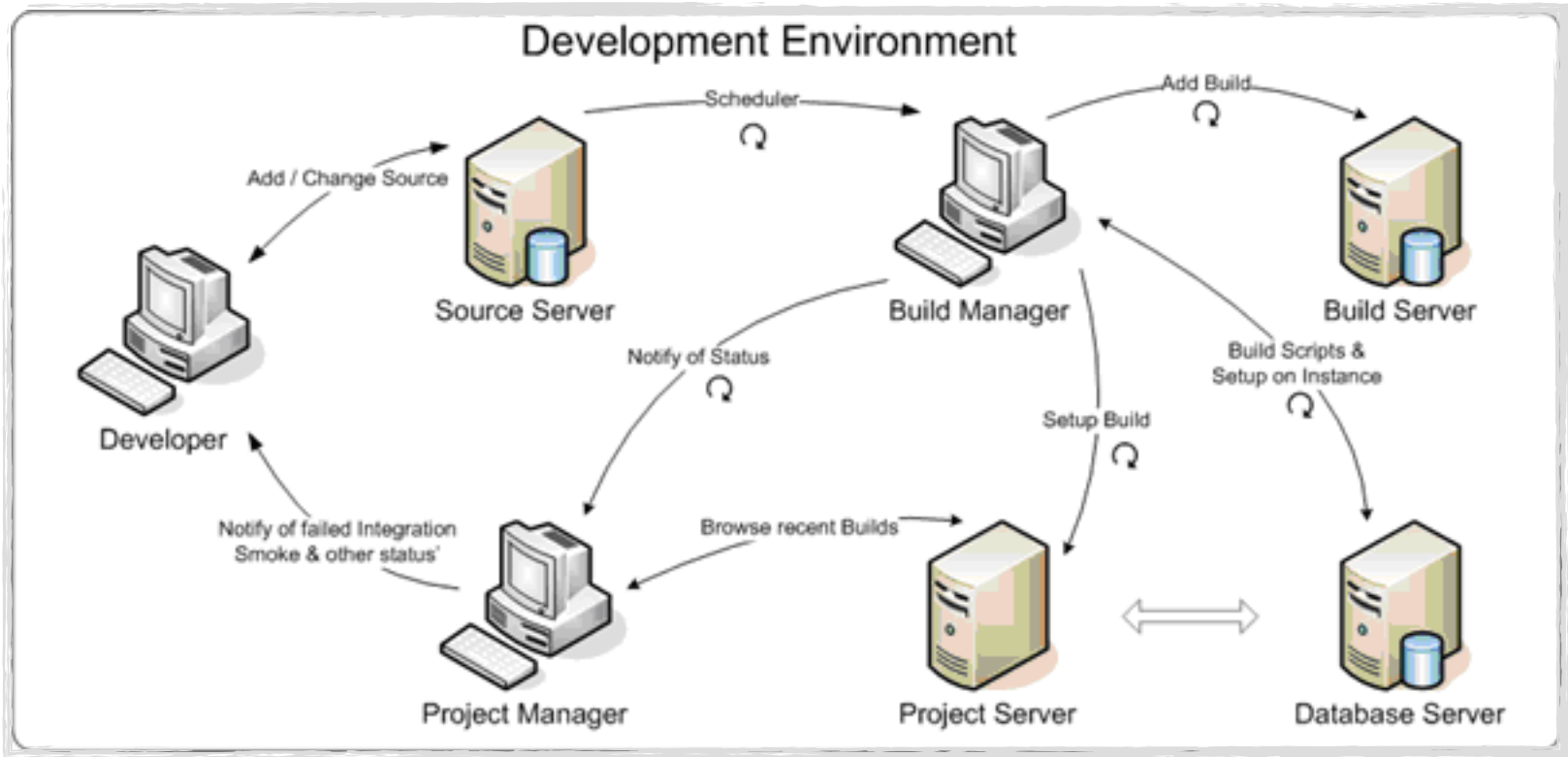
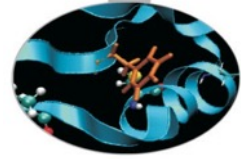


Development environment

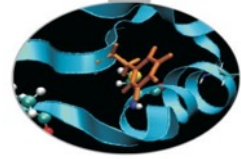


- A development environment is a collection of procedures and tools for developing, testing and debugging an application or program.
- A **set of processes and tools** that are used to develop a source code or program
 - software development
 - testing and debugging an application or program

Development environment ?

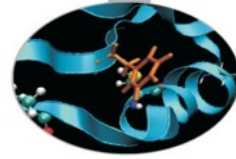


Dev env != IDE



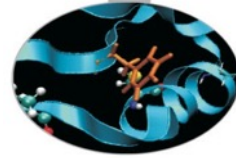
- This term is sometimes used synonymously with integrated development environment (**IDE**)
 - Software tool used to write, build, test and debug a program.
 - Provide developers with a common user interface (UI)
- Generally speaking, the term development environment would refer to the entire environment, including:
 - development
 - staging
 - and production servers
 - ...IDE just refers to the local application used to code!

Virtual environment



- Virtual machine (VM)
 - A software program or operating system
 - Performing tasks (e.g. running applications) like a separate computer
 - known as a **guest**
 - It is created within another computing environment
 - referred as a **host**
 - Multiple virtual machines can exist within a single host
- Virtual environment
 - One or more virtual computer running in a virtual space, which is the combination of virtual machine monitor and hardware platform
 - Manager app is the *hypervisor* (monitor and debugging)

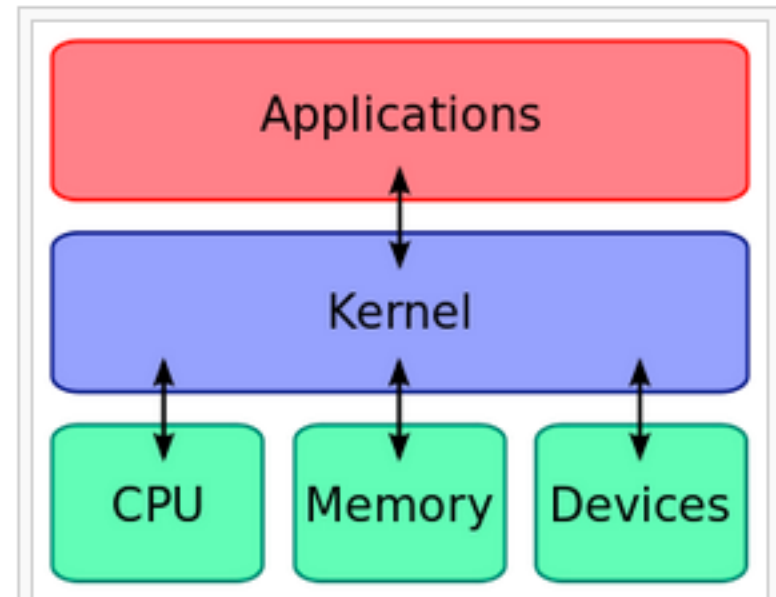
Kernel




The kernel is the **core** part of a modern computer's operating system.

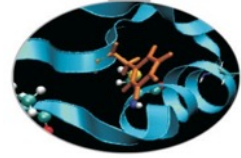
Linux kernel:

<https://github.com/torvalds/linux>



A kernel connects the **application software** to the hardware of a computer 

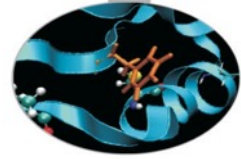
Docker



- Lightweight **container** virtualization platform
 - Run without the extra load of a *hypervisor*
 - Get more out of your hardware
 - Helping you with the development lifecycle
 - Separate your applications from your infrastructure
 - Treat your infrastructure like a managed application
- Portable
 - Run on a developer's local host
 - Run on physical or virtual machines in a data center
 - Run in the Cloud
 - scale up or tear down applications and services

[Source: *docker website*]

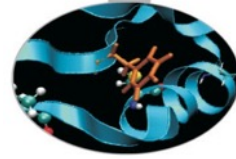
Docker



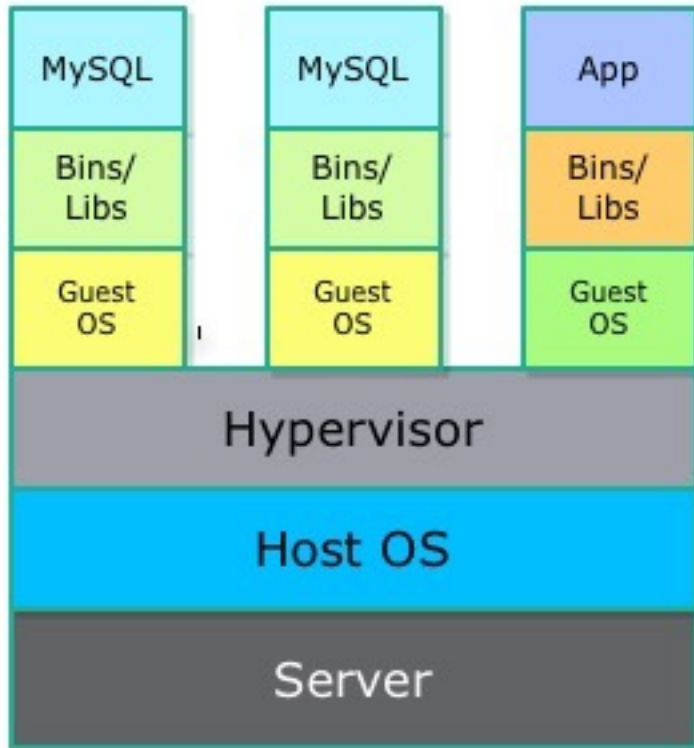
- **Linux** virtual environment
 - Isolation (running *container*) works on running kernel
 - Fast container startup
 - Each container share also same libraries
 - Less resources
 - Easy commands
 - Easy networks and volumes set up
- Developers write code locally
 - share their development stack via Docker with their colleagues

[Source: my self]

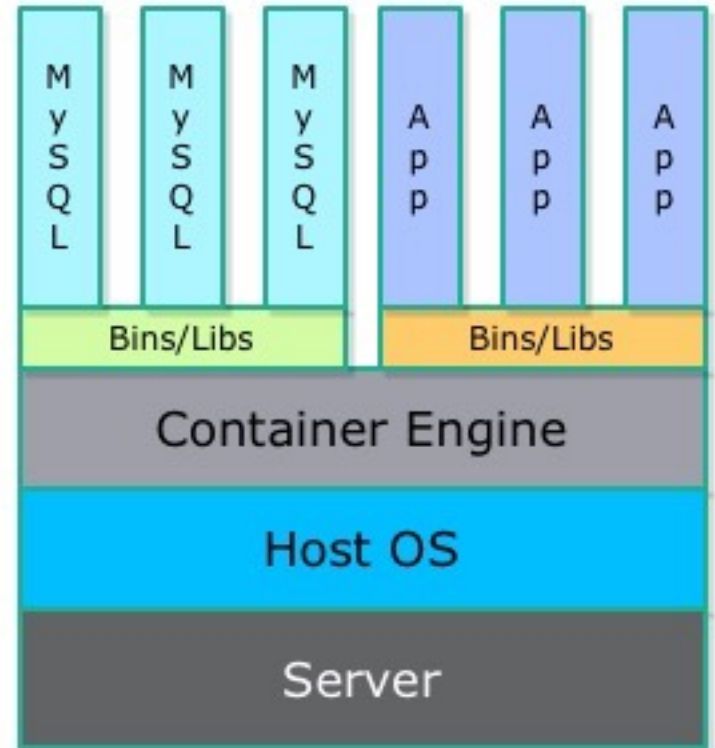
Docker vs Virtual Machines

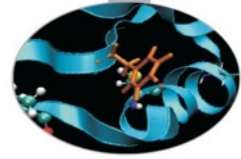


Virtual Machines



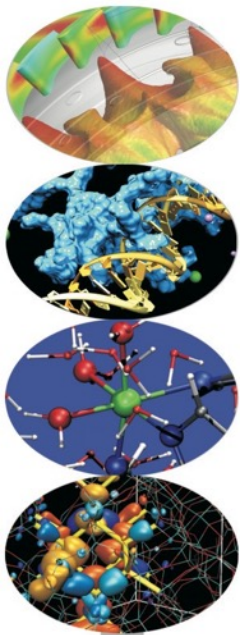
Containers



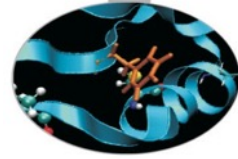


Docker

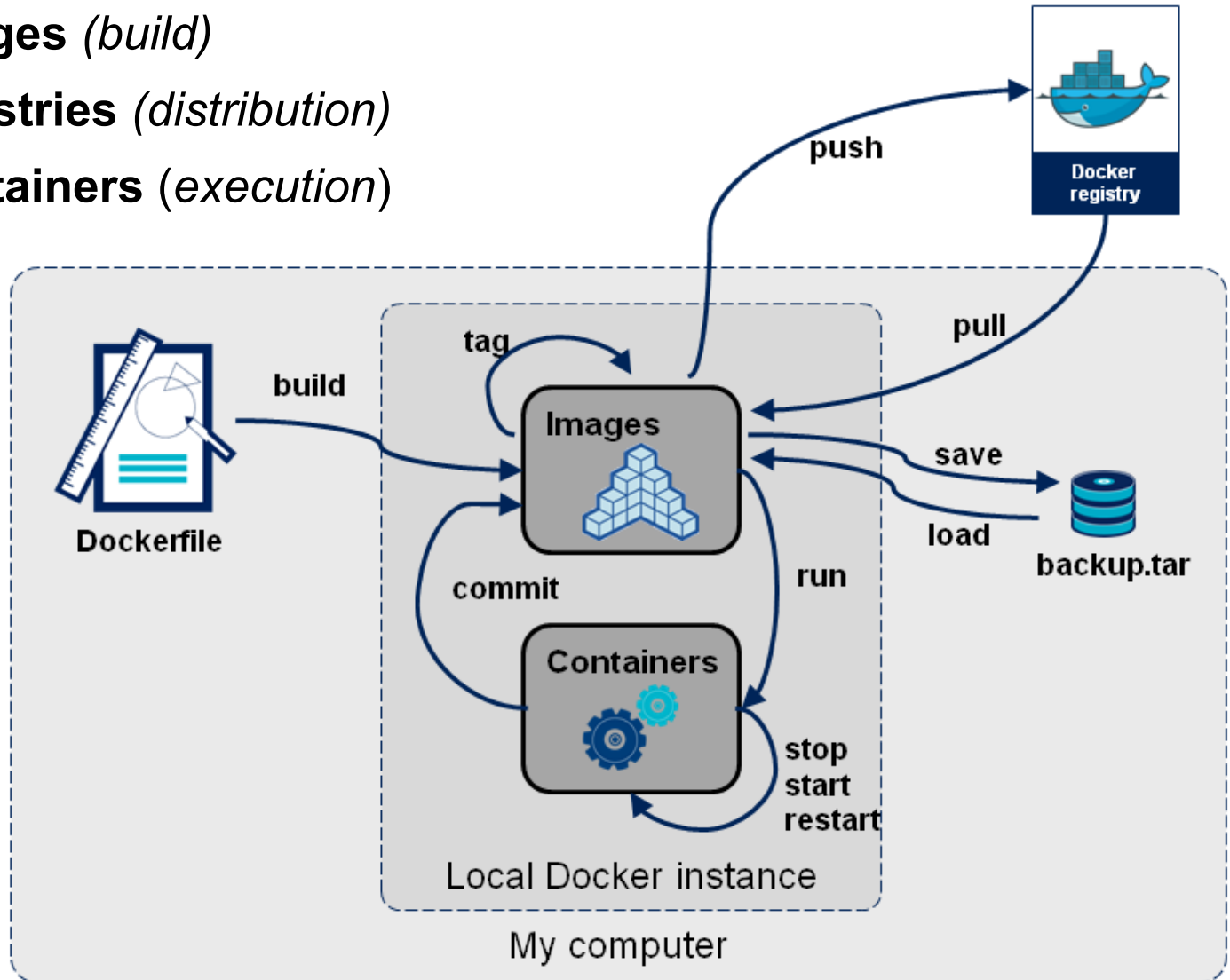
How does it work



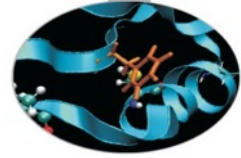
Docker components



- Docker images (*build*)
- Docker registries (*distribution*)
- Docker containers (*execution*)

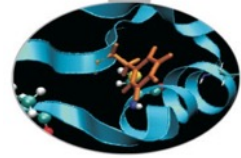


Docker components



- Docker **images** (*build*)
 - A Docker image is a read-only template
 - Images are used to create Docker containers
 - Simple to build new images or update existing
 - **Pull** Docker images that other people have already created
- Docker **registries** (*distribution*)
 - Docker registries hold images
 - Public or private stores where you upload or download images
 - The public Docker registry is called Docker Hub
 - “Social” or personal sharing

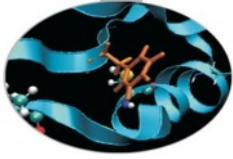
Docker components



(execution)

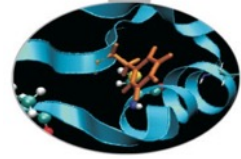
- Docker **containers** *are similar to a directory*
 - Holds everything that is needed for an application to run
- Each container is created from a Docker image
 - Run, start, stop, move, and delete
 - The Docker image is read-only
 - A running container adds a read-write layer on top of the image
 - in which your application can be executed
 - You can run commands on a running containers from outside
 - You can attach your screen to background containers
 - *Each container is an isolated application platform*
 - Technology called namespaces to provide the isolated workspace

Docker hello world



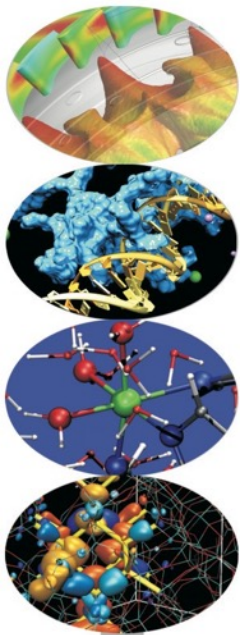
```
$ docker run ubuntu echo "Hello World"
```

```
Hello World
```

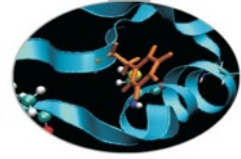


Docker

Tutorial: build and run your image



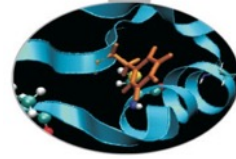
Building an image: Dockerfile



- This file contains a series of operations
 - build container following that order
 - each operation is translated with a commit

```
$ cd PATH/TO/DOCKERFILEDIR/  
$ docker build -t IMAGE_NAME .
```

Two simple steps

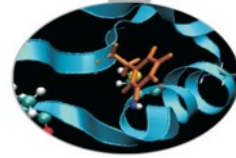


Dockerfile

```
FROM ubuntu  
MAINTAINER "Paolo" @cineca.it
```

```
path/to/file/$ docker build -t test .  
  
Sending build context to Docker  
daemon 2.56 kB  
Sending build context to Docker  
daemon  
Step 0 : FROM ubuntu  
---> 5506de2b643b  
Step 1 : MAINTAINER Paolo @cineca.it  
---> Running in a95b2719228c  
---> 71146070d486  
Removing intermediate container  
a95b2719228c  
Successfully built 71146070d486
```

A new image



```
$ docker build -t test .
```

```
Sending build context to Docker daemon 2.56 kB
```

```
Sending build context to Docker daemon
```

```
Step 0 : FROM ubuntu
```

```
---> 5506de2b643b
```

```
Step 1 : MAINTAINER Paolo @cineca.it
```

```
---> Running in a95b2719228c
```

```
---> 71146070d486
```

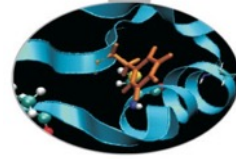
```
Removing intermediate container a95b2719228c
```

```
Successfully built 71146070d486
```

```
$ docker images
```

test	latest	71146070d486	2 minutes ago	199.3 MB
ubuntu	14.04	5506de2b643b	6 weeks ago	199.3 MB
ubuntu	latest	5506de2b643b	6 weeks ago	199.3 MB

Interactive container



```
$ docker build -t test .
```

```
Sending build context to Docker daemon 2.56 kB
```

```
Sending build context to Docker daemon
```

```
Step 0 : FROM ubuntu
```

```
---> 5506de2b643b
```

```
Step 1 : MAINTAINER Paolo @cineca.it
```

```
---> Running in a95b2719228c
```

```
---> 71146070d486
```

```
Removing intermediate container a95b2719228c
```

```
Successfully built 71146070d486
```

```
$ docker run -it test bash
```

```
root@5088cb900eef:/# lsb_release -a
```

```
No LSB modules are available.
```

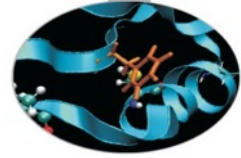
```
Distributor ID: Ubuntu
```

```
Description: Ubuntu 14.04.1 LTS
```

```
Release: 14.04
```

```
Codename: trusty
```

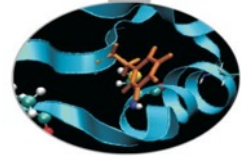
Docker build



```
$ docker build -t IMAGE_NAME PATH
```

- “docker” is the binary
- “build” is the command
- -t option to decide image name
- path is where you have saved your Dockerfile
 - WARNING: each file inside that path will be copied/sync inside the container context

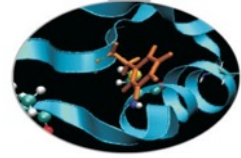
Docker run



```
$ docker run [-it] IMAGE_NAME [COMMAND]
```

- “docker” is the binary
- “run” is the command
- -it option run interactive tty (e.g. for a shell)
- you can specify a command to be executed on your running container

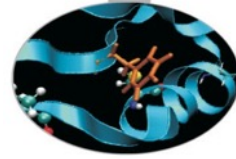
Docker implicits



But what if you try to start/run a container on a non local image?

- docker will try to pull it from public Hub
 - e.g. `docker run ubuntu:12.04`
 - docker will try to find ubuntu on its Hub, version 12.04
 - e.g. `docker run mysql`
 - docker will try to find “mysql” on its Hub, version “latest”

Demonized container



```
$ docker build -t test .
```

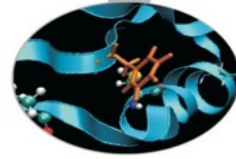
```
$ docker run -d test sleep 1000
```

```
24b0c8b1efc5d37fff7c5d9f7c35147ad546ef2af32794ecb439d88ecdbcfdb7
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	
STATUS	PORTS	NAMES		
24b0c8b1efc5	test:latest	"sleep 1000"	6 seconds ago	Up
5 seconds		hopeful_goodall		

Demonized container



```
$ docker run -d ubuntu sh -c "while true; do echo hello world; sleep 1; done"  
2a358de827d309be6b53ff68756af60acc85e7cd4bccb420611b920290feb084
```

```
# Container LONG ID
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
2a358de827d3	ubuntu:14.04	"sh -c 'while true; kickass_hoover	43 hours ago
Up 2 seconds			

```
# Container SHORT ID and Random name
```

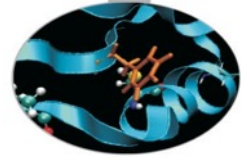
```
$ docker logs kickass_hoover
```

```
hello world  
hello world  
hello world  
hello world  
hello world  
hello world
```

```
# CTRL + c
```

```
$ docker stop kickass_hoover
```

Tutorial: image step by step



Dockerfile

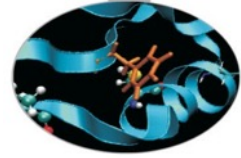
```
FROM ubuntu:12.04
# using old LTS version
MAINTAINER Paolo @cineca.it

# Start modifications from here
```

```
$ docker build -t test .
$ docker run -it test bash

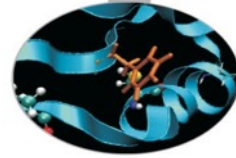
root@8f0a8e14f4de:/# lsb_release -a
bash: lsb_release: command not
found
```

Dockerfile: FROM



- Decide the base of my new image
 - It has to be the first command of your image
 - Pull from docker Hub, if not configured
- Many lightweight
 - Busybox
- Many linux distro available (Ubuntu, Debian, CentOS, Suse)
 - Server versions
 - Most recent versions (e.g. **ubuntu:12.04**)
 - If no version is specified -> get “latest” tag
 - You should **NOT** work your image on “latest” tag
- Many working images for famous services
 - Mysql, MongoDB

Version



```
$ docker build -t test:0.1 .
```

```
Sending build context to Docker daemon
```

```
Step 0 : FROM ubuntu:12.04
```

```
ubuntu:12.04: The image you are pulling has been verified
```

```
03390508e7a5: Downloading [==> ] 8.099 MB/135.9 MB 3m30s
```

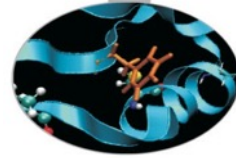
```
344b4267f7ef: Download complete
```

```
18ee37ab2090: Download complete
```

```
58c0a77963ea: Download complete
```

```
511136ea3c5a: Already exists
```

Version

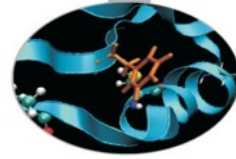


```
$ docker build -t test:0.1 .

Sending build context to Docker daemon 2.56 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:12.04
ubuntu:12.04: The image you are pulling has been verified

03390508e7a5: Pull complete
344b4267f7ef: Pull complete
18ee37ab2090: Pull complete
58c0a77963ea: Pull complete
511136ea3c5a: Already exists
Status: Downloaded newer image for ubuntu:12.04
---> 58c0a77963ea
Step 1 : MAINTAINER Paolo @cineca.it
---> Running in bf2f1191e1d1
---> 9dc227984092
Removing intermediate container bf2f1191e1d1
Successfully built 9dc227984092
```

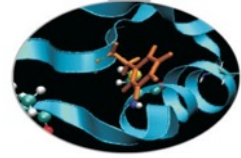

Version



```
$ docker images
```

test	0.1	9dc227984092	About a minute ago	130.1 MB
test	latest	71146070d486	9 minutes ago	199.3 MB
ubuntu	12.04	58c0a77963ea	3 days ago	130.1 MB
ubuntu	14.04	5506de2b643b	6 weeks ago	199.3 MB
ubuntu	latest	5506de2b643b	6 weeks ago	199.3 MB

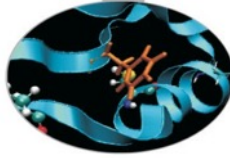
Docker tag



```
$ docker tag IMAGE[:TAG] [REGISTRYHOST/] [USERNAME/] NAME[:TAG]
```

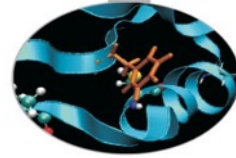
- “docker” is the binary
- “tag” is the command
- “image” is the existing image
- Last option defines a new registry, username, image name and tag
 - e.g. `docker tag ubuntu:12.04 myubuntu:0.1`
 - e.g. `docker tag ubuntu myubuntu:0.2`
 - e.g. `docker tag mysql:5.5 CinecaHub/pdonorio/mysql`

??



```
$ docker run busybox ls  
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys  
tmp usr var  
  
$ docker run busybox echo "Hello World"  
Hello World  
  
$ docker run busybox touch /root/test  
  
$ docker run busybox ls /root  
  
$
```

Zombies



```
root@8f0a8e14f4de:/# exit
```

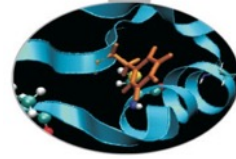
```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
9c09f7cf8d1e	test:latest	"bash"	4 minutes
ago	Exited (0) 2 minutes ago		
determined_ptolemy			
8f0a8e14f4de	test:0.1	"bash"	5 minutes
ago	Exited (0) 2 minutes ago		happy_franklin

Zombies: clean!



```
# CLEAN ZOMBIES
```

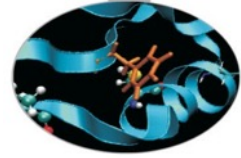
```
$ docker stop $(docker ps -a -q)
```

```
$ docker rm $(docker ps -a -q)
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	

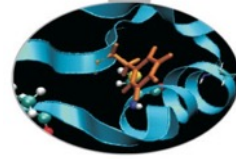
run: Behind the curtains



Docker run [options] IMAGE PROCESS

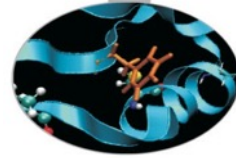
- Pulls the image
 - checks for the presence of the image
 - if it doesn't exist locally downloads it from Docker Hub
- Creates a new container from the read-only image
 - The container is created in the file system
 - Creates a set of *namespaces* for that container
 - A read-write layer is added to the image.
- Allocates a network / bridge interface
 - that allows the container to talk to the local host
- Sets up an IP address
 - Finds and attaches an available IP address from a pool.
- Executes a process that you specify
 - Captures and provides application output
 - Connects and logs standard input, outputs and errors

run: Behind the curtains



```
$ docker create --name mine ubuntu sh -c "while true; do sleep 100; done"
600ae919f9dbda6b686de109abed46cf5be9a85444438758688c3a44aab9eddd
$ docker ps
CONTAINER ID          IMAGE          COMMAND          CREATED
STATUS              PORTS         NAMES
$ docker ps -a
CONTAINER ID          IMAGE          COMMAND          CREATED
STATUS              PORTS         NAMES
600ae919f9db         ubuntu:14.04  "sh -c 'while true;
mine                43 hours ago
$ docker start mine
mine
$ docker ps
CONTAINER ID          IMAGE          COMMAND          CREATED
STATUS              PORTS         NAMES
600ae919f9db         ubuntu:14.04  "sh -c 'while true;
mine                43 hours ago
Up 2 seconds
$ docker exec -it mine bash
root@600ae919f9db:/# ps -ax
  PID TTY          STAT       TIME COMMAND
    1 ?           Ss          0:00 sh -c while true; do sleep 100; done
    7 ?           S           0:00 sleep 100
    8 ?           S           0:00 bash
   24 ?          R+          0:00 ps -ax
```

Update packages



Dockerfile

```
FROM ubuntu:14.04
# using LTS version
MAINTAINER Paolo @cineca.it

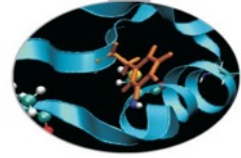
# Start modifications from here
RUN apt-get update
```

```
$ docker build -t test .
```

```
Sending build context to Docker
daemon 3.584 kB
Sending build context to Docker
daemon
Step 0 : FROM ubuntu:14.04
---> 9bd07e480c5b
Step 1 : MAINTAINER "Paolo D'Onorio
De Meo" p.donoriodemeo@cineca.it
---> Using cache
---> d8d7e8e1b9e7
Step 2 : RUN apt-get update &&
apt-get install -y python-setuptools
libpython2.7-dev libyaml-dev &&
echo "Done"
---> Running in 520e25842671
Ign http://archive.ubuntu.com trusty
InRelease
Ign http://archive.ubuntu.com trusty-
updates InRelease

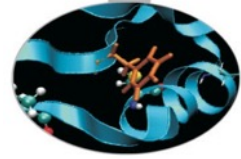
[...]
```


Dockerfile: RUN command



- Execute a shell command **at build time**
 - RUN *<command>*
 - run the *command* in a shell (/bin/sh) or
 - RUN [“executable”, “option1”, “option2”]
 - equivalent to \$ executable option1 option2
 - e.g. RUN [“/bin/bash”, “script”, “-o outfile”]
- The RUN instruction will execute any commands in a new layer
 - on top of the current image
 - and commit the results
- The committed image will be used for the next step in the Dockerfile

Install packages



Dockerfile

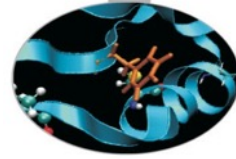
```
FROM ubuntu:14.04
# using LTS version
MAINTAINER Paolo @cineca.it

# Install packages
RUN apt-get update
RUN apt-get install -y python-
setuptools

# Set up python
RUN easy_install pip
RUN pip install ipython mrjob
```

```
$ docker build -t test .
Sending build context to Docker
$ docker run -it test bash
# ipython
[1] import mrjob
[2]
```

Last fixes



Dockerfile

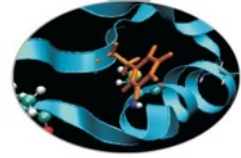
```
FROM ubuntu:14.04
# using LTS version
MAINTAINER Paolo @cineca.it

# Install packages
RUN apt-get update
RUN apt-get install -y python-
setuptools
# Set up python
RUN easy_install pip
RUN pip install ipython mrjob

ADD myscript.py /opt
WORKDIR /opt
```

```
$ docker build -t test .
Sending build context to Docker
$ docker run -it test bash
/opt# python myscript.py
Hello world
```

Docker sharing host dir

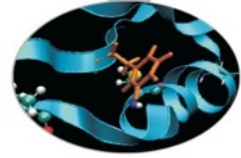


- Mount a Host Directory as a Data Volume
 - Have local code you can edit from host and test on the guest
 - Persistence of databases or data files

```
$ docker run -it -v /local/dir:/guest/dir ubuntu bash
```

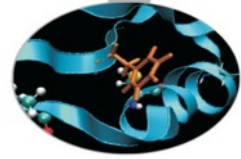
- *Managing data in containers*
 - <https://docs.docker.com/userguide/dockervolumes/>

Docker networking



- Container linking
- Ports
 - EXPOSE Docker file command
 - intra-network
 - -p docker binary flag
 - from the outside world
 - implicit EXPOSE

Docker linking



```
$ docker run -d --name mysqldb \  
-e MYSQL_ROOT_PASSWORD="test" mysql
```

```
$ docker run -it --link mysqldb:db ubuntu bash
```

```
root@7d5e2101effe:/# telnet db 3306
```

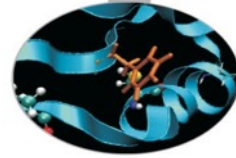
```
Trying 172.17.0.39...
```

```
Connected to db.
```

```
Escape character is '^]'.  
J
```

```
5.6.22fCZzxn<'DcDhb3rh3cxmysql_native_password
```

Docker hub: Finding images

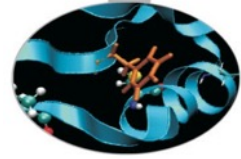


```
$ docker search -s 2 sinatra
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
training/sinatra		5		

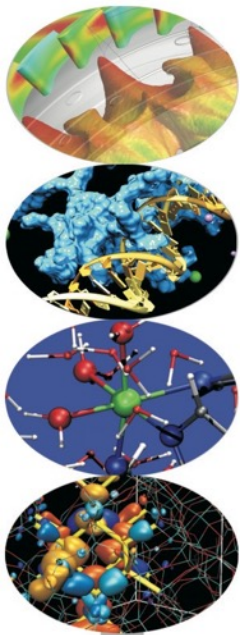
```
$ docker search -s 10 mysql
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTO
mysql	MySQL is a widely used, open-source relati...	321	[OK]	
tutum/mysql	MySQL Server image - listens in port 3306...	81		[OK]
tutum/lamp	LAMP image - Apache listens in port 80, an...	40		[OK]
orchardup/mysql		37		[OK]
tutum/wordpress	Wordpress Docker image - listens in port 8...	29		[OK]

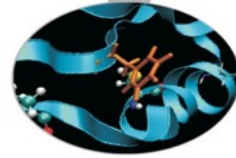


Docker

Final thoughts



Docker commands: summary



docker binary

images
tag
pull

create
start
exec
stop

run

rm

ps

Dockerfile

```
FROM image:tag
```

```
RUN command
```

```
ADD file imagepath/
```

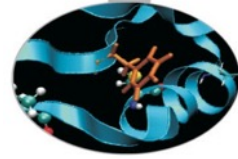
```
WORKDIR dir
```

```
EXPOSE port
```

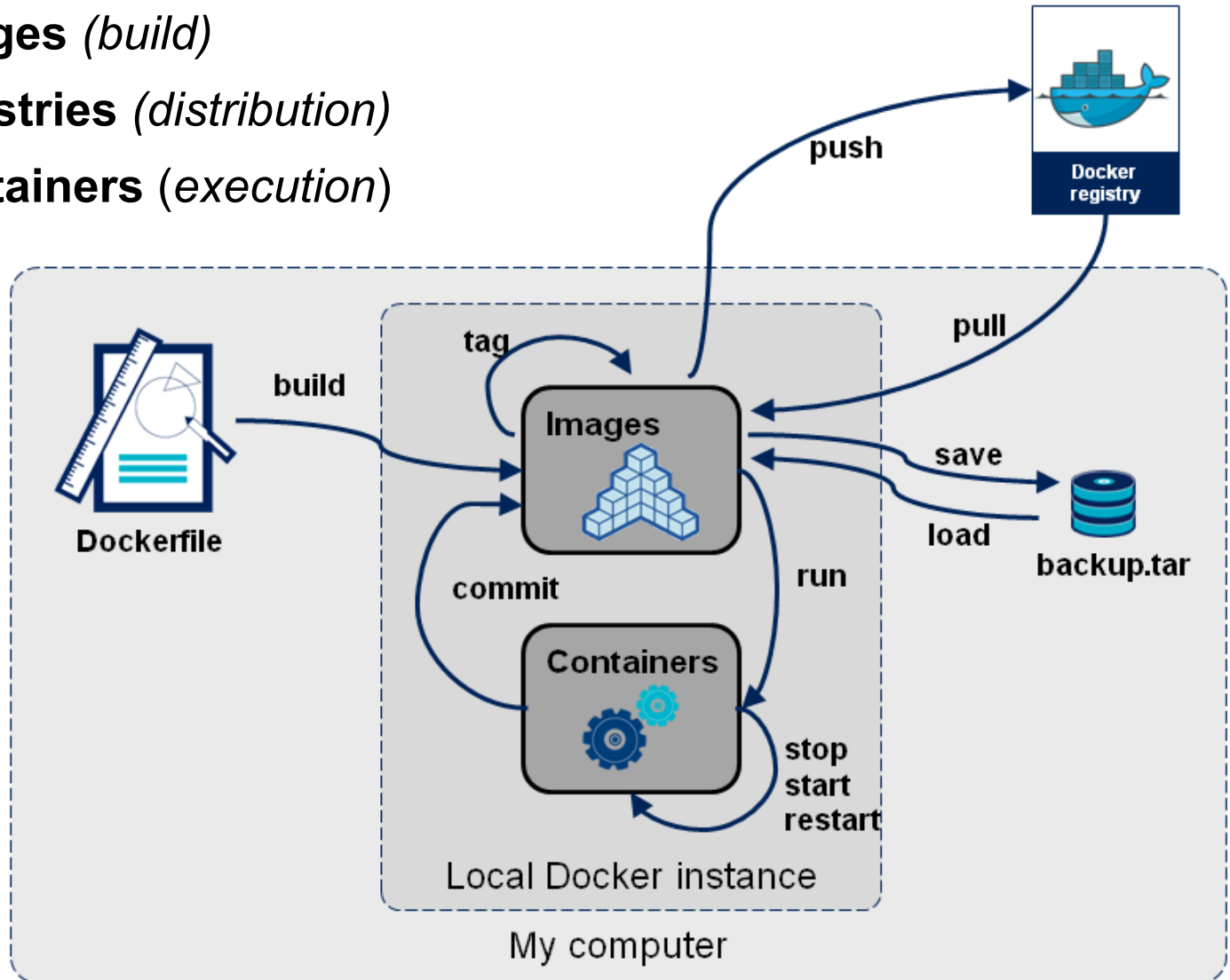
```
ENTRYPOINT ["bin"]
```

```
CMD ["command", "options"]
```

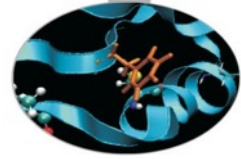
Docker components



- Docker images (*build*)
- Docker registries (*distribution*)
- Docker containers (*execution*)

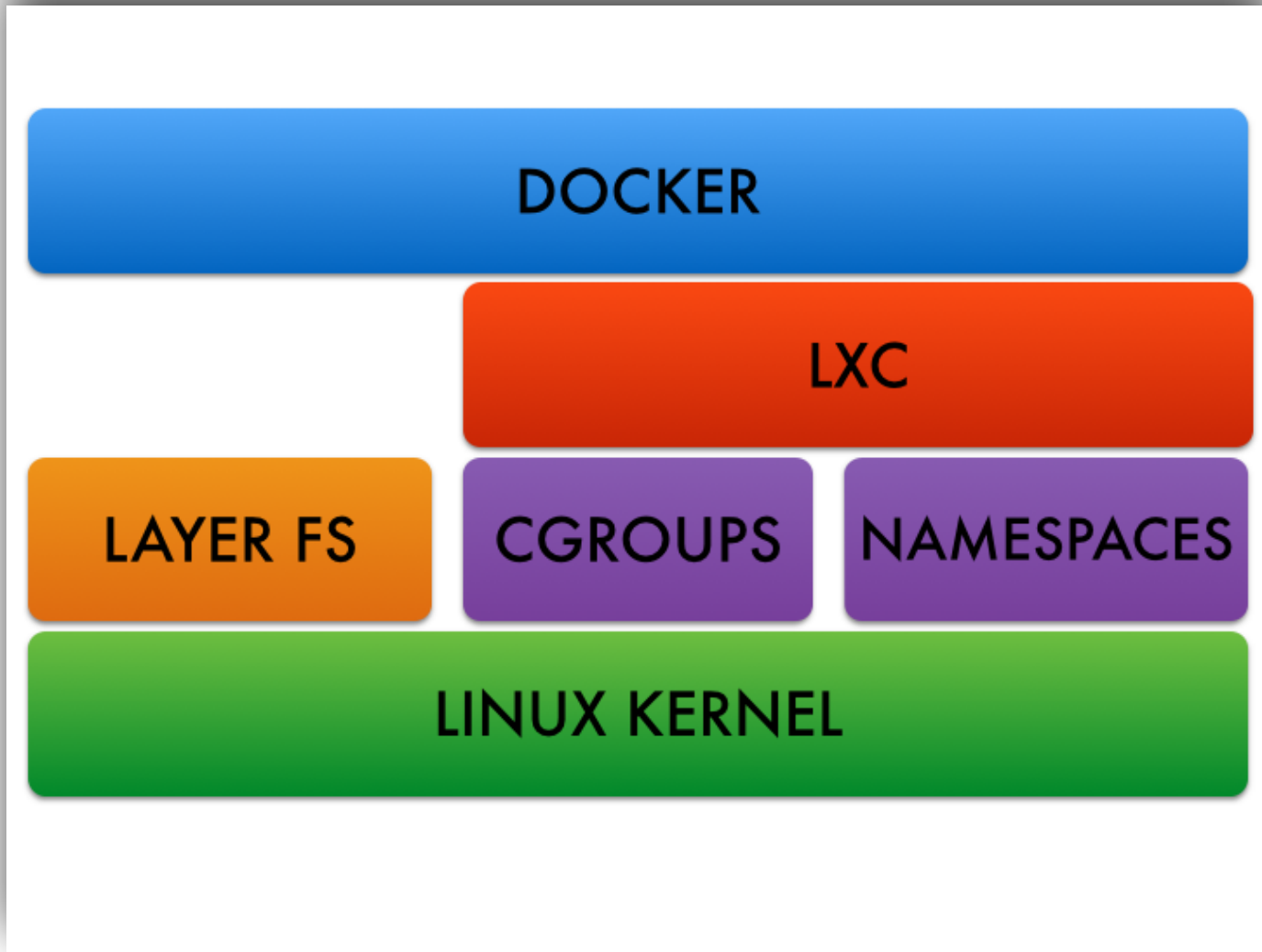
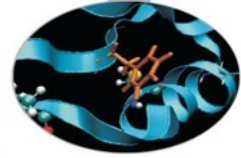


Other platforms: Boot2docker

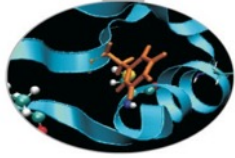


- Win and Mac
 - A Virtualbox Linux virtual machine with docker
 - Forwards
 - Ports
 - Volumes (C:\Users and /Users partition only)

Docker architecture



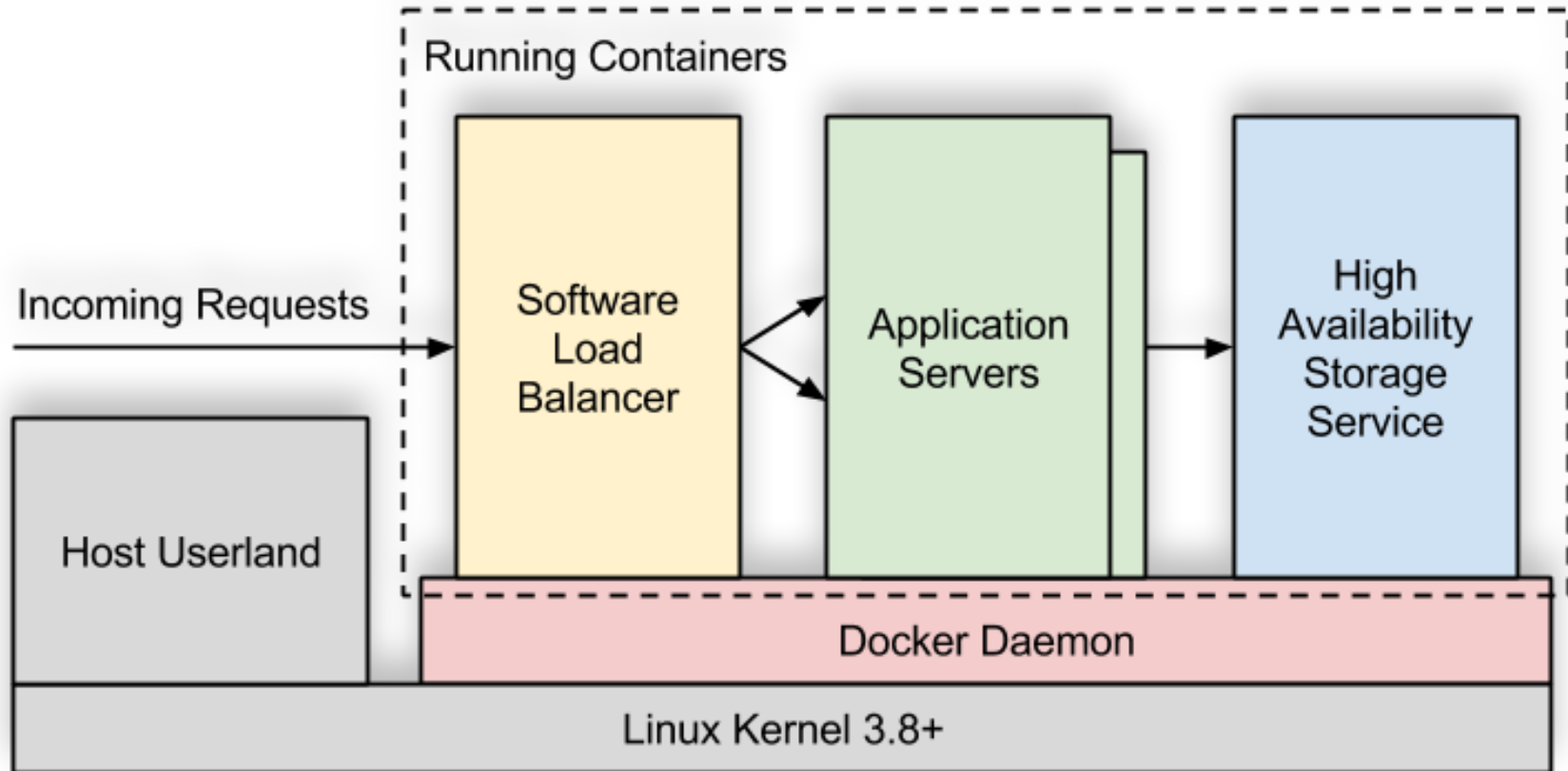
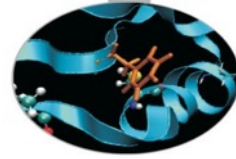
Dockerize your software



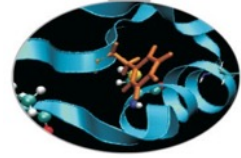
Dockerizing a production system: simple steps

- Network
- Storage and Volumes
 - Backup, restore, or migrate data volumes
- Setup scripts
- Daemonize your application

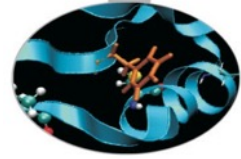
Dockerize your software



Docker: your image

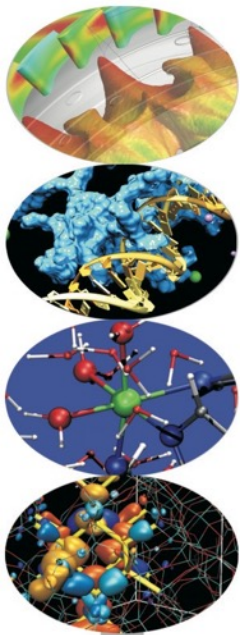


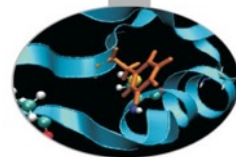
- Cineca hadoop/mrjob example
- Do not forget:
 - <https://docs.docker.com/userguide/>
 - Introspection
 - port
 - inspect
 - logs
 - top



Docker

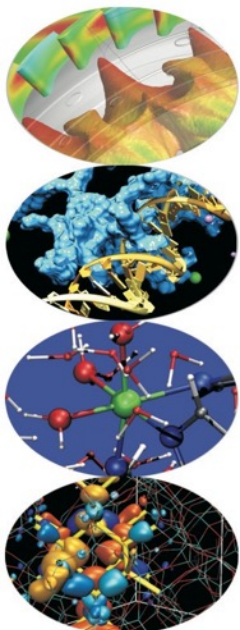
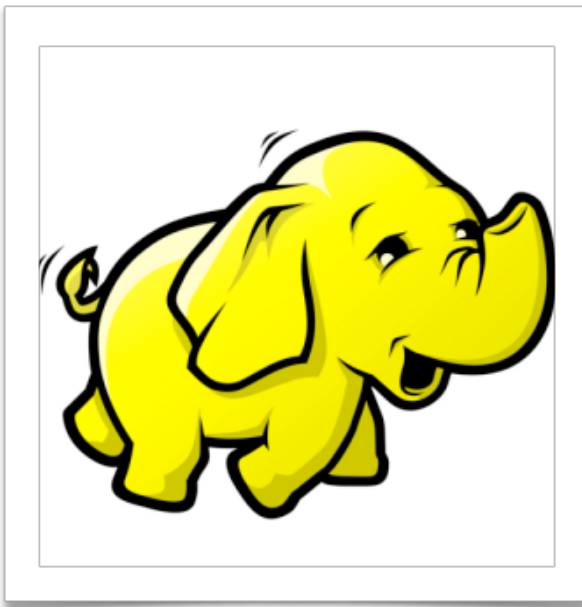
Well done!



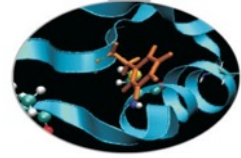


Hadoop

Java

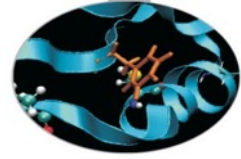


MapReduce: summary

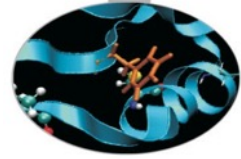


- The map/reduce is a completely different *paradigm*
 - Solving a certain subset of *parallelizable* problems
 - around the bottleneck of ingesting input data from disk
- Traditional parallelism *brings the data to the computing machine*
 - Map/reduce does the opposite, it brings the compute to the data
 - Input data is not stored on a separate storage system
 - Data exists in little pieces and is permanently stored on each computing node

MapReduce: summary

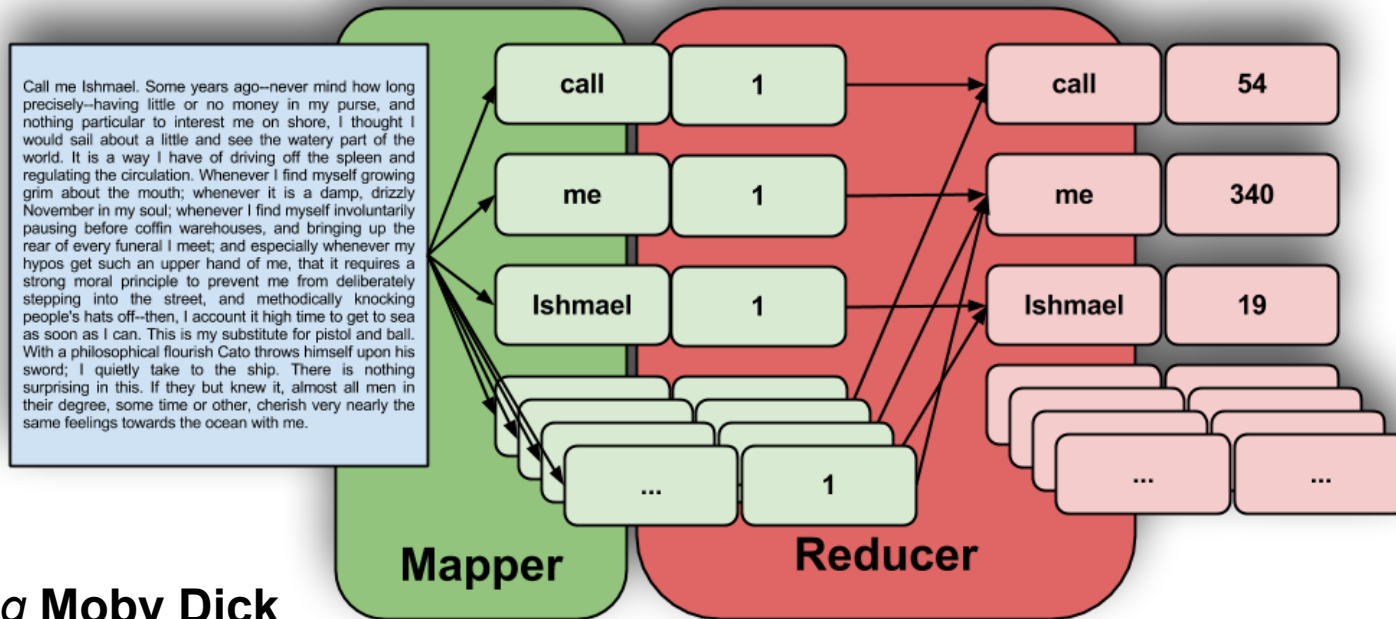


- The map/reduce is a completely different *paradigm*
 - **HDFS** is fundamental to Hadoop
 - it provides the data chunking
 - distribution across compute elements
 - necessary for map/reduce applications to be efficient



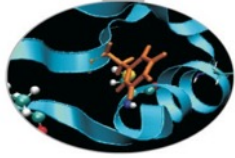
Word count

- Counting the number of words in a large document
 - This is the "hello world" of map/reduce
 - Among the simplest of full Hadoop jobs you can run

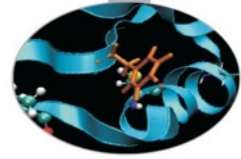


Reading Moby Dick

Word count



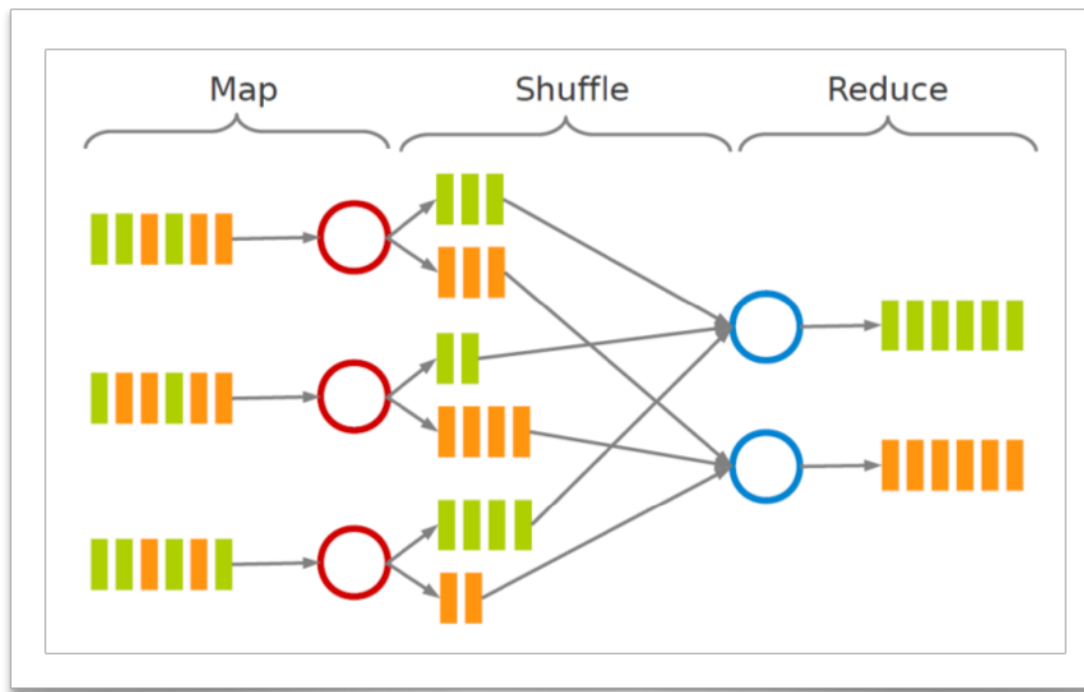
- The MAP step will take the raw text and convert it to **key/value pairs**
 - Each key is a word
 - All keys (words) will have a value of 1
- The REDUCE step will combine all **duplicate keys**
 - By adding up their values (*sum*)
 - Every key (word) has a value of 1 (Map)
 - Output is *reduced* to a list of unique keys
 - Each key's value corresponding to key's (word's) count.



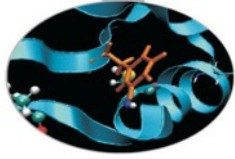
Word count

Map function : processes data and generates a set of intermediate key/value pairs.

Reduce function : merges all intermediate values associated with the same intermediate key.



Word count execution

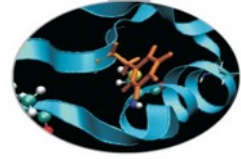


Consider doing a word count of the following file using MapReduce:

```
$ cat file.txt
```

```
Hello World Bye World  
Hello Hadoop Goodbye Hadoop
```

Word count: map



- The map function reads in words
 - one at a time
 - outputs (“word”, 1) for each parsed input word

Map output is:

(Hello, 1)

(World, 1)

(Bye, 1)

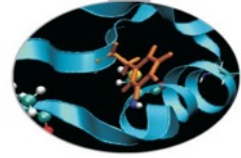
(World, 1)

(Hello, 1)

(Hadoop, 1)

(Goodbye, 1)

Word count: shuffle



- The shuffle phase between map and reduce
 - creates a list of values associated with each key

Shuffle output / Reduce input is:

(Bye, (1))

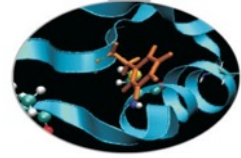
(Goodbye, (1))

(Hadoop, (1, 1))

(Hello, (1, 1))

(World, (1, 1))

Word count: reduce



- The reduce function sums the numbers in the list
 - for each key and outputs (word, count) pairs

Reduce output is:

(also the output of the MapReduce job)

(Bye, 1)

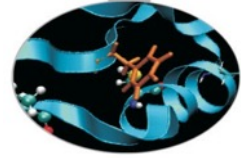
(Goodbye, 1)

(Hadoop, 2)

(Hello, 2)

(World, 2)

Word count: Java code



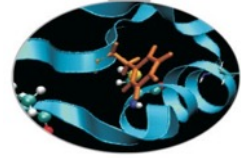
```
package org.myorg;
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.*

public class WordCount {

    public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text,
    IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

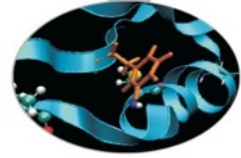
        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
        Reporter reporter) throws IOException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }
    }
}
```

Word count: Java code



```
public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text,
IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable>
output, Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

Word count: Java code



```
public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

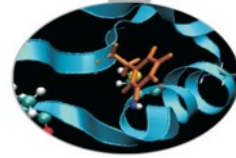
    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
}
}
```

Word count: Java execution

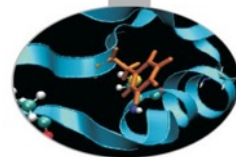


```
$ hadoop jar hadoop-examples.jar wordcount dft dft-output
```

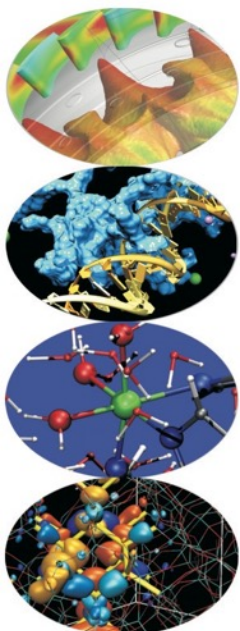
```
10/03/16 11:40:51 INFO mapred.FileInputFormat: Total input
paths to process : 1 10/03/16 11:40:51 INFO mapred.JobClient:
Running job: job_201003161102_0002 10/03/16 11:40:52 INFO
mapred.JobClient: map 0% reduce 0% 10/03/16 11:40:55 INFO
mapred.JobClient: map 9% reduce 0% 10/03/16 11:40:56 INFO
mapred.JobClient: map 27% reduce 0% 10/03/16 11:40:58 INFO
mapred.JobClient: map 45% reduce 0% 10/03/16 11:40:59 INFO
mapred.JobClient: map 81% reduce 0% 10/03/16 11:41:01 INFO
mapred.JobClient: map 100% reduce 0% 10/03/16 11:41:09 INFO
mapred.JobClient: Job complete: job_201003161102_0002
```

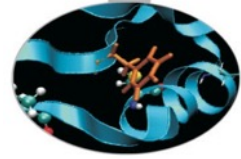
```
$ hadoop dfs -cat dft-output/part-00000 | less
```

```
"Information" 1
"J" 1
"Plain" 2
"Project" 5
```

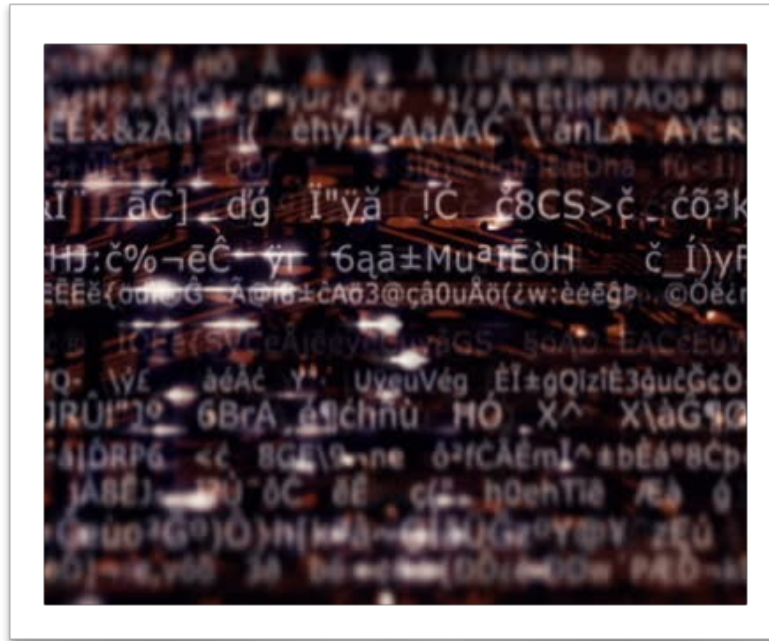
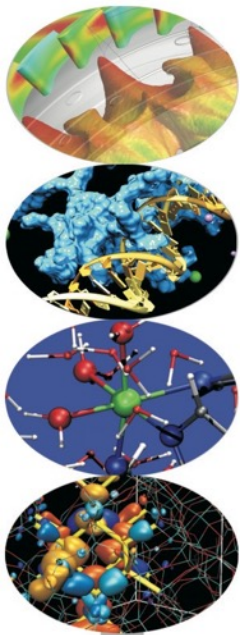


Break!

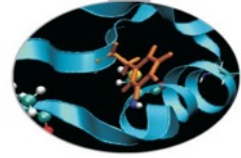




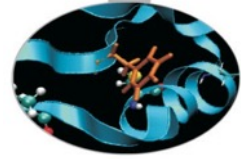
Hadoop Streaming



Streaming - our schedule

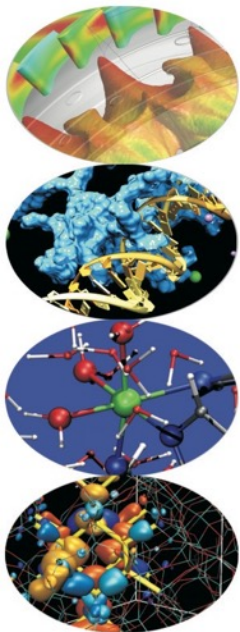


- Simulation of streaming via bash pipes
- Hadoop Streaming
 - The mechanism behind
 - Command line
- Python code with Hadoop streaming
 - Debug mode
 - Mapper
 - Input
 - Producing output
 - Reducer
 - Input
 - Producing final output
 - Compression

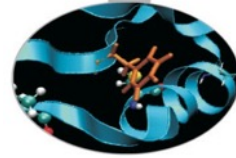


Hadoop streaming

Starting from the bases: bash pipes

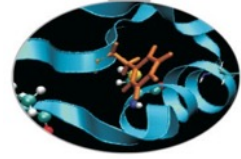


Hadoop via bash pipes



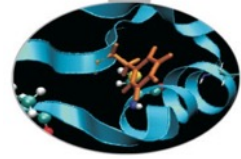
```
# head -2000 data/ngs/input.sam | tail -n 10 \  
  | awk '{ print $3":"$4 }' | sort | uniq -c  
  
 3 chr1:142803456  
 3 chr1:142803458  
 1 chr1:142803465  
 1 chr1:142803470  
 2 chr1:142803471
```

Hadoop via bash pipes



```
head -2000 data/ngs/input.sam | tail -n 10  
awk '{ print $3":"$4 }'  
sort  
uniq -c
```

Hadoop via bash pipes



```
# INPUT STREAMING
```

```
head -2000 data/ngs/input.sam | tail -n 10
```

```
# MAPPER
```

```
awk '{ print $3":"$4 }'
```

```
# SHUFFLE
```

```
sort
```

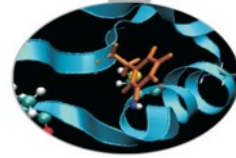
```
# REDUCER
```

```
uniq -c
```

```
# OUTPUT STREAMING
```

```
<STDOUT>
```

Hadoop via bash pipes

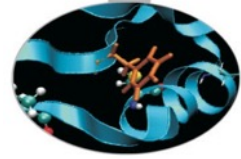


```
# head -2000 data/ngs/input.sam | tail -n 10
HSCAN:421:C47DAACXX:3:1206:5660:99605      99      chr1      142803456      10
      75M      =      142803517      136
CTGTGCATTCTTATGATTTTAATATTCTGTACATTTATTATTGATTTAAAATGCATTTTACCTTTTTCTTTAATA
@@CDDDD?DBFHFHIHIJJJJGHEIFHIIIHGDFEGEIAHAHGIIGIIC?CBFCGHFCEG?@DGIIIGHIGGHC X0:i:
3      X1:i:1      XA:Z:chr21,+9746045,75M,0;chr1,+143355186,75M,
0;chr1,-143233123,75M,1; MD:Z:75 RG:Z:1 XG:i:0 AM:i:0 NM:i:0 SM:i:0 XM:i:
0      X0:i:0      MQ:i:18      XT:A:R
```

[...]

```
# head -2000 data/ngs/input.sam | tail -n 10 | awk '{ print $3":"$4 }'
chr1:142803471
chr1:142803456
chr1:142803465
chr1:142803458
chr1:142803456
chr1:142803471
chr1:142803458
chr1:142803456
chr1:142803470
chr1:142803458
```

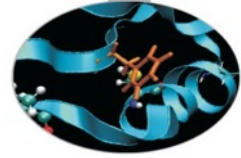
Hadoop via bash pipes



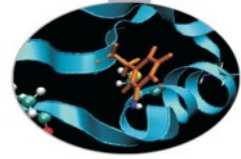
```
# head -2000 data/ngs/input.sam | tail -n 10 | awk '{ print $3:"$4 }' | sort  
chr1:142803456  
chr1:142803456  
chr1:142803456  
chr1:142803458  
chr1:142803458  
chr1:142803458  
chr1:142803465  
chr1:142803470  
chr1:142803471  
chr1:142803471
```

```
# head -2000 data/ngs/input.sam | tail -n 10 | awk '{ print $3:"$4 }' | sort |  
uniq -c  
3 chr1:142803456  
3 chr1:142803458  
1 chr1:142803465  
1 chr1:142803470  
2 chr1:142803471
```

Hadoop via bash pipes

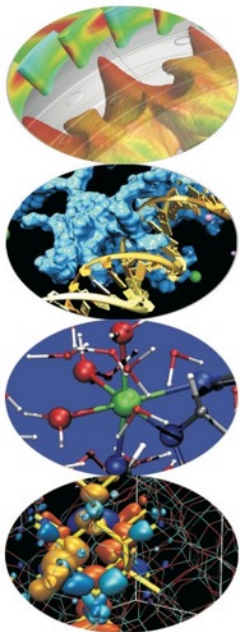


- Serial steps
- No file distribution
- Single node
 - Single mapper
 - Single reducer
- Combiner?

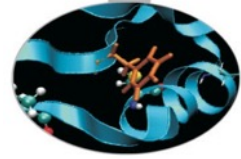


Hadoop streaming

Concepts and mechanisms

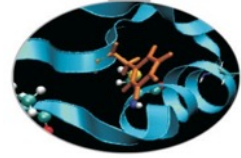


Streaming



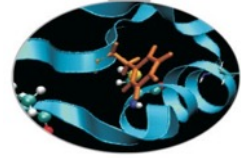
- Hadoop streaming is a **utility**
 - comes with the Hadoop distribution
- Allows to create and run Map/Reduce jobs
 - with **any executable or script** as the mapper and/or the reducer
- Protocol steps:
 - Create a Map/Reduce job
 - Submit the job to an appropriate cluster
 - Monitor the progress of the job until it completes
 - Links to Hadoop HDFS job directories

Why?



- One of the unappetizing aspects of Hadoop to users of traditional HPC is that it is written in Java.
 - Java is not originally designed to be a high-performance language
 - Learning it is not easy for domain scientists
- Hadoop allows you to write map/reduce code **in any language you want** using the **Hadoop Streaming interface**
 - It means turning an existing Python or Perl script into a Hadoop job
 - Does not require learning any Java at all

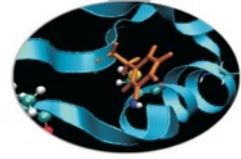
Map&Reduce as executables



Hadoop Streaming utility

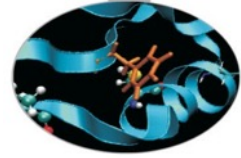
- Executable is specified for mappers and reducers
 - each mapper task will launch the executable as a *separate process*
 - converts inputs into lines and feed to the *stdin* of the process
 - the mapper collects the line oriented outputs from the *stdout* of the process
 - converts each line into a key/value pair
 - By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) will be the value
 - e.g. "this is the key\tvalue is the rest\n"
 - If there is no tab character in the line, then entire line is considered as key and the value is null (!)

Command line example

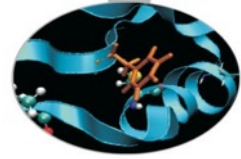


```
$ hadoop jar $HADOOP_HOME/hadoop-streaming.jar \  
  -input myInputDirs \  
  -output myOutputDir \  
  -mapper org.apache.hadoop.mapred.lib.IdentityMapper \  
  -reducer /bin/wc
```

Command line python



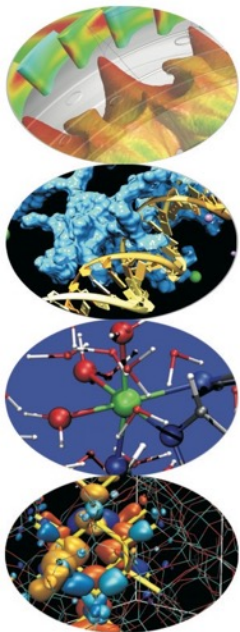
```
$ hadoop jar hadoop-streaming-1.2.1.jar \  
  -input input_dir/ \  
  -output output_dir/ \  
  -mapper mapper.py \  
  -file mapper.py \  
  -reducer reducer.py \  
  -file reducer.py
```



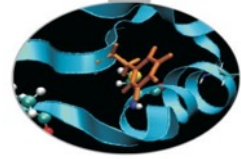
Hadoop streaming

Python code

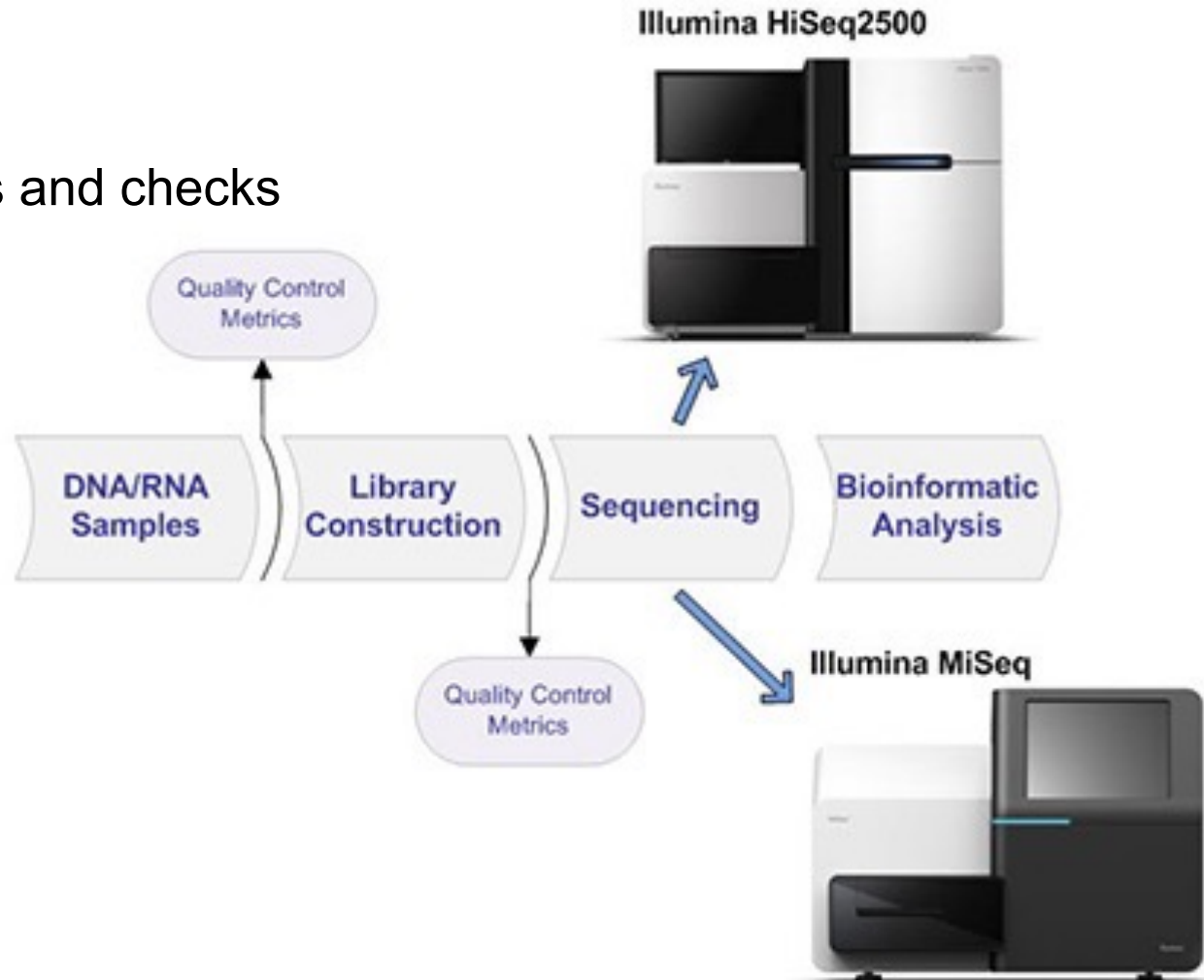
(based on a real scientific case)



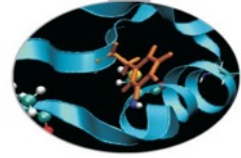
The "situation"



- DNA analysis
 - New generation sequencing (NGS) platform
 - High throughput
 - Big text files
 - Lots of statistics and checks

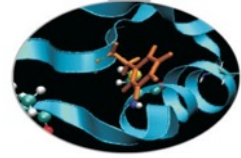


One Sequence



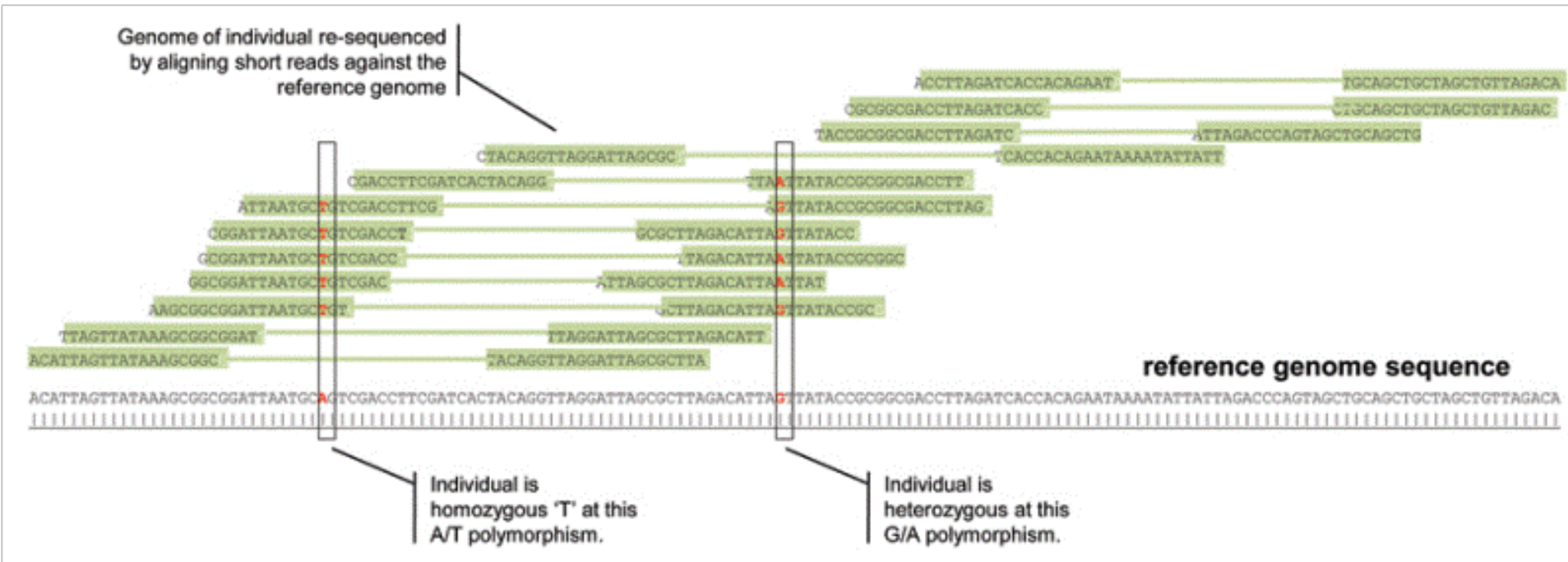
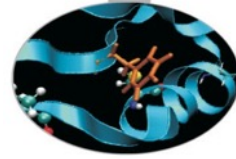
```
@HWI-EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1  
TTAATTGGTAAATAAATCTCCTAATAGCTTAGATNTTACCTTNNNNNNNNN  
+HWI-EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1  
efcffffcfeeffcfffffddf`feed]` ]_Ba_^__[YBBBBBBBBB
```

Mapping



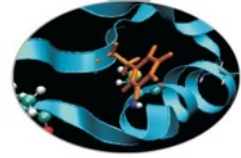
- Input sequences
 - Short-reads
- Alignment
 - Map input to a reference genome
 - e.g. Against latest release of reference human genome
 - output
 - **aligned sequences**
 - Standards (*SAM* format)

Mapping



Mapping coordinates:
Chromosome, Start position, End position

SAM specifications

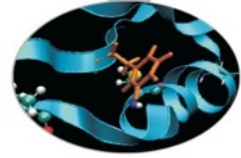


<http://samtools.sourceforge.net/samtools.shtml#5>

- Sequence Alignment/Map (SAM) format is TAB-delimited
- The header lines are started with the '@' symbol
- Each line is one aligned sequence

Col	Field	Description
1	QNAME	Query template/pair
2	FLAG	bitwise FLAG
3	RNAME	Reference sequence
4	POS	1-based leftmost
5	MAPQ	MAPPing Quality
6	CIAGR	extended CIGAR
7	MRNM	Mate Reference
8	MPOS	1-based Mate
9	TLEN	inferred Template
10	SEQ	query SEQUENCE on
11	QUAL	query QUALity
12+	OPT	variable OPTional

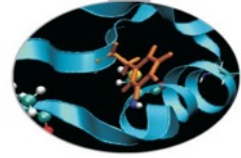
SAM header



```
$ head input.sam
```

```
@HD VN:1.4 G0:none S0:coordinate
@SQ SN:chrM LN:16571
@SQ SN:chr1 LN:249250621
@SQ SN:chr2 LN:243199373
@SQ SN:chr3 LN:198022430
@SQ SN:chr4 LN:191154276
@SQ SN:chr5 LN:180915260
@SQ SN:chr6 LN:171115067
@SQ SN:chr7 LN:159138663
@SQ SN:chr8 LN:146364022
```


First approach: split



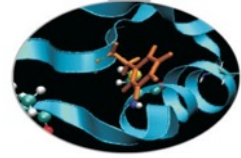
- I want to do something with each line
- Hadoop streaming works with text
 - Need to split my string base on a character separator
 - e.g. “ ” or “\t” or “,”

```
import sys
```

```
for line in sys.stdin:  
    line = line.strip()  
    pieces = line.split("\t")
```

```
print pieces #prints ["piece1", "piece2"...]
```

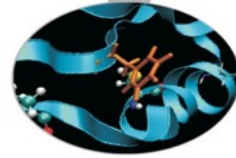
Debug



- Before submitting the Hadoop job
 - Make sure your scripts have no errors
 - *Do mapper and reducer scripts actually work?*
- This is just a matter of running them through pipes
 - On a *little bit* of sample data
 - **cat** or **head** linux bash commands
 - with **pipes**

```
$ cat $file | python mapper.py | sort | python reducer.py
```


Tutorial



Mapper

```
TAB = "\t"

# Cycle current streaming data
for line in sys.stdin:

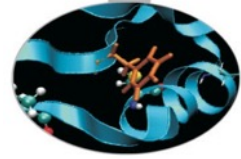
    # Clean input
    line = line.strip()
    # Skip SAM/BAM headers
    if line[0] == "@":
        continue

    # Use data
    pieces = line.split(TAB)
    mychr = pieces[2]
    mystart = int(pieces[3])
    myseq = pieces[9]
    print mychr, mystart.__str__()
    exit()
```

```
root@5a57dab12438:/course-exercises
#cat data/ngs/input.sam \
    | python ngs/hs/hsmapper.py

chrM 14
```

Tutorial



Mapper

```
# Cycle current streaming data
for line in sys.stdin:

    # Use data
    pieces = line.split(TAB)

[...]
```

```
    mystop = mystart + len(myseq)

    # Each element with coverage
    for i in range(mystart, mystop):
        results = [mychr+SEP+i.__str__(), "1"]
        print TAB.join(results)
    exit()
```

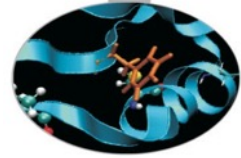
```
root@5a57dab12438:/course-exercises
#cat data/ngs/input.sam \
    | python ngs/hs/hsmapper.py

chrM 14 1

[...]
```

```
chrM:79 1
chrM:80 1
chrM:81 1
chrM:82 1
chrM:83 1
chrM:84 1
chrM:85 1
chrM:86 1
chrM:87 1
chrM:88 1
```

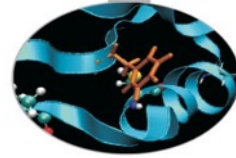
Between Map and Reduce



● Shuffle step

- A lot happens, transparent to the developer
- Mappers's output is transformed and distributed to the reducers
 - All key/value pairs are sorted before sent to reducer function
 - Pairs sharing the same key are sent to the same reducer
 - If you encounter a key that is different from the last key you processed, you know that previous key will never appear again
 - If your keys are all the same
 - only use one reducer and gain no parallelization
 - come up with a more unique key if this happens

Tutorial



Reducer

```
# Cycle current streaming data
for line in sys.stdin:

    # Clean input
    line = line.strip()
    value, count = line.split(TAB)
    count = int(count)

    # if this is the first iteration
    if not last_value:
        last_value = value

    # if they're the same, log it
    if value == last_value:
        value_count += count
    else:
        # state change
        result = [last_value, value_count]
        print TAB.join(str(v) for v in result)
        last_value = value
        value_count = 1

# LAST ONE after all records have been received
print TAB.join(str(v) for v in [last_value,
value_count])
```

```
root@5a57dab12438:/course-exercises
```

```
#cat data/ngs/input.sam \  
| python ngs/hs/hsmapper.py \  
| sort \  
| python ngs/hs/hsreducer.py
```

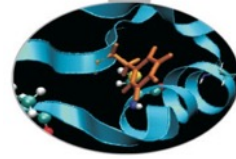
```
chr10:103270024 1  
chr10:103270025 1
```

```
[...]
```

```
chr21:48110960 2  
chr21:48110961 3  
chr21:48110962 3  
chr21:48110963 6  
chr21:48110964 6  
chr21:48110965 6  
chr21:48110966 6  
chr21:48110967 8  
chr21:48110968 8  
chr21:48110969 11
```

```
[...]
```

Debug: fail



```
$ cat $file | python mapper.py | sort | python reducer.py \  
1> out.txt 2> out.log
```

```
$ head out.log
```

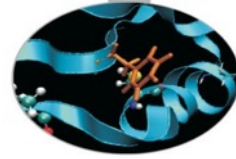
```
File "ngs/hs/hsmapper.py", line 16  
    line = line.strip()  
                    ^
```

```
SyntaxError: invalid syntax
```

```
$ head out.txt
```

```
None 0
```

Debug: ok



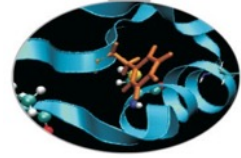
```
$ cat $file | python mapper.py | sort | python reducer.py \  
1> out.txt 2> out.log
```

```
$ head out.log
```

```
$ head out.txt
```

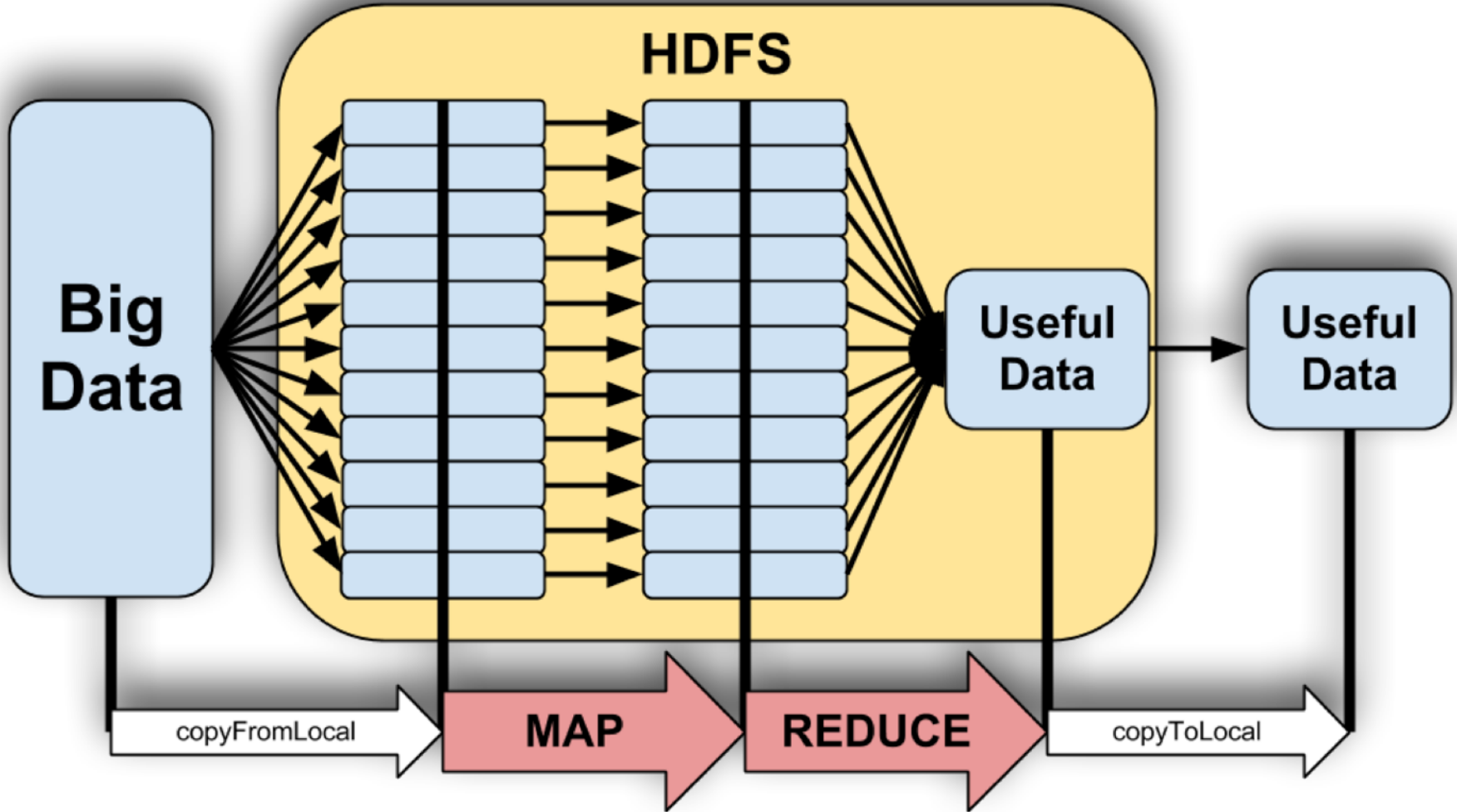
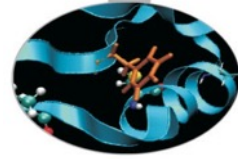
```
chrM:988      3  
chrM:989      3  
chrM:99       5  
chrM:990      4  
chrM:991      4  
chrM:992      4  
chrM:993      4
```

Switching to Hadoop

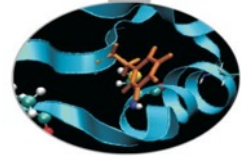


- A working python code tested on pipes should work with Hadoop Streaming
- To make this work we need to handle copy of input and output file
 - inside the Hadoop FS
- Also the job tracker logs will be found inside HDFS
- We are going to build a bash script to make our workflow

Workflow

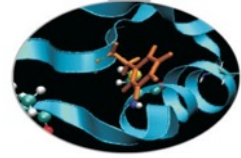


Steps overview



1.	Preprocessing of data
2.	Mapping
3.	Shuffling
4.	Reducing
5.	Postprocessing of data

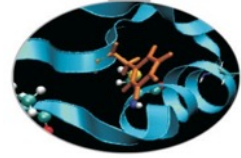
Preprocessing



HDFS commands to interact with Hadoop file system

- Create dir
 - `hadoop fs -mkdir`
- Copy file
 - `hadoop fs -put`
- Check if file is there
 - `hadoop fs -ls`
- Remove recursively data
 - `hadoop fs -rmr`

Preprocessing



```
bin="$HADOOP_HOME/bin/hadoop
```

```
# Clean previous output
```

```
tmp=`$bin fs -rmr $out 2>&1`
```

```
# Create dir and copy file
```

```
tmp=`$bin fs -rmr $datadir 2>&1`
```

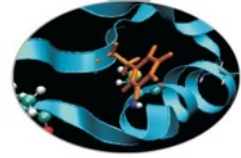
```
tmp=`$bin fs -mkdir $datadir 2>&1`
```

```
tmp=`$bin fs -put $file $datadir/ 2>&1`
```

```
# Cleaning old logs (warning!)
```

```
tmp=`rm -rf $userlogs/* 2>&1`
```

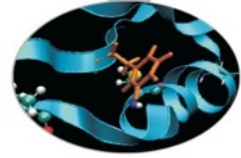
Pipe your python scripts



Hadoop Streaming needs “*binaries*” to execute

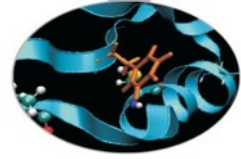
- You need to specify interpreter inside the script
 - `#!/usr/bin/env python`
- Make the script executable
 - `chmod +x hs*.py`

Command line



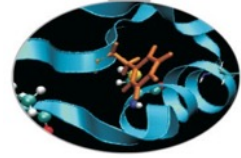
```
hadoop jar hadoop-streaming-1.2.1.jar \  
  -input input_dir/ \  
  -output output_dir/ \  
  -mapper mapper.py \  
  -file mapper.py \  
  -reducer reducer.py \  
  -file reducer.py
```

Postprocessing



```
# Check if last command failed
if [ $? -ne 0 ]; then
    # Easier cli debug
    echo "Failed..."
    sleep 3
    cat $userlogs/job_*/**/* | less
fi
```

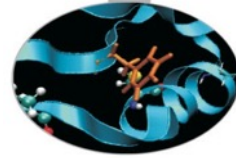
Postprocessing



```
if [ $? == "0" ]; then
    echo "Output is:"
    $bin fs -ls $out

    $bin fs -get $out/part-00000 $d/test.out
    echo "Copied output to $d/test.out"
else
    echo "Failed..."
    sleep 3
    cat $userlogs/job_*/**/* | less
fi
```

Tutorial: final launch

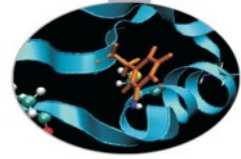


```
root@65bc152d5565:/course-exercises# bash ngs/hs/hstream.sh
Data init completed
packageJobJar: [ngs/hs/hsmapper.py, ngs/hs/hsreducer.py, /tmp/hadoop-root/hadoop-
unjar4307936164982400895/] [] /tmp/streamjob5710848067265482260.jar tmpDir=null

14/12/10 23:22:58 INFO mapred.FileInputFormat: Total input paths to process : 1
14/12/10 23:22:58 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-root/mapred/local]
14/12/10 23:22:58 INFO streaming.StreamJob: Running job: job_201412101439_0009
14/12/10 23:22:58 INFO streaming.StreamJob: To kill this job, run:
14/12/10 23:22:58 INFO streaming.StreamJob: /usr/local/hadoop/bin/hadoop job -
Dmapred.job.tracker=65bc152d5565:9001 -kill job_201412101439_0009
14/12/10 23:22:58 INFO streaming.StreamJob: Tracking URL: http://65bc152d5565:50030/
jobdetails.jsp?jobid=job_201412101439_0009
14/12/10 23:22:59 INFO streaming.StreamJob: map 0% reduce 0%
14/12/10 23:23:10 INFO streaming.StreamJob: map 5% reduce 0%
14/12/10 23:23:13 INFO streaming.StreamJob: map 8% reduce 0%

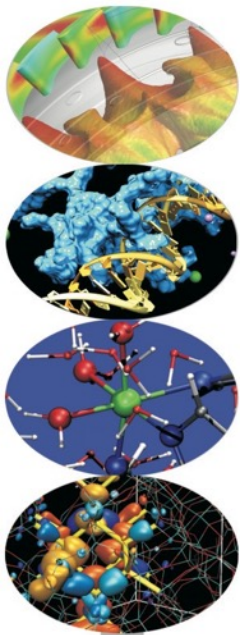
[...]

14/12/10 23:26:13 INFO streaming.StreamJob: map 100% reduce 96%
14/12/10 23:26:16 INFO streaming.StreamJob: map 100% reduce 98%
14/12/10 23:26:19 INFO streaming.StreamJob: map 100% reduce 100%
14/12/10 23:26:22 INFO streaming.StreamJob: Job complete: job_201412101439_0009
14/12/10 23:26:22 INFO streaming.StreamJob: Output: outdata
Output is:
Found 3 items
-rw-r--r-- 1 root supergroup 0 2014-12-10 23:26 /user/root/outdata/_SUCCESS
drwxr-xr-x - root supergroup 0 2014-12-10 23:22 /user/root/outdata/_logs
```

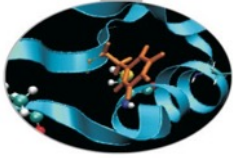



Hadoop streaming

(Intermission)



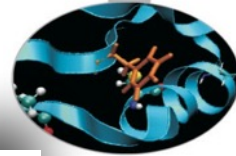
The real world



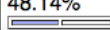
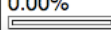
This entire example is actually **very simple** ...

- It illustrates the concepts quite neatly but...
- ...splitting text from ~ 1 kb file is useless if done through Hadoop
 - By default Hadoop chunks to mappers in increments of 64 MB
 - Hadoop is meant to handle multi-gigabyte files!

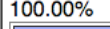
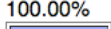
Job tracker



Running Jobs

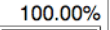
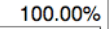
Jobid	Started	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
job_201412132122_0001	Sat Dec 13 21:23:33 UTC 2014	NORMAL	root	streamjob8491557782315478474.jar	48.14% 	2	0	0.00% 	1	0

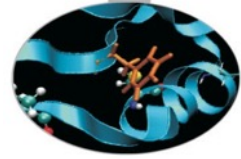
Completed Jobs

Jobid	Started	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
job_201412132122_0001	Sat Dec 13 21:23:33 UTC 2014	NORMAL	root	streamjob8491557782315478474.jar	100.00% 	2	2	100.00% 	1	1

Hadoop job_201412132122_0001 on [bb96d757cdc7](#)

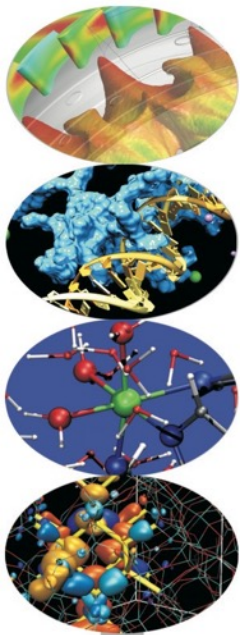
User: root
Job Name: streamjob8491557782315478474.jar
Job File: hdfs://bb96d757cdc7:9000/tmp/hadoop-root/mapred/staging/root/staging/job_201412132122_0001/job.xml
Submit Host: bb96d757cdc7
Submit Host Address: 172.17.0.44
Job-ACLs: All users are allowed
Job Setup: [Successful](#)
Status: Succeeded
Started at: Sat Dec 13 21:23:33 UTC 2014
Finished at: Sat Dec 13 21:25:37 UTC 2014
Finished in: 2mins, 4sec
Job Cleanup: [Successful](#)

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00% 	2	0	0	2	0	0 / 0
reduce	100.00% 	1	0	0	1	0	0 / 0

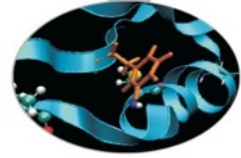


Hadoop streaming

Compression



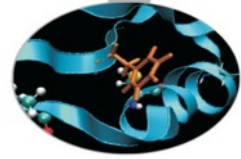
Zip, bzip, gzip



```
$ du -shc data/ngs/*
```

- 17M data/ngs/input.bz2
 - 21M data/ngs/input.tgz
 - 104M data/ngs/input.sam
-
- Streamed content may be decompressed by **Hadoop**
 - via command line option
 - `-jobconf stream.recordreader.compression=bz2`

BAM format



4 The **BAM** Format Specification

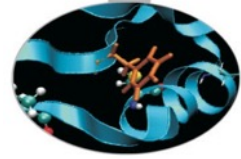
4.1 The **BGZF** compression format

BGZF is block compression implemented on top of the standard gzip file format. The goal of BGZF is to provide good compression while allowing efficient random access to the BAM file for indexed queries. The BGZF format is ‘gzip compatible’, in the sense that a compliant gzip utility can decompress a BGZF compressed file.⁹

A BGZF file is a series of concatenated BGZF blocks. Each BGZF block is itself a spec-compliant gzip archive which contains an “extra field” in the format described in RFC1952. The gzip file format allows the inclusion of application-specific extra fields and these are ignored by compliant decompression implementation. The gzip specification also allows gzip files to be concatenated. The result of decompressing concatenated gzip files is the concatenation of the uncompressed data.

Each BGZF block contains a standard gzip file header with the following standard-compliant extensions:

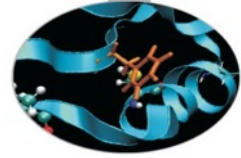
Binary SAM (BAM)



```
$ du -shc data/ngs/*
```

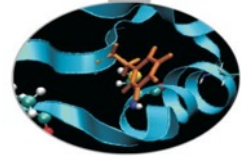
- 17M data/ngs/input.bz2
- 21M data/ngs/input.tgz
- 23M data/ngs/input.bam
- 23M data/ngs/input.bam.bz2
- 104M data/ngs/input.sam

samtools & pysam



- **samtools** is the suite of open utilities (written in C)
 - to handle SAM/BAM format conversion, viewing and manipulation
- **pysam** is the python module that implements samtools actions
 - it also reads python object directly from SAM or BAM files
 - does not work with Hadoop distribution of data stream...

BAM streaming failure



```
stdout:0 -1  
stderr:0 -1  
syslog:0 -1
```

Traceback (most recent call last):

```
File "/tmp/hadoop-root/mapred/local/taskTracker/root/jobcache/  
job_201412101439_0007/attempt_201412101439_0007_m_000000_0/work/./  
hsmapper_deco.py", line 32, in <module>
```

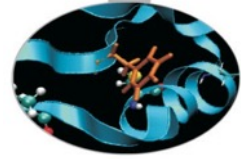
```
    samstream = pysam.AlignmentFile("-", "rb")
```

```
File "calignmentfile.pyx", line 302, in  
pysam.calignmentfile.AlignmentFile.__cinit__ (pysam/calignmentfile.c:4834)
```

```
File "calignmentfile.pyx", line 485, in  
pysam.calignmentfile.AlignmentFile._open (pysam/calignmentfile.c:6789)
```

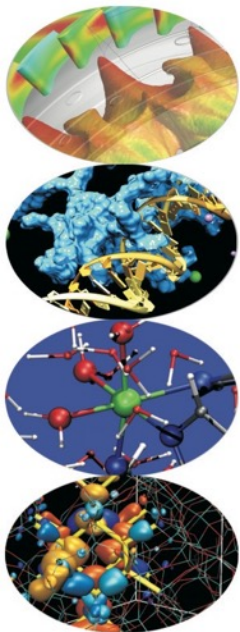
```
ValueError: file header is empty (mode='rb') - is it SAM/BAM format?
```

```
java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess  
failed with code 1
```

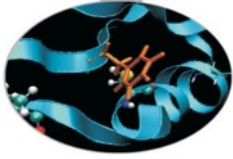


Hadoop streaming

Final thoughts

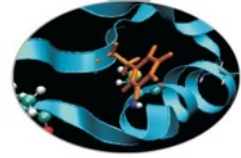


Pros



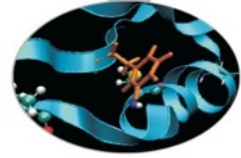
- Provides options to write MapReduce jobs in other languages
 - One of the best examples of flexibility available to MapReduce
- Fast
- Simple
- Close to the original standard Java API power
- Even executables can be used to work as a MapReduce job (!)

Where it really works



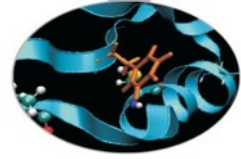
- When the developer do not have knowhow of Java
 - Write Mapper/Reducer in **any** scripting language
 - Faster

Cons



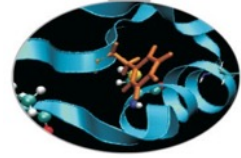
- Force scripts in a Java VM
 - Although free overhead
- The program/executable should be able to take input from STDIN and produce output at STDOUT
 - Restrictions on the input/output formats
 - Does not take care of input and output file and directory preparation
 - User have to implement hdfs commands “hand-made”

Where it falls short

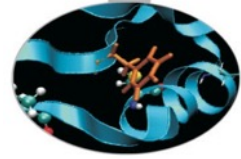


- No pythonic way to work the MapReduce code
 - Because it was not written specifically for python

Summing all up

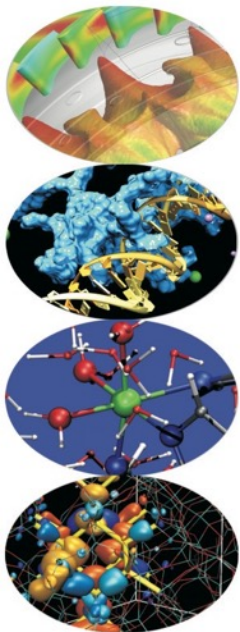


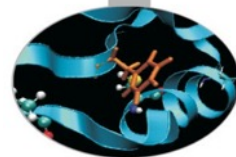
- Hadoop streaming handles Hadoop in almost a *classic* manner
 - Wrap any executable (and script)
 - Wrap python scripts
 - Runnable on a cluster
 - using a non-interactive, all-encapsulated job



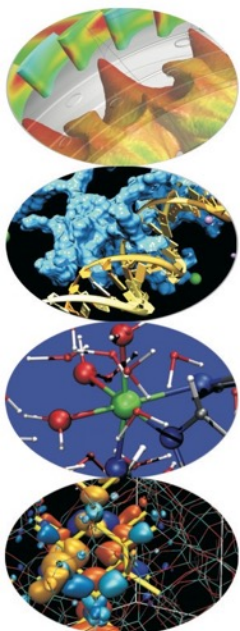
Hadoop streaming

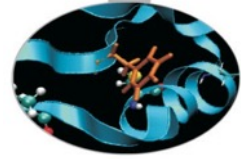
Well done!





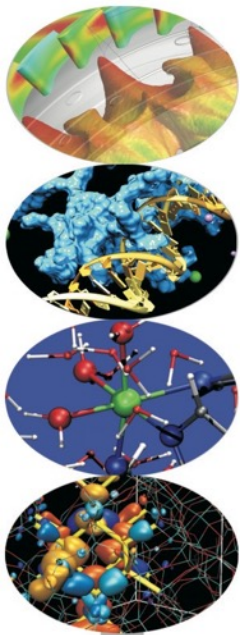
Digest informations...



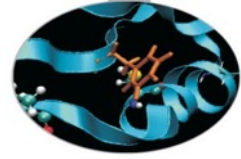


mrjob

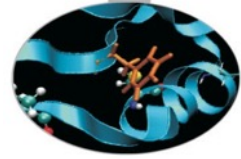
A more pythonic MapReduce



mrjob - our schedule



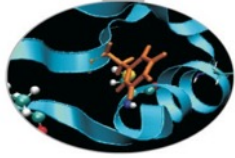
- Concepts
 - What is it
 - How it works
 - Hadoop steps
- Hands on
 - First job Word Count
 - Use case: NGS coverage
 - Command line parameters (ipython notebook *live*)
 - Running inline, local, hadoop
 - Optimizations
- Conclusions
 - At the end of the day



mrjob: what

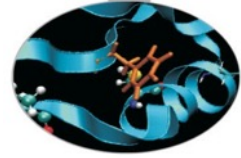
- “*Easiest route to Python programs that run on Hadoop*”
 - install with: “pip install mrjob”
- Running modes
 - Test your code *locally* without installing Hadoop
 - or run it on a cluster of your choice
 - also extensive integration with Amazon Elastic MapReduce
- Modes set up
 - same code with local, Hadoop, EMR
 - easy to run your job in the cloud as it is to run it on laptop

mrjob: how



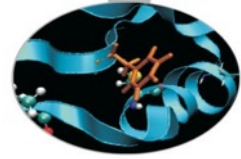
- Python module built on top of Hadoop Streaming
 - jar opens a subprocess to your code
 - sends it input via stdin
 - gathers results via stdout.
 - Wrap HDFS pre and post processing if hadoop exists
 - a consistent interface across every environment it supports
 - automatically serializes/deserializes data flow out of each task
 - **JSON**: `json.loads()` and `json.dumps()`

mrjob: steps



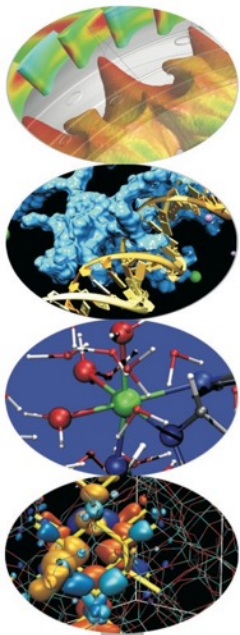
- mrjob can be configured to run different steps
 - for each step you can specify which part has to be executed
 - and the method to use within the class you wrote

```
def steps(self):  
    return [  
        MRStep(mapper=self.mapper_get_words,  
                combiner=self.combiner_count_words,  
                reducer=self.reducer_count_words),  
        MRStep(reducer=self.reducer_find_max_word)  
    ]
```

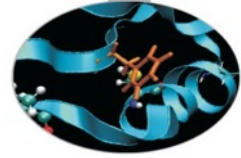


mrjob

Getting hands dirty



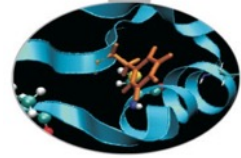
mrjob - hands on



- A job is defined by a class extended from MRJob package
 - Contains methods that define the steps of a Hadoop job
- A “step” consists of a mapper, a combiner, and a reducer.
 - All of those are optional, though you must have at least one

```
class myjob(MRJob):  
    def mapper(self, _, line):  
        pass  
    def combiner(self, key, values):  
        pass  
    def reducer(self, key, values):  
        pass  
    def steps(self):  
        return [ MRStep(mapper=self.mapper, ... ), ... ]
```


Word Count



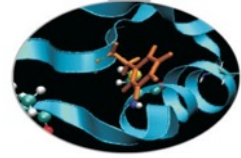
The mapper() method takes a key and a value as args

- E.g. key is ignored and a single line of text input is the value
- Yields as many key-value pairs as it likes
 - Warning: **yield != return**
 - **yield** return a *generator*, the one you usually use with
`print i; for i in generator`
- Example

```
def mygen():  
    for i in range(1,10):  
        # THIS IS WHAT HAPPENS INSIDE MAPPER  
        yield i, "value"
```

```
for key, value in mygen():  
    print key, value
```

Word Count



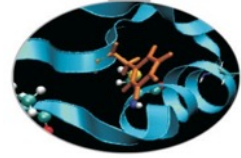
- The reduce() method takes a key and an iterator of values
 - Also yields as many key-value pairs as it likes
 - E.g. it sums the values for each key
 - Represent the numbers of characters, words, and lines in the initial input

- Example

```
def mygen():  
    for i in range(1,10):  
        yield i, "value"
```

```
for key, value in mygen():  
    # THIS IS WHAT HAPPENS INSIDE A REDUCER  
    print key, value
```

MRjob Word Count



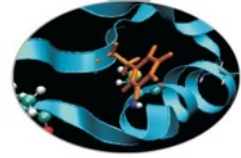
```
from mrjob.job import MRJob
class MRWordCount(MRJob):

    def mapper(self, key, line):
        for word in line.split(' '):
            yield word.lower(),1

    def reducer(self, word, occurrences):
        yield word, sum(occurrences)

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

MRjob: input



By default, output will be written to stdout.

```
$ python my_job.py input.txt
```

You can pass input via stdin

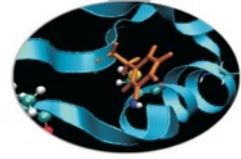
but be aware that mrjob will just dump it to a file first:

```
$ python my_job.py < input.txt
```

You can pass multiple input files, mixed with stdin (using the - character)

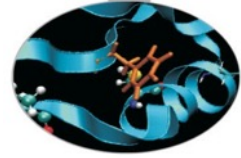
```
$ python my_job.py input1.txt input2.txt - <  
input3.txt
```

MRjob: input



- *By default, mrjob will run your job in a single Python process*
- *This provides the friendliest debugging experience...*
 - *...but it's not exactly distributed computing*
- You change the way the job is run with the `-r/--runner` option (`-r inline`, `-r local`, `-r hadoop`, or `-r emr`)
- Use `--verbose` to show all the steps

MRjob Word Count



```
$ git clone https://github.com/gfiameni/course-exercises.git
```

```
$ docker run -v ~/course-exercises:/course-exercises -it \  
  cineca/hadoop-mrjob:1.2.1 /etc/bootstrap.sh -bash
```

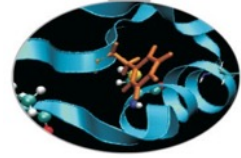
```
root$ source /root/mrjob0.4.2/bin/activate
```

```
(mrjob0.4.2)root$ show-exercises
```

```
(mrjob0.4.2)root$ python word_count.py data/txt/2261.txt.utf-8
```

```
[-r hadoop]
```


Simple example: categories



```
from mrjob.job import MRJob

class MRcoverage(MRJob):

    def mapper(self, _, line):

        if line[0] == "@":
            yield "header", 1
        else:
            yield "content", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == "__main__":
    MRcoverage.run()
```

job.py

```
root@5a57dab12438:/course-exercises
# source /root/mrjob0.4.2/bin/activate

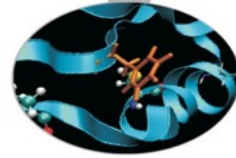
(mrjob0.4.2)# head -200 input.sam \
    | python job.py -r "inline"

using configs in /root/.mrjob.conf
creating tmp directory /tmp/job.root.
20141213.195716.080609
reading from STDIN

[...]

Streaming final output from /tmp/
job.root.20141213.195716.080609/output
"content" 103
"header" 97
removing tmp directory /tmp/job.root.
20141213.195716.080609
```


Yield the key



```
[...] #MAPPER else:
```

```
# Recover data
```

```
pieces = line.split("\t")
```

```
mychr = pieces[2]
```

```
mystart = int(pieces[3])
```

```
myseq = pieces[9]
```

```
mystop = mystart + len(myseq)
```

```
# For each piece of the sequence
```

```
for i in range(mystart, mystop):  
    yield str(i), 1
```

```
[...]
```

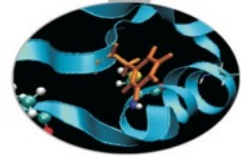
job.py

```
(mrjob0.4.2)# head -200 input.sam \  
| python job.py -r "inline"
```

```
[...]
```

```
"987"      3  
"988"      3  
"989"      3  
"99"       5  
"990"      4  
"991"      4  
"992"      4  
"993"      4  
"994"      4  
"995"      4  
"996"      4  
"997"      4  
"998"      4  
"999"      4
```

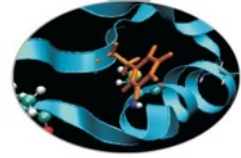
SAM columns: the hex flag



Each bit in the FLAG field is defined as the Hex table where the second column gives the string representation of the FLAG field.

Flag	Chr	Description
0x0001	p	the read is paired in sequencing
0x0002	P	the read is mapped in a proper pair
0x0004	u	the query sequence itself is unmapped
0x0008	U	the mate is unmapped
0x0010	r	strand of the query (1 for reverse)
0x0020	R	strand of the mate
0x0040	1	the read is the first read in a pair
0x0080	2	the read is the second read in a pair
0x0100	s	the alignment is not primary
0x0200	f	the read fails platform/vendor quality checks
0x0400	d	the read is either a PCR or an optical duplicate

Sequence strand



```
# Handle sequence STRAND
```

```
flag = int(pieces[1])
```

```
# Convert the flag to decimal, and check
```

```
# the bit in the 5th position from the right.
```

```
# 0x0010
```

```
if flag & 16:          # Strand forward
```

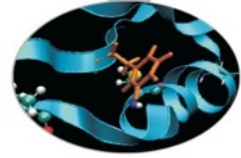
```
    mystop = mystart + len(myseq)
```

```
else:                 # Strand reverse
```

```
    mystop = mystart + 1
```

```
    mystart = mystop - len(myseq)
```

Chromosome code



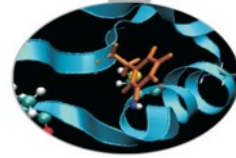
- String sorting could be a problem
 - we have chr1, chr11, chr12, [...] chr2, chr21
 - we want 01, 02, [...] 10, 11, [...] 21, 22, 88, 89
 - we could work on our string with python:

```
tmp = mychr[3:] #remove "chr" from the string
try:
    int(tmp)
    code = tmp.zfill(2) #padding
except ValueError:
    code = ord(tmp).__str__() #ascii code for letters
```

Or Hadoop streaming options?

- D mapred.output.key.comparator.class=
org.apache.hadoop.mapred.lib.KeyFieldBasedComparator
- D mapred.text.key.comparator.options=-n

Yield the right key



```
# Illumina 1.8+ Phred+33
PHRED_INIT = 33
PHRED_MIN = 0
PHRED_MAX = 41
PHRED_THRESHOLD = 20

[...] #MAPPER else:

# Recover data
pieces = line.split("\t")
flag = int(pieces[1])
myqc = pieces[10]

[...] FLAG (start and stop)

    for i in xrange(mystart, mystop):

        mypos = i - mystart
        cqc = ord(myqc[mypos]) - PHRED_INIT
        # Quality checks?

        label = code+SEP+str(i)+SEP+mychr
        yield label, 1
```

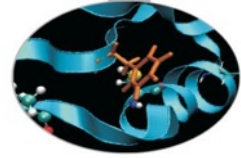
job.py

```
(mrjob0.4.2)# head -200 input.sam \  
| python job.py -r "inline"
```

[...]

```
"89:28735301:chrY"      8  
"89:28735302:chrY"      7  
"89:28735303:chrY"      7  
"89:28735304:chrY"      7  
"89:28735305:chrY"      7  
"89:28735306:chrY"      7  
"89:28735307:chrY"      7  
"89:28735308:chrY"      8  
"89:28735309:chrY"      8  
"89:28735310:chrY"      8  
"89:28735311:chrY"      7  
"89:28735312:chrY"      7  
"89:28735313:chrY"      7  
"89:28735314:chrY"      7  
"89:28735315:chrY"      6
```

Two jobs at once *also counting the letters*



```
# Compute quality value
```

```
# which should be in the range 0..40
```

```
current_qc = ord(myqc[mypos]) - PHRED_INIT
```

```
# Check Phred33+ coding
```

```
if current_qc < PHRED_MIN or current_qc > PHRED_MAX:
```

```
    raise Exception("Wrong encoding 'Sanger' Phred33+...!\n" + \
        "Found " + current_qc.__str__() + " at " + \
        mychr + " Pos " + i.__str__())
```

```
# It will be used to skip bad sequencing
```

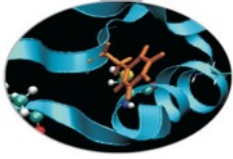
```
if current_qc > PHRED_THOLD:
```

```
    label = code + SEP + i.__str__() + SEP + mychr
    current_letter = myseq[mypos]
```

```
yield label, 1
```

```
yield label + SEP + current_letter, 1
```

Final code

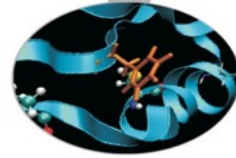


The code is available as the exercise:

`ngs/mrjob/job.py`

from the github project

Tutorial: final launch



```
(mrjob0.4.2) root@5a57dab12438:/course-exercises# python ngs/mrjob/job.py data/ngs/input.sam
```

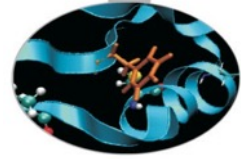
```
using configs in /root/.mrjob.conf
creating tmp directory /tmp/job.root.20141213.210511.414254
writing to /tmp/job.root.20141213.210511.414254/step-0-mapper_part-00000
Counters from step 1:
  (no counters found)
writing to /tmp/job.root.20141213.210511.414254/step-0-mapper-sorted
> sort /tmp/job.root.20141213.210511.414254/step-0-mapper_part-00000
writing to /tmp/job.root.20141213.210511.414254/step-0-reducer_part-00000
Counters from step 1:
  (no counters found)
Moving /tmp/job.root.20141213.210511.414254/step-0-reducer_part-00000 -> /tmp/job.root.
20141213.210511.414254/output/part-00000
Streaming final output from /tmp/job.root.20141213.210511.414254/output
removing tmp directory /tmp/job.root.20141213.210511.414254
```

OUT

```
"01:10000:chr1"      2
"01:10001:chr1"      2
"01:10002:chr1"      2
"01:10003:chr1"      2
"01:10004:chr1"      2
"01:10005:chr1"      2
```

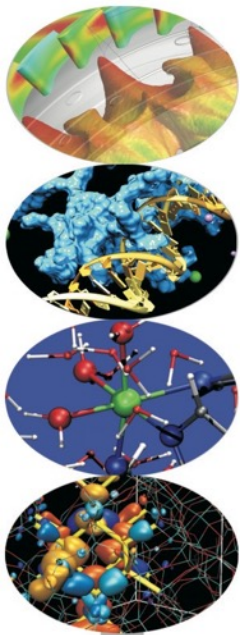
TOTAL LINES OF OUTPUT

```
7643091
```

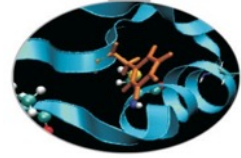


mrjob

Getting expert



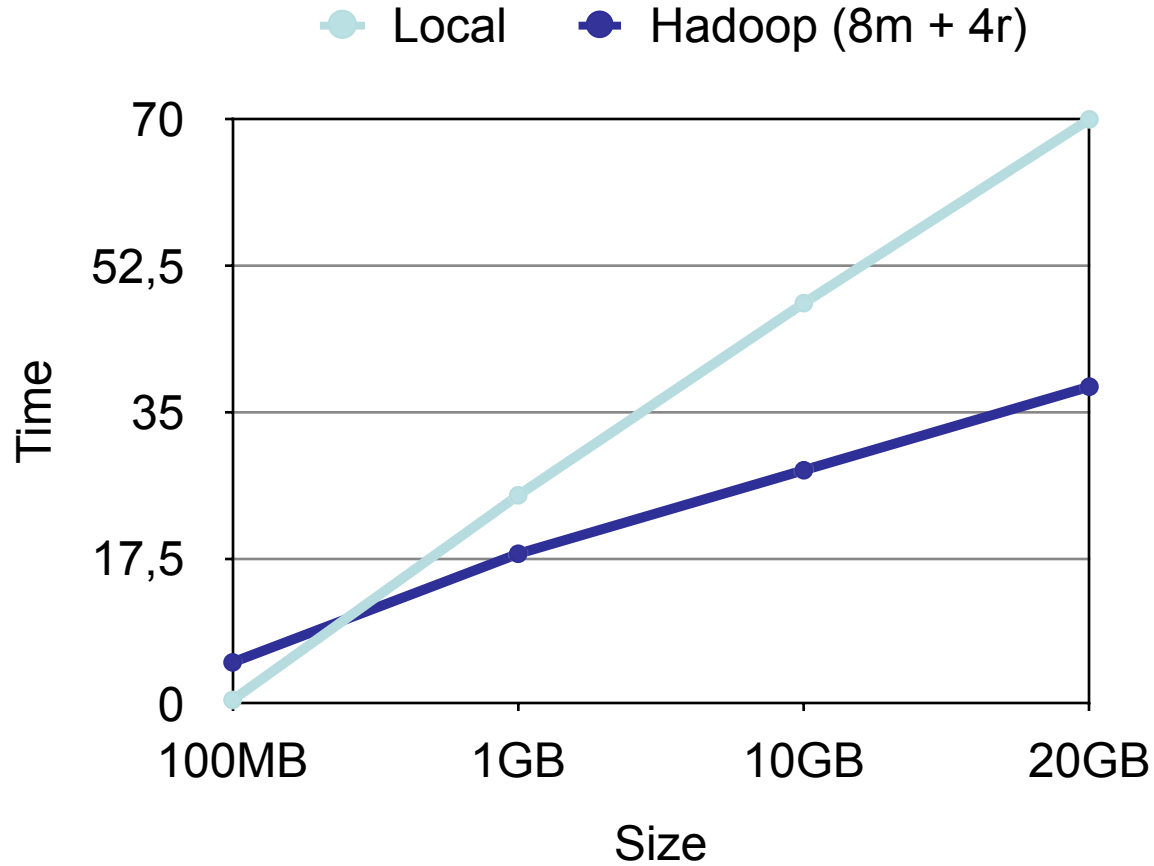
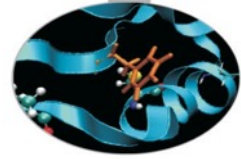
Inline or Local: debug



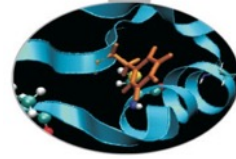
From mrjob docs

- mrjob.inline - debugger-friendly local testing
- class mrjob.inline.InlineMRJobRunner
 - Runs an MRJob in the same process
 - easy to attach a debugger.
 - This is the default way to run jobs
 - you'll spend some time debugging your job before you're ready to run it on EMR or Hadoop
 - **To more accurately simulate your environment** prior to running on Hadoop/EMR, use `-r local`
 - LocalMRJobRunner
- *Suggestion*: break down the map and reduce steps into a sequence of smaller functions
 - write *unittests* for each small step

Local or Hadoop



Encapsulation



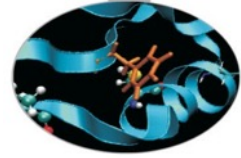
```
from job import MRcoverage

if __name__ == '__main__':

    # Create object
    mrjob = MRcoverage(args=[
        '-r', 'hadoop',
        '--jobconf=mapreduce.job.maps=10',
        '--jobconf=mapreduce.job.reduces=4'
    ])

    # Run
    mrjob.make_runner()
    runner.run()
```

Encapsulation



```
# Run and handle output
```

```
with mrjob.make_runner() as runner:
```

```
    # Redirect hadoop logs to stderr
```

```
    log_to_stream("mrjob")
```

```
    # Execute the job
```

```
    runner.run()
```

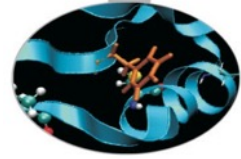
```
# Do something with stream output
```

```
for line in runner.stream_output():
```

```
    key, value = mrjob.parse_output_line(line)
```

```
    print key, value
```

Encapsulation



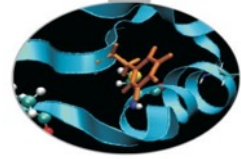
- Mrjob Runner

The code is available as the exercise:

```
ngs/mrjob/runner.py
```

from the github project

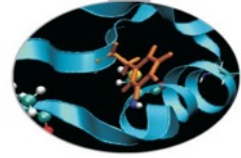
Optimization



- *Read the docs!*
- Input Output formats
 - Pickle?

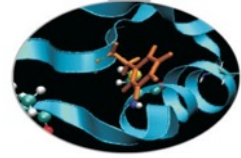
```
class MRcoverage(MRJob):  
    """ Implementation of a job MapReduce """  
  
    # Optimization on internal protocols  
    INTERNAL_PROTOCOL = PickleProtocol  
    OUTPUT_PROTOCOL = PickleProtocol
```


Multiple input in real science



- Multiple input is an easy task for Hadoop streaming
 - Works with the same run on one or a list of files
- **Paired end** sequencing
 - Sequence two times the same sample
 - Get the two stranded pair in two separated files
 - Alignment tools are able to couple the separated pair
 - Becomes a flag inside the BAM
- **Biological duplicates**
 - Sequencing dna replicated in same individual/tissue
 - Should be analyzed all together

Multiple step



- Max per group (e.g. chromosome)

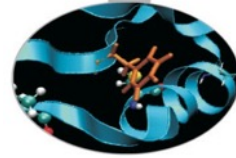
```
def reducer(self, key, values):  
    yield key, sum(values)
```

```
def set_max_key(self, key, values):  
    m = re.search(':(chr.*)$', key)  
    if m != None:  
        yield m.group(1), values
```

```
def get_max(self, key, values):  
    yield key, max(values)
```

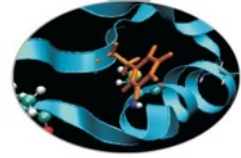
```
def steps(self):  
    """ Steps required from our MapReduce job """  
    return [  
        MRStep(mapper=self.mapper, reducer=self.reducer),  
        MRStep(mapper=self.set_max_key, reducer=self.get_max)  
    ]
```

Multiple step



```
# head -3500 data/ngs/input.sam | python ngs/mrjob/runner.py -  
creating tmp directory /tmp/job.root.20141214.001519.478788  
reading from STDIN  
  
[...]  
  
Streaming final output from /tmp/job.root.20141214.001519.478788/output  
  
chr1 101  
chrM 16
```

And much more!

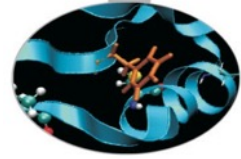


What if you can handle a database inside the MapReduce?

```
class SqliteJob(MRJob):  
  
    def mapper_init(self):  
        # make sqlite3 database available to mapper  
        self.sqlite_conn = sqlite3.connect(self.options.database)
```

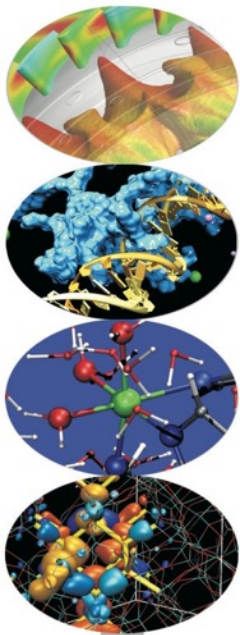
It works in every condition

```
$ python sqlite_job.py -r local --database=/etc/my_db.sqlite3  
$ python sqlite_job.py -r hadoop --database=/etc/my_db.sqlite3  
$ python sqlite_job.py -r hadoop --database=hdfs://my_dir/my_db.sqlite3  
$ python sqlite_job.py -r emr --database=/etc/my_db.sqlite3  
$ python sqlite_job.py -r emr --database=s3://my\_bucket/my\_db.sqlite3
```

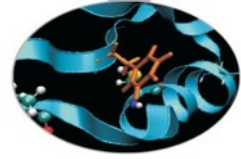


mrjob

Final thoughts

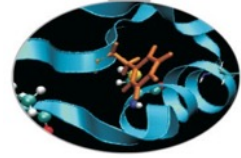


Pros



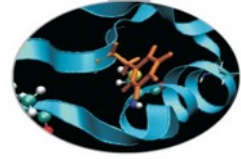
- More documentation than any other framework or library
- Write code in a single class (per Hadoop job)
 - Map and Reduce are single methods
 - Very clean and simple
- Advanced configuration
 - Configure multiple steps
 - Handle command line options inside the python code (!)
- Easily wrap output
 - Put data inside a database?

Pros



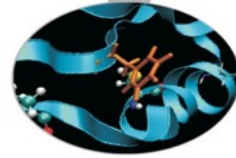
- No data copy required with HDFS
 - input and output
 - logs
 - May debug/exit directly with same code and commands
 - local or Hadoop (!)
 - Display runtime errors *traceback* on Hadoop
Exception: Command ['/root/mrjob0.4.2/bin/python', 'job.py', '--step-num=0', '--mapper', '/tmp/job.root.20141210.235014.836653/input_part-00000'] returned non-zero exit status 1: Traceback (most recent call last)
- **Forget about Hadoop cluster, fs and job tracker!**

Cons



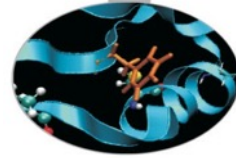
- Doesn't give you the same level of access to Hadoop APIs
 - Better: Dumbo and Pydoop
- Other libraries can be faster if you use typedbytes

Comparison



	Java	Streaming	mrjob	dumbo	hadoopy	pydoop	
Underlying framework	Hadoop	Hadoop Streaming	Hadoop Streaming	Hadoop Streaming	Hadoop Streaming	Hadoop Pipes	
Ease of installing framework	Easy with CDH (Whirr for cloud)	Easy with CDH (Whirr for cloud)	pip on client only	Build as egg, manually distribute to cluster	pip on client only (performance penalty) or manually install across cluster	Failed to build	
Documentation quality	Extensive/complex	Good	Very good	Poor	Good	Good	
Work with non-text objects	Yes	Manual ser/de	Built-in JSON	Built-in typedbytes	Built-in typedbytes	Unclear	
Data formats supported	All	Text/manual	Text, Repr (string), JSON, Pickle	Text, SequenceFile (typedbytes), any Java InputFormat	Text, SequenceFile (typedbytes), Pickle	Text, SequenceFile for I/O (but unclear)	
Implement custom SerDe	Yes	manual	Yes	Yes	No	No	
Integrate with arbitrary Java classes	Yes	Yes	Experimental	Yes	Unclear	Unclear	
Multistep MapReduce workflows	Yes, but awkward	No	Yes	Yes	No	No	
Implement Partitioner in Python		No	No	No	No	Yes	
HBase integration	Yes	No	No	Experimental / unofficial	Experimental	No	
Support for AWS/EMR	Manual	Yes; EMR exposes Streaming API	Yes; flawlessly integrated through boto	Manually setup EC2 cluster	Setup EC2 with supplied Whirr config	No	
Actively developed			Very	Somewhat	Yes	Yes	
Commits in last year				1063	19	167	272
SLOC add+delete in last year				66094	1314	75091	172223
First commit				10/13/2010	6/15/2008	10/17/2009	3/10/2009
Sponsored?			Yelp		Individual (Last.fm?)	Individual	CRS4
Hosted on			GitHub		GitHub	GitHub	Sourceforge
License			Apache v2.0		Apache v2.0	GPL v3	Apache v2.0

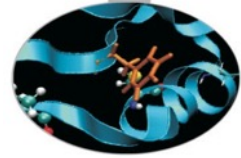
Comparison



	Java	Streaming*	mrjob*	dumbo*	hadoopy*
FILE: bytes read	22,726,677,381	0.94	1.34	2.55	1.97
FILE: bytes written	33,468,535,411	0.93	1.35	2.57	1.99
HDFS: bytes read	21,934,848,598	1.00	1.00	1.00	1.00
HDFS: bytes written	7,629,045,090	1.00	0.99	1.00	1.06
Map output bytes	12,978,686,993	0.91	1.40	2.11	2.11
Reduce shuffle bytes	11,336,515,993	0.92	1.35	2.53	1.97
Reduce input records	428,755,439	1.04	1.00	1.46	1.04
Time spent all maps (ms)	14,256,288	1.37	5.98	2.39	3.76
Time spent in all reduces (ms)	4,348,716	1.76	8.91	6.14	4.86
CPU time (ms)	14,016,540	1.17	4.68	3.68	2.76
Job run time (s)	1,074	1.54	7.31	3.90	4.20

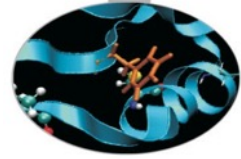
*Ratios are relative to Java values

Comparison



<http://blog.cloudera.com/blog/2013/01/a-guide-to-python-frameworks-for-hadoop/>

- All the Python frameworks look like pseudocode, which is a huge plus.
- Streaming appears to be the fastest Python solution, without any magic under the hood
 - it requires care when implementing the reducer
 - also when working with more complex objects.
- mrjob seems highly active, easy-to-use, and mature.
 - It makes multistep MapReduce flows easy, and can easily work with complex objects. It also works seamlessly with EMR.
 - But it appears to perform the slowest.
- The other Python frameworks appear to be somewhat less popular.



End of the day

Well done!

