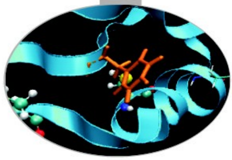
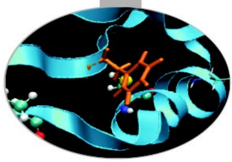


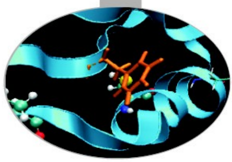
Welcome!



Emerging tools and techniques for massive data analysis

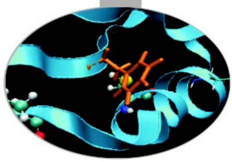
SuperComputing Applications and Innovation Department
15/16 December 2014
Bologna, Italy

Giuseppe Fiameni – g.fiameni@cineca.it



Why this workshop?

- Data are becoming more and more important
- Processing large data sets has become an issue for young researchers
- Many interesting technologies are emerging and entering the HPC domain
- HPC classic technologies, although only available solutions in many cases, have a steep learning curve which limits their wide adoption



Goals

During this two-day workshop you will learn:

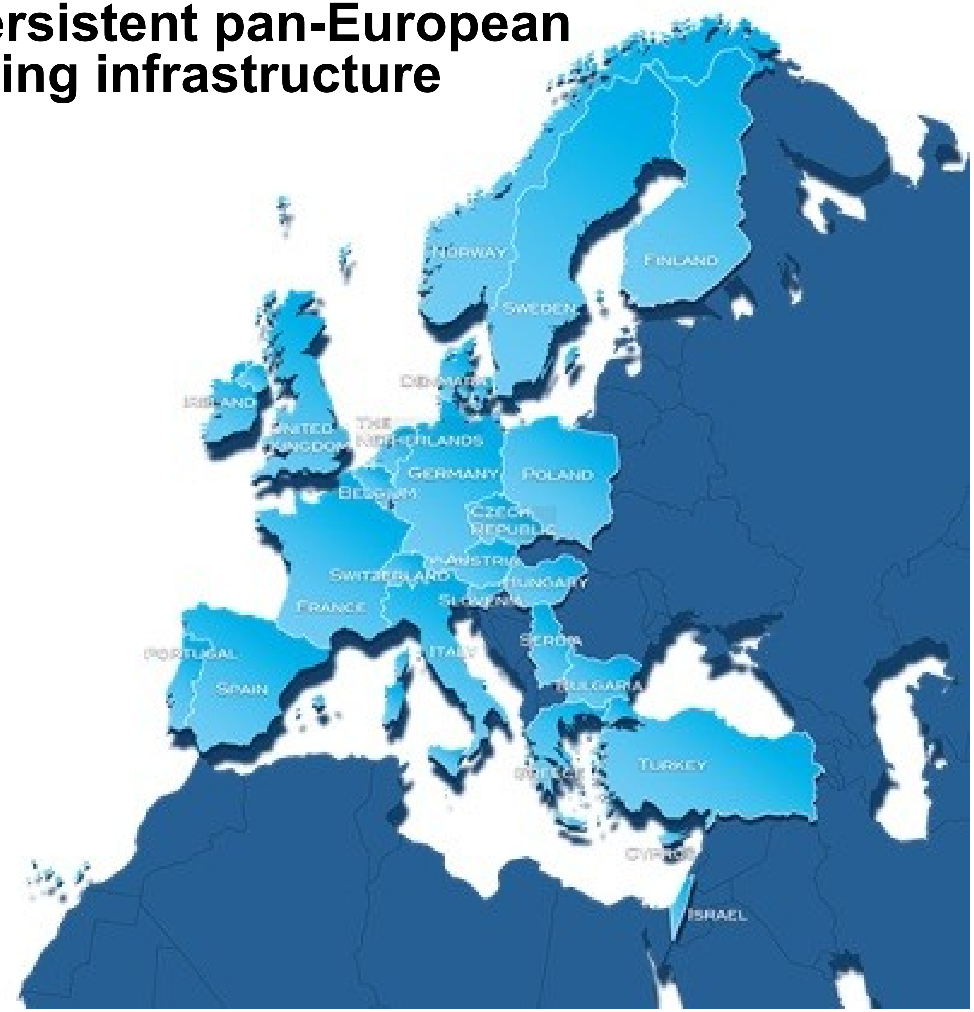
- the trends and challenges surrounding the BigData definition
- how the most relevant technologies and methods in this area (*Hadoop, Map-Reduce and Spark*) work
- how to structure and program your code using Python
- how to launch an Hadoop job both on a Linux container (Docker) and on Cineca HPC resources (PICO)

PRACE

PRACE

PARTNERSHIP FOR
ADVANCED COMPUTING IN EUROPE

PRACE aisbl, a persistent pan-European supercomputing infrastructure



25 members

4 hosting members:
France, Germany, Italy an

Enables world-class science
through **large scale simulations**

Offers HPC services on **leadin edge capability** systems

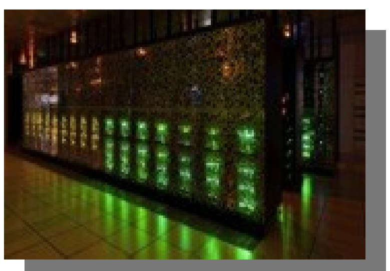
Awards its resources
through a **single** and **fair**
pan-European **peer reviev process for open research**

PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

PRACE's awards in 4 years

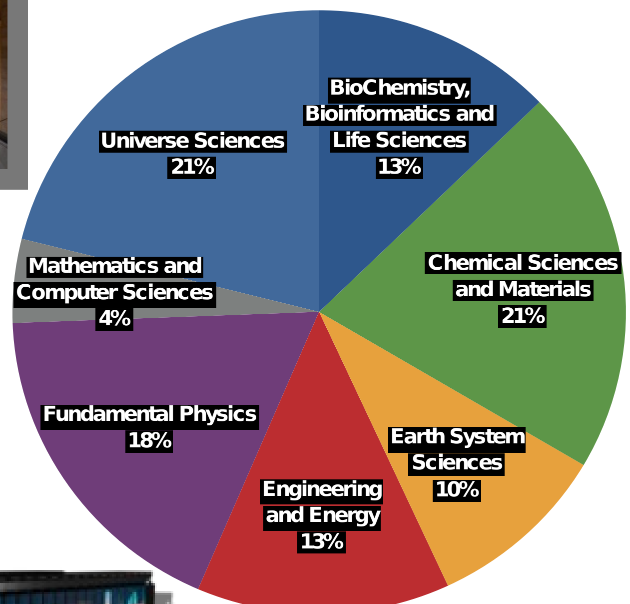
346 projects and 9.2 thousand million core-hours awarded

MareNostrum: IBM
BSC, Barcelona, Spain

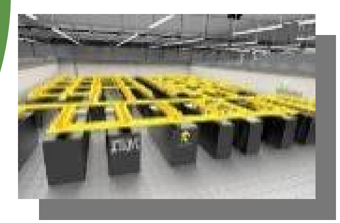


CURIE: Bull Bullx
GENCI/CEA
Bruyères-le-Châtel, France

HORNET: Cray
GAUSS/HLRS, Stuttgart, Germany



JUQUEEN: IBM
BlueGene/Q
GAUSS/FZJ
Jülich, Germany



SuperMUC: IBM
GAUSS/LRZ
Garching, Germany

FERMI: IBM BlueGene/Q
CINECA , Bologna, Italy

Scientific Steering Committee (SSC)

- It is composed of European leading researchers that are responsible for advice and guidance on all matters of a scientific and technical nature which may influence the scientific work carried out by the use of the Association’s resources.

- The SSC includes scientists from diverse areas: materials science, universe sciences, environmental science, particle physics, computational earth sciences, life sciences, plasma physics, computational physics, mathematics, astrophysics, chemistry and engineering

PRACE User Forum

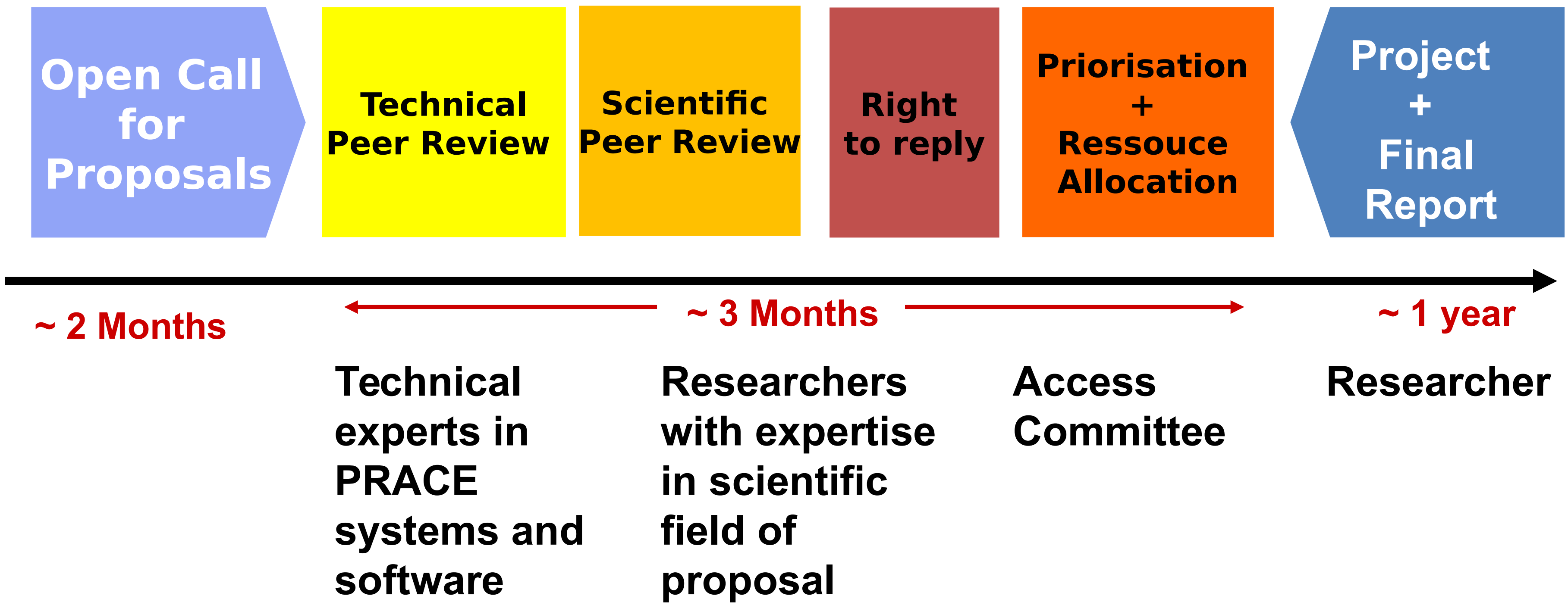
- The User Forum was set up in December 2011 through an initiative of PRACE itself.
- It is an independent entity where PRACE users can discuss their experiences and express their future needs as well as feedback on the current services and resources of the PRACE HPC Research Infrastructure. The aim is to provide an effective mechanism through which the Tier-0 user community can give feedback to PRACE.
- The PRACE User Forum takes the outcomes of these discussions to PRACE on behalf of the User Community and it has visibility in different social networks

PRACE peer-review access

Criterion: Scientific Excellence

- Free-of-charge, need to publish results at the end of the award period
- PRACE calls are open for international projects
- Types of resource allocations for scientists
  - **Project Access (every 6 months)**
    - For a specific project, award period ~ 1 to 3 years
    - For individual researchers and research groups (no restriction of nationality for both researcher and centre)
    - Requires to demonstrate technical feasibility of project
  - **Programmatic access**
    - purpose: to ensure a stable and reliable minimum access to the necessary computational resources for large-scale, long term projects of very high scientific quality and with a broad European scope, importance and relevance
    - maximum of 20% of the total resources available for programmatic access
  - **Preparatory Access**
    - Optionally with support from PRACE experts
    - Prepare proposals for Project Access

Project Access



Hadoop (1.2.1) useful commands

Create a directory in HDFS at given path(s).

$ hadoop fs -mkdir <paths>

List the contents of a directory.

$ hadoop fs -ls <args>

Upload and download a file in HDFS.

$ hadoop fs -put <localsrc> ... <HDFS_dest_Path>

Download.

$ hadoop fs -get <hdfs_src> <localdst>

See contents of a file

$ hadoop fs -cat <path[filename]>

Copy a file from source to destination

This command allows multiple sources as well in which case the destination must be a directory.

$ hadoop fs -cp <source> <dest>

Copy a file from/To Local file system to HDFS

copyFromLocal

$ hadoop fs -copyFromLocal <localsrc> URI

copyToLocal

$ hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>

Move file from source to destination.

Note:- Moving files across filesystem is not permitted.

$ hadoop fs -mv <src> <dest>

Remove a file or directory in HDFS.

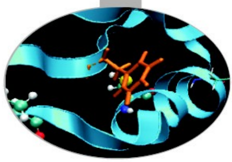
$ hadoop fs -rm(r) <arg>

Display last few lines of a file.

$ hadoop fs -tail <path[filename]>

Display the aggregate length of a file.

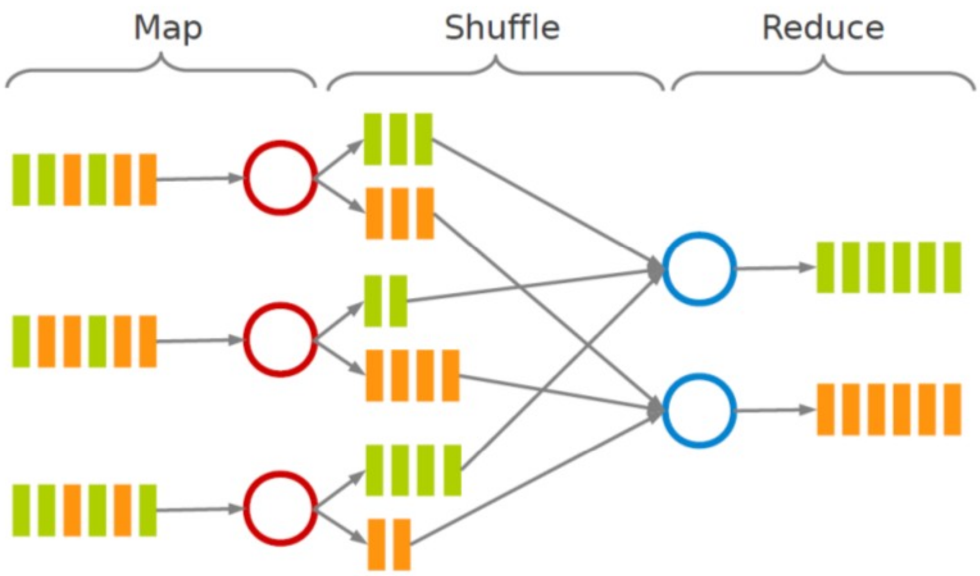
$ hadoop fs -du <path>



Word count

First example

- **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:
  - **Hello World Bye World**
  - **Hello Hadoop Goodbye Hadoop**

Word count

- The map function reads in words one a time and outputs (word, 1) for each parsed input word.
- The map function output is:

(Hello, 1)
(World, 1)
(Bye, 1)
(World, 1)
(Hello, 1)
(Hadoop, 1)
(Goodbye, 1)
(Hadoop, 1)

Word count

- The shuffle phase between map and reduce phase creates a list of values associated with each key.
- The reduce function input is:

(Bye, (1))
(Goodbye, (1))
(Hadoop, (1, 1)
(Hello, (1, 1))
(World, (1, 1))

Word count

- The reduce function sums the numbers in the list for each key and outputs (word, count) pairs.
- The output of the reduce function is the output of the MapReduce job:

(Bye, 1)
(Goodbye, 1)
(Hadoop, 2)
(Hello, 2)
(World, 2)

SuperComputing Applications and Innovation

MRJob code

```
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()
```

Testing the code

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

$ docker run -v ~/course-exercises:/course-exercises -i -t **cineca/hadoop-mrjob:1.2.1** /etc/bootstrap.sh -bash

root$ show-exercises

root$ python word_count.py ../data/txt/2261.txt.utf-8 (-r hadoop)

Word count (combiner)

```
from mrjob.job import MRJob
class MRWordCount2(MRJob):

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

    # Combiner step
    def combiner(self, word, occurrences):
        yield word, sum(occurrences)

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

if __name__ == '__main__':
    MRWordCount2.run()
```

How to execute jobs with MRJob

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`**

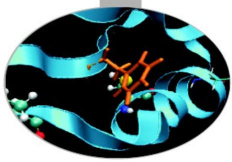
You can pass multiple input files, mixed with stdin (using the – character)

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

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



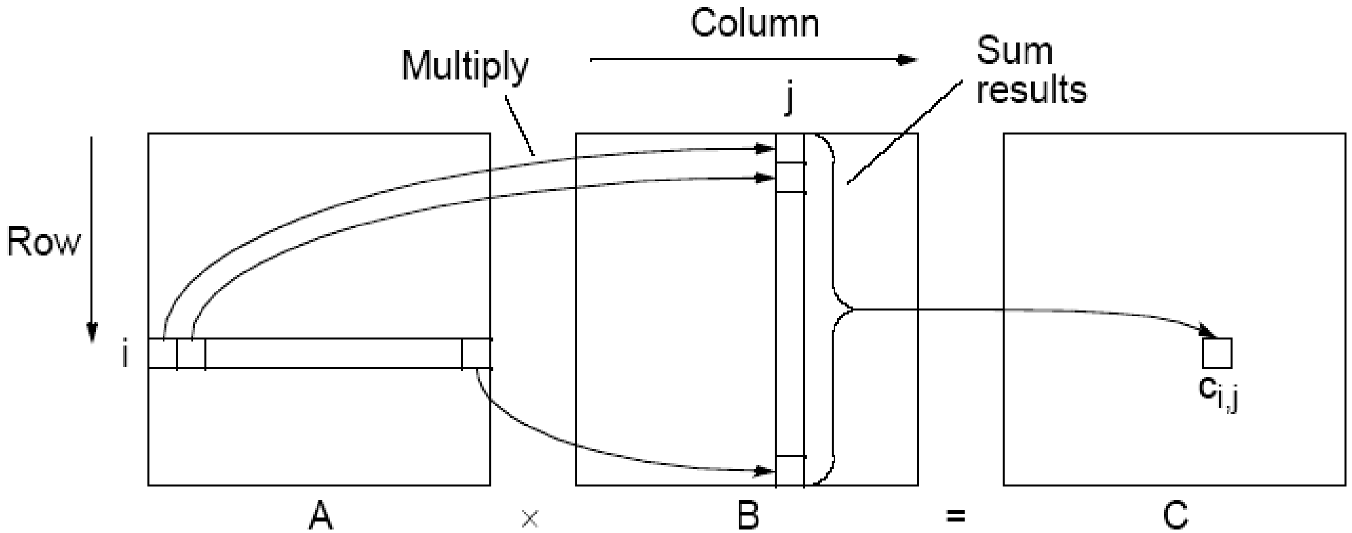
Matrix-matrix product v1

Part. 4

Matrix-matrix product

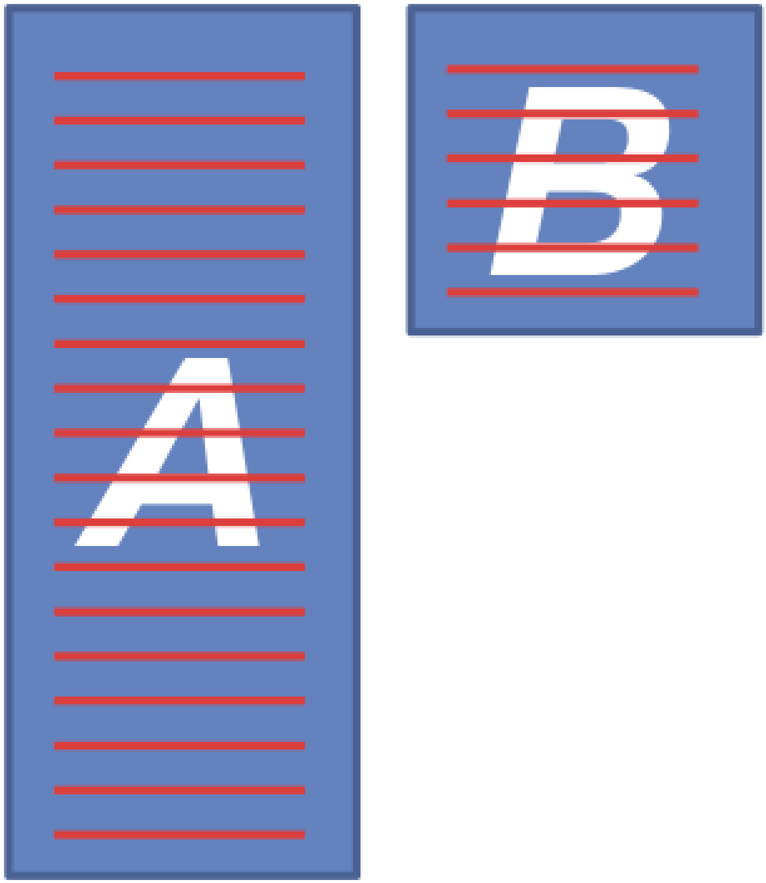
- Basic matrix multiplication on a 2-D grid
- Matrix multiplication is an important application in HPC and appears in many areas (linear algebra)
- **C = A * B** where A, B, and C are matrices (two-dimensional arrays)
- A restricted case is when B has only one column, matrix-vector product, which appears in representation of linear equations and partial differential equations

C = A x B



$$c_{i,j} = \sum_{k=0}^{l-1} a_{i,k} b_{k,j}$$

Matrix-matrix product



$$AB = C$$

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Matrix-matrix product

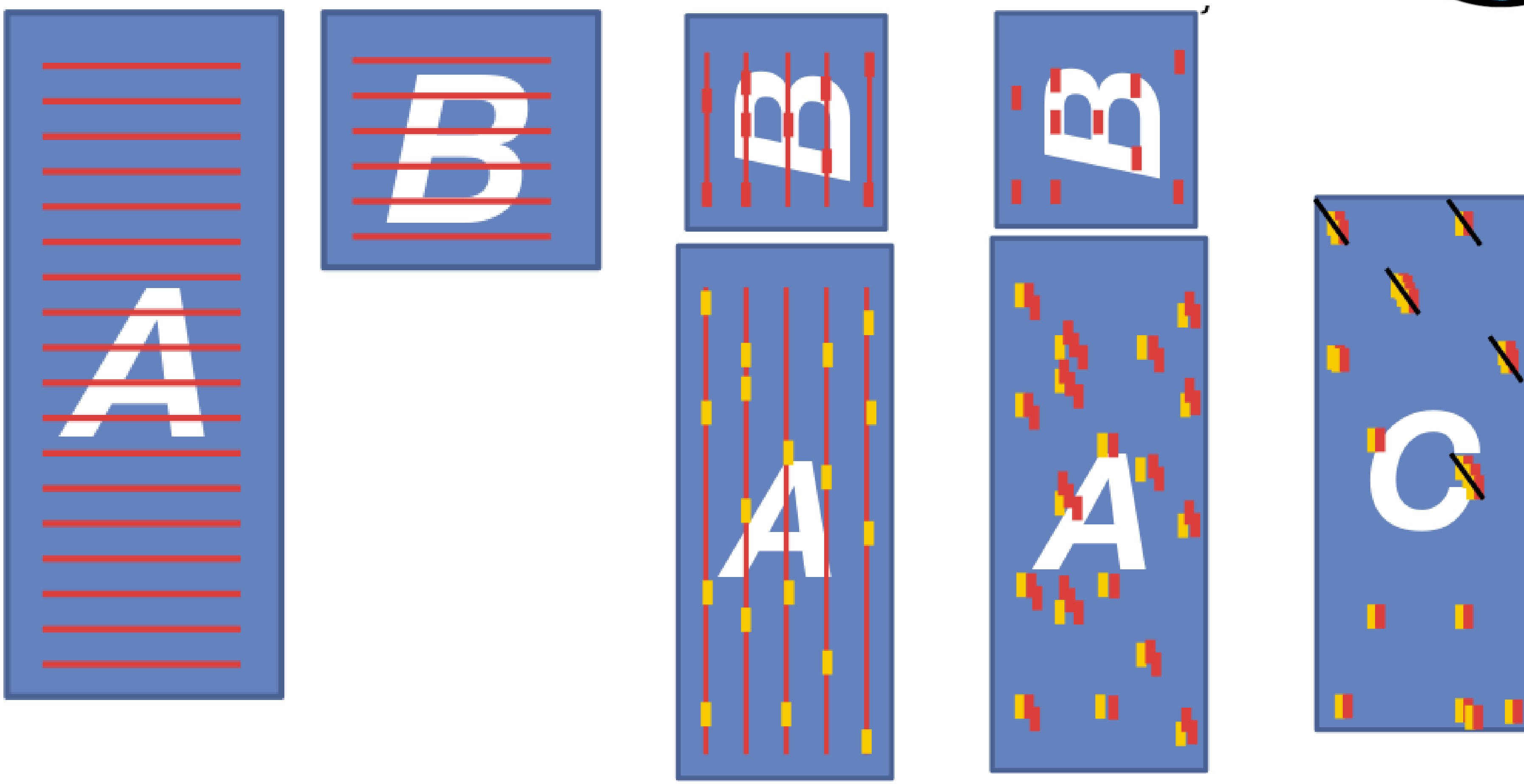
A is stored by row (`$ head data/mat/smat_10x5_A`)

```
0 0 0.599560659528 4 -1.53589644057
1
2 2 0.260564861569
3
4 0 0.26719729583 1 0.839470246524
5 2 -1.49761307371
6 0 0.558321894518 1 1.22774377511
7 2 -1.09283410126
8 1 -0.912374571316 3 1.40678001003
9 0 -0.402945890763
```

B is stored by row (`$ head data/mat/smat_5x5_B`)

```
0 0 0.12527732342 3 1.02407852061 4 0.121151207685
1 0 0.597062100484
2 2 1.24708888756
3 4 -1.45057798535
4 2 0.0618772663296
```

Matrix-matrix product



$$\boldsymbol{AB = C}$$

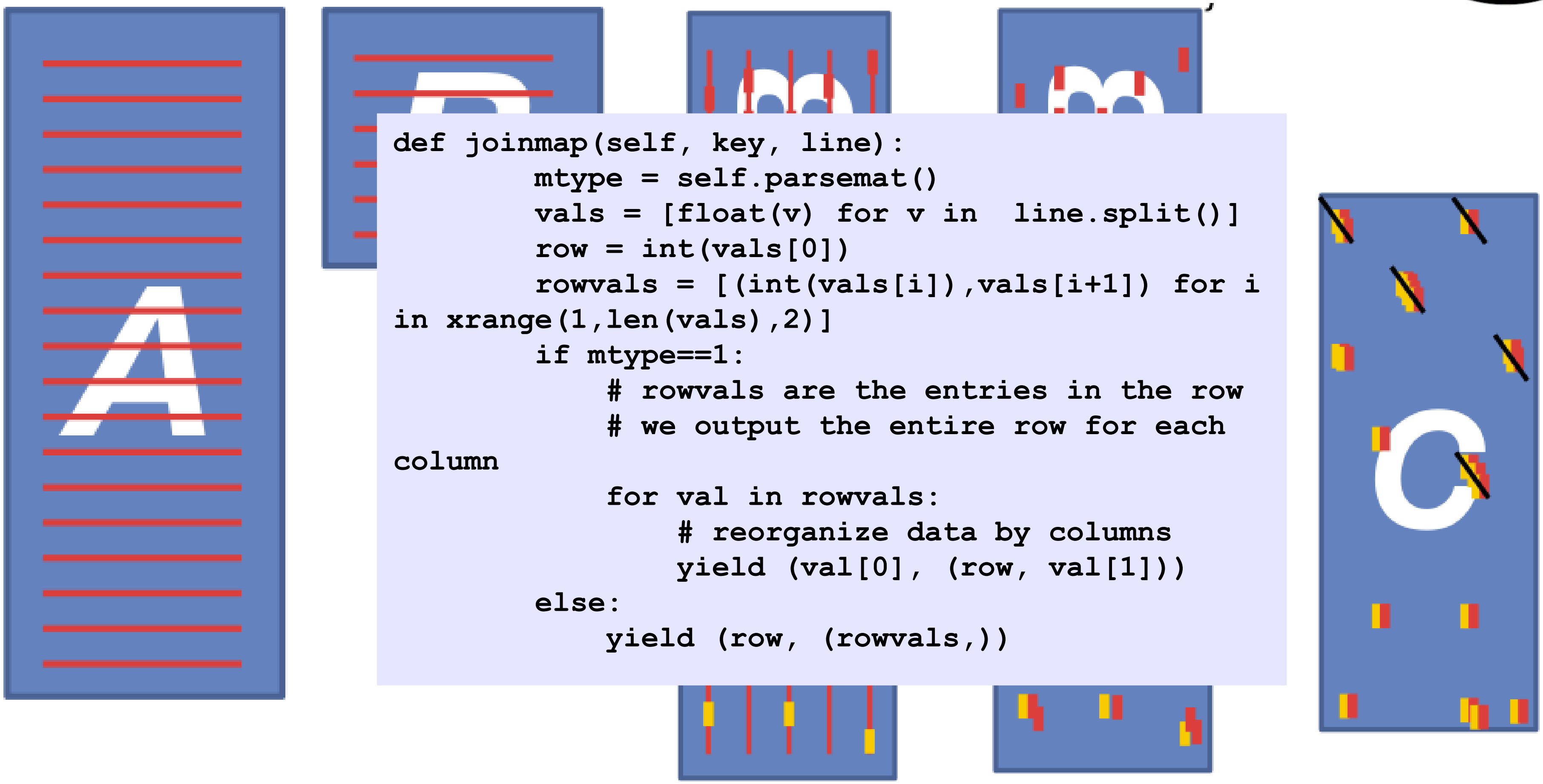
$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Map 1
Align on columns

Reduce 1
Output A_{ik}, B_{kj} keyed on (i,j)

Reduce 2
Output sum(A_{ik}, B_{kj})

Joinmap



```
def joinmap(self, key, line):
        mtype = self.parsemat()
        vals = [float(v) for v in  line.split()]
        row = int(vals[0])
        rowvals = [(int(vals[i]),vals[i+1]) for i
in xrange(1,len(vals),2)]
        if mtype==1:
            # rowvals are the entries in the row
            # we output the entire row for each
column
            for val in rowvals:
                # reorganize data by columns
                yield (val[0], (row, val[1]))
        else:
            yield (row, (rowvals,))
```

$$AB = C$$

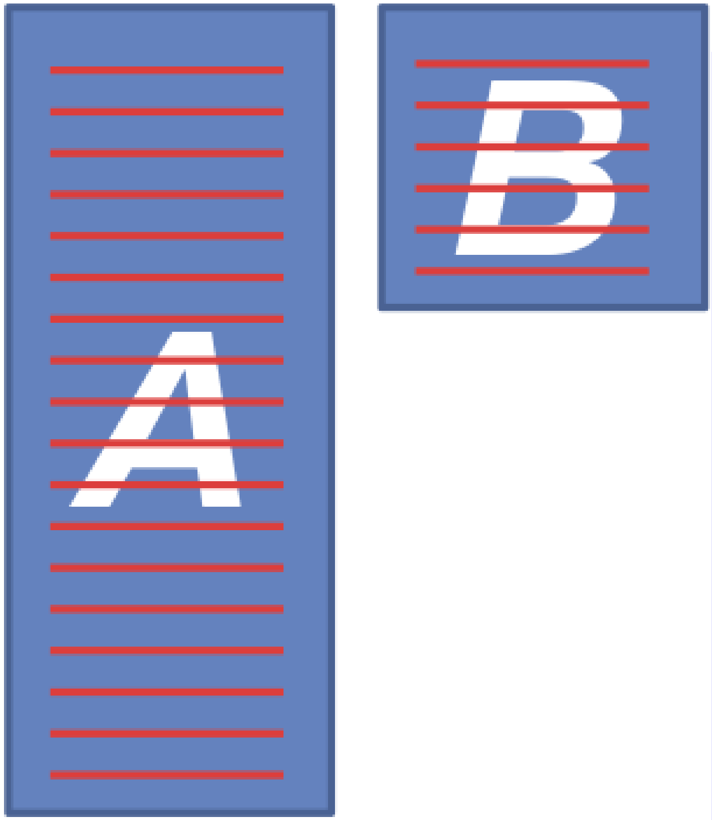
$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Map 1
Align on columns

Reduce 1
Output A_{ik}, B_{kj} keyed on (i,j)

Reduce 2
Output sum(A_{ik}, B_{kj})

Joinred




```
def joinred(self, key, vals):
        # each key is a column of the matrix.
        # and there are two types of values:
        #  len == 2 (1, row, A_row,key) # a column of A
        #  len == 1 rowvals # a row of B

        # load the data into memory
        brow = []
        acol = []
        for val in vals:
            if len(val) == 1:
                brow.extend(val[0])
            else:
                acol.append(val)

        for (bcol,bval) in brow:
            for (arow,aval) in acol:
                yield ((arow,bcol), aval*bval)
```

$$AB = C$$

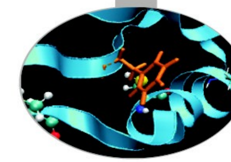
$$C_{ij} = \sum_k A_{ik} B_{kj}$$

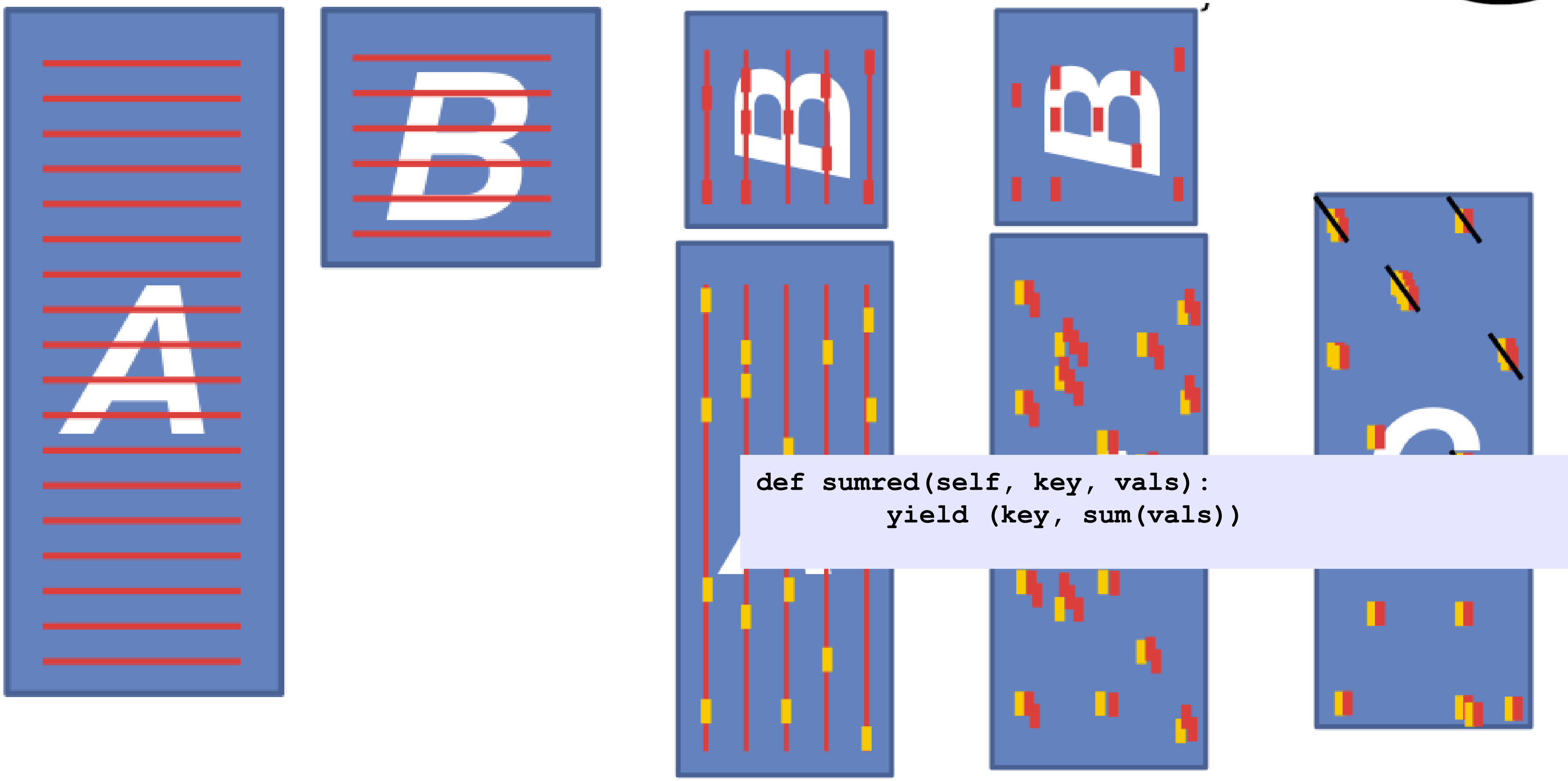
Map 1
Align on columns

Reduce 1
Output $A_{ik}B_{kj}$ keyed on (i,j)

Reduce 2
Output sum(A_{ik}, B_{kj})

Sumred






```
def sumred(self, key, vals):
        yield (key, sum(vals))
```

$$\boldsymbol{AB} = \boldsymbol{C}$$

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Map 1
Align on columns

Reduce 1
Output A_{ik}, B_{kj} keyed on (i,j)

Reduce 2
Output sum(A_{ik}, B_{kj})

mrjob/sparse_matmat.py

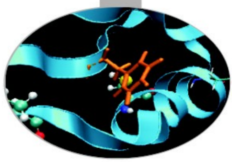
```
from mrjob.job import MRJob
from mrjob.compat import get_jobconf_value
import itertools
import sys

class SparseMatMult(MRJob):

    def configure_options(self):
        super(SparseMatMult,self).configure_options()
        self.add_passthrough_option('--A-
matrix',default='A',
            dest='Amatname')

    def parsemat(self):
        """ Return 1 if this is the A matrix, otherwise
return 2"""
        fn = get_jobconf_value('map.input.file')
        if self.options.Amatname in fn:
            return 1
        else:
            return 2

    def joinmap(self, key, line):
        mtype = self.parsemat()
        vals = [float(v) for v in  line.split()]
        row = int(vals[0])
        rowvals = [(int(vals[i]),vals[i+1]) for i in
xrange(1,len(vals),2)]
        if mtype==1:
            # rowvals are the entries in the row
            # we output the entire row for each column
            for val in rowvals:
                # reorganize data by columns
                yield (val[0], (row, val[1]))
        else:
            yield (row, (rowvals,))
```

```
def joinred(self, key, vals):
        brow = []
        acol = []
        for val in vals:
            if len(val) == 1:
                brow.extend(val[0])
            else:
                acol.append(val)
        for (bcol,bval) in brow:
            for (arow,aval) in acol:
                yield ((arow,bcol), aval*bval)


    def sumred(self, key, vals):
        yield (key, sum(vals))


    def rowgroupmap(self, key, val):
        yield key[0], (key[1], val)


    def appendred(self, key, vals):
        yield key, list(itertools.chain.from_iterable(vals))


    def steps(self):
        return [self.mr(mapper=self.joinmap,
reducer=self.joinred),
            self.mr(mapper=None, reducer=self.sumred),
            self.mr(mapper=self.rowgroupmap,
reducer=self.appendred)]


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

How to launch the code

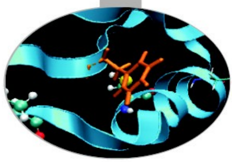
```
$ python mrjob/sparse_matmat.py (-r hadoop)
../data/mat/smat_100x10_A ../data/mat/smat_10x200_B
```

```
$ python utils/make_sparse_test_data_v1.py <nrows>
<ncols> <density>
```



Matrix-matrix product v2

Part. 4



Matrix-matrix product v2

We can think of a matrix as a relation with three attributes:

- the row number, the column number, and the value in that row and column.
- **M as a relation M (I, J, V), with tuples (i, j, m_{ij})**
- **N as a relation N (J, K, W), with tuples (j, k, n_{jk})**
- The product M N is almost the **natural join of M (I, J, V) and N (J, K, W)**, having only attribute J in common, would produce tuples (i, j, k, v, w) from each tuple (i, j, v) in M and tuple (j, k, w) in N
- This five-component tuple represents the pair of matrix elements **(m_{ij},n_{jk})**. What we want instead is the product of these elements, that is, the four-component tuple **(i, j, k, v × w), because that represents the product $m_{ij}n_{jk}$**
- Once we have this relation as the result of one Map Reduce operation, we can perform grouping and aggregation, with I and K as the grouping attributes and the sum of V × W as the aggregation.

Matrix-matrix product v2

The Map Function:

- For each matrix element m_{ij}, produce the key value pair j, (M, i, m_{ij}) . Likewise, for each matrix element n_{jk}, produce the key value pair j, (N, k, n_{jk}) . Note that M and N in the values are not the matrices themselves but rather a bit indicating whether the element comes from M or N

The Reduce Function:

- For each key j, examine its list of associated values. For each value that comes from M , say (M, i, m_{ij}) , and each value that comes from N , say (N, k, n_{jk}), produce a key-value pair with key equal to (i, k) and value equal to the product of these elements, $m_{ij}n_{jk}$

The Map Function:

- This function is just the identity. That is, for every input element with key (i, k) and value v, produce exactly this key-value pair

The Reduce Function:

- For each key (i, k), produce the sum of the list of values associated with this key. The result is a pair (i, k), v , where v is the value of the element in row i and column k of the matrix P = MN

mrjob/matmat.py

```
import sys
import random
import numpy
import pickle

from mrjob.job import MRJob
from mrjob.compat import get_jobconf_value
import os

class MatMult(MRJob):

    def configure_options(self):
        super(MatMult, self).configure_options()
        self.add_passthrough_option('--A-matrix', default='A',
            dest='Amatname')

    def parsemat(self):
        """ Return 1 if this is the A matrix, otherwise return 2"""
        fn = get_jobconf_value('map.input.file')
        if self.options.Amatname in fn:
            return 1
        else:
            return 2

    def emit_values(self, _, line):
        mtype = self.parsemat()
        a, b, v = line.split()

        v = float(v)

        if mtype == 1:
            i = int(a)
            j = int(b)
            yield j, (0, i, v)
        else:
            j = int(a)
            k = int(b)
            yield j, (1, k, v)

    def multiply_values(self, j, values):

        values_from1 = []

        values_from2 = []
        for v in values:
            if v[0] == 0:
                values_from1.append(v)
            elif v[0] == 1:
                values_from2.append(v)

        for (m, i, v1) in values_from1:
            for (m, k, v2) in values_from2:
                yield (i, k), v1*v2

    def identity(self, k, v):
        yield k, v

    def add_values(self, k, values):
        yield k, sum(values)

    def steps(self):
        return [self.mr(mapper=self.emit_values,
                        reducer=self.multiply_values),
                self.mr(mapper=self.identity,
                        reducer=self.add_values)]


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

Matrix-matrix product v2

Matrix is stored by value (**`$ head matmat_3x2_A`**)

0 0 1

0 1 2

1 0 2

1 1 3

2 0 4

2 1 5

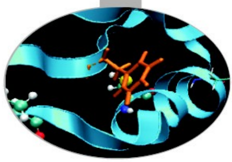
How to launch the code

```
$ python mrjob/matmat.py (-r hadoop)
../data/mat/matmat_3x2_A ../data/mat/matmat_2x2_B
```

```
$ python utils/make_sparse_test_data_v2.py <nrows>
<ncols> <density>
```

SCAI
SuperComputing Applications and Innovation

Log based debug



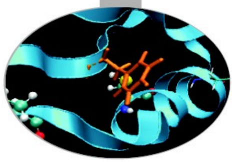
- Python

```
sys.stderr(out).write("REDUCER INPUT: ({0},{1})\n".format(j,
values))
```

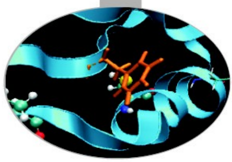
- Java

```
System.err.println("Temperature over 100 degrees for input: " +
value);
```

PRACE CINECA



MapReduce Weaknesses and Solving Techniques



When to use MR + Hadoop

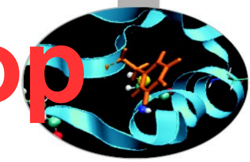
When to use MR + Hadoop

- **Your Data Sets Are Really Big**
  - *Don't even think about Hadoop if the data you want to process is measured in MBs or GBs.* If the data driving the main problem you are hoping to use Hadoop to solve is measured in GBs, save yourself the hassle and use Excel, a SQL BI tool on Postgres, or some similar combination. On the other hand, if it's several TB or (even better) measured in petabytes, Hadoop's superior scalability will save you a considerable amount of time and money
- **You Celebrate Data Diversity**
  - One of the advantages of the Hadoop Distributed File System (HDFS) is it's really flexible in terms of data types. It doesn't matter whether your raw data is structured, semi-structured (like XML and log files), unstructured (like video files).

When to use MR + Hadoop

- **You Find Yourself Throwing Away Perfectly Good Data**
  - One of the great things about Hadoop is its capability to store petabytes of data. If you find that you are throwing away potentially valuable data because its costs too much to archive, you may find that setting up a Hadoop cluster allows you to retain this data, and gives you the time to figure out how to best make use of that data.

When to NOT use MR + Hadoop



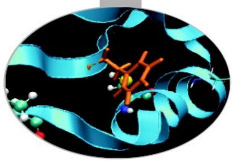
- **You Need Answers in a Hurry**
  - Hadoop is probably not the ideal solution if you need really fast access to data. The various SQL engines for Hadoop have made big strides in the past year, and will likely continue to improve. But if you're using Map-Reduce to crunch your data, expect to wait days or even weeks to get results back.
- **Your Queries Are Complex and Require Extensive Optimization**
  - Hadoop is great because it gives you a massively parallel cluster for low-cost Lintel servers and scads of cheap hard disk capacity. While the hardware and scalability is straightforward, getting the most out of Hadoop typically requires a hefty investment in the technical skills required to optimize queries.

When to NOT use MR + Hadoop

- **You Require Random, Interactive Access to Data**
  - The pushback from the limitations of the batch-oriented MapReduce paradigm in early Hadoop led the community to improve SQL performance and boost its capability to serve interactive queries against random data. While SQL on Hadoop is getting better, in most cases it’s not a reason in of itself to adopt Hadoop.
- **You Want to Store Sensitive Data**
  - Hadoop is evolving quickly and is able to do a lot of things that it couldn’t do just a few years ago. But one of the things that it’s not particularly good at today is storing sensitive data. Hadoop today has basic data and use access security. And while these features are improving by the month, the risks of accidentally losing personally identifiable information due to Hadoop’s less-than-stellar security capabilities is probably not worth the risk.

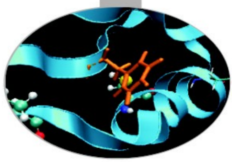
Advantages/Disadvantages

- **Now it's easy to program for many CPUs**
  - Communication management effectively gone
    - I/O scheduling done for us
  - Fault tolerance, monitoring
    - machine failures, suddenly-slow machines, etc are handled
  - Can be much easier to design and program!
  - Can cascade several (many?) Map-Reduce tasks
- **But … it further restricts solvable problems**
  - Might be hard to express problem in Map-Reduce
  - Data parallelism is key
  - Need to be able to break up a problem by data chunks
  - Map-Reduce is closed-source (to Google) C++
  - Hadoop is open-source Java-based rewrite



What if

- If you have access to a Hadoop cluster and you want a one-off quick-and-dirty job…
  – *Hadoop Streaming*
- If you don't have access to Hadoop and want to try stuff out…
  – *MrJob*
- If you're heavily using AWS…
  – *MrJob*
- If you want to work interactively…
  – *PySpark*
- If you want to do in-memory analytics…
  – *PySpark*
- If you want to do anything…*
  – *PySpark*
- If you want ease of Python with high performance
  – *Impala + Numba*



Debugging

Debug mechanisms

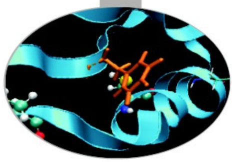
- The Web Interface
- Runtime monitor
- Log based debug

The Web User Interface

- **Hadoop** comes with a web UI for viewing information about your jobs. It is useful for following a job's progress while it is running, as well as finding job statistics and logs after the job has completed.

- You can find the UI at *http://127.0.0.1:50030/*

- **$** `docker run -p 127.0.0.1:50030:50030 -p 127.0.0.1:50070:50070 -i -t  cineca/hadoop-mrjob:1.2.1 /etc/bootstrap.sh -bash`



db75867f9e2c Hadoop Map/Reduce Administration

State: RUNNING
Started: Sun Nov 30 11:13:55 UTC 2014
Version: 1.2.1, r1503152
Compiled: Mon Jul 22 15:23:09 PDT 2013 by mattf
Identifier: 201411301113
SafeMode: OFF

Cluster Summary (Heap Size is 72 MB/889 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
0	0	1	1	0	0	0	0	2	2	4.00	0	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)

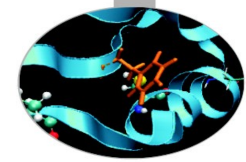
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

---

Running Jobs

none

Hadoop job_201411301113_0001 on db75867f9e2c



User: root
Job Name: streamjob5169397935625686158.jar
Job File: hdfs://db75867f9e2c:9000/tmp/hadoop-root/mapred/staging/root/.staging/job_201411301113_0001/job.xml
Submit Host: db75867f9e2c
Submit Host Address: 172.17.0.6
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started at: Sun Nov 30 11:19:03 UTC 2014
Finished at: Sun Nov 30 11:19:23 UTC 2014
Finished in: 20sec
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

	Counter	Map	Reduce	Total
File Input Format Counters	Bytes Read	0	0	554,451
Job Counters	SLOTS_MILLIS_MAPS	0	0	12,887
	Launched reduce tasks	0	0	1
	Total time spent by all reduces waiting after reserving slots (ms)	0	0	0
	Total time spent by all maps waiting after reserving slots (ms)	0	0	0
	Launched map tasks	0	0	2
	Data-local map tasks	0	0	2
	SLOTS_MILLIS_REDUCES	0	0	10,031
File Output Format Counters	Bytes Written	0	0	43
FileSystemCounters	FILE_BYTES_READ	0	545,621	545,621
	HDFS_BYTES_READ	554,781	0	554,781
	FILE_BYTES_WRITTEN	667,327	606,362	1,273,689
	HDFS_BYTES_WRITTEN	0	43	43

Hadoop Reporter

- The fastest way of debugging programs is via print statements, and this is certainly possible in Hadoop.
- However, there are complications to consider: *with programs running on tens, hundreds, or thousands of nodes,* ***how do we find and examine the output of the debug statements, which may be scattered across these nodes?***
- For a particular case, where we are looking for (what we think is) an unusual case, **we can use a debug statement to log to standard error, in conjunction with a message to update the task's status message to prompt us to look in the error log.** The web UI makes this easy, as you will see.

Hadoop Reporter

"A facility for Map-Reduce applications to report progress and update counters, status information etc."

```
if (temperature > 1000) {

   System.err.println("Temperature over 100 degrees
   for input: " + value);
   reporter.setStatus("Detected possibly corrupt
   record: see logs.");
   reporter.incrCounter(Temperature.OVER_100, 1);

}
```

Hadoop Reporter

Hadoop map task list for job_200904110811_0003 on ip-10-250-110-47

Completed Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_200904110811_0003_m_000043	100.00%	hdfs://ip-10-250-110-47.ec2.internal/user/root/input/ncdc/all/1949.gz:0+220338475	11-Apr-2009 09:00:06	11-Apr-2009 09:01:25 (1mins, 18sec)		10
task_200904110811_0003_m_000044	100.00%	Detected possibly corrupt record: see logs.	11-Apr-2009 09:00:06	11-Apr-2009 09:01:28 (1mins, 21sec)		11
task_200904110811_0003_m_000045	100.00%	hdfs://ip-10-250-110-47.ec2.internal/user/root/input/ncdc/all/1970.gz:0+208374610	11-Apr-2009 09:00:06	11-Apr-2009 09:01:28 (1mins, 21sec)		10

Runtime monitor

- The Java Platform Debugger Architecture is a collection of APIs to debug Java code.

- Java Debugger Interface (JDI) - defines a high-level Java language interface that developers can easily use to write remote debugger application tools.

```
$ export HADOOP_OPTS="-agentlib:jdwp=transport=dt_socket,server=y,suspend=y, address=8000"
```

http://docs.oracle.com/javase/6/docs/technotes/guides/jpda/

Log based debug

- **Python**

```
sys.stderr(out).write("REDUCER INPUT: ({0},{1})\n".format(j,
values))
```

- **Java**

```
System.err.println("Temperature over 100 degrees for input: " +
value);
```

Debugging/profiling

Job Configuration: JobId - job_201411301113_0001

name	
job.end.retry.interval	30000
io.bytes.per.checksum	512
mapred.job.tracker.retiredjobs.cache.size	1000
mapreduce.jobhistory.cleaner.interval-ms	86400000
mapred.queue.default.acl-administer-jobs	*
dfs.image.transfer.bandwidthPerSec	0
mapred.task.profile.reduces	0-2
mapreduce.jobtracker.staging.root.dir	${hadoop.tmp.dir}/mapred/staging
mapreduce.job.cache.files.visibilities	true,true
mapred.job.reuse.jvm.num.tasks	1
dfs.block.access.token.lifetime	600
mapred.reduce.tasks.speculative.execution	true
mapred.job.name	streamjob5169397935625686158.jar
hadoop.http.authentication.kerberos.keytab	${user.home}/hadoop.keytab
dfs.permissions.supergroup	supergroup
io.seqfile.sorter.recordlimit	1000000
stream.reduce.output.reader.class	org.apache.hadoop.streaming.io.TextOutputReader
hadoop.relaxed.worker.version.check	false
mapred.task.tracker.http.address	0.0.0.0:50060
stream.reduce.input.writer.class	org.apache.hadoop.streaming.io.TextInputWriter
dfs.namenode.delegation.token.renew-interval	86400000
mapred.cache.archives.timestamps	1417346338423
fs.ramfs.impl	org.apache.hadoop.fs.InMemoryFileSystem
mapred.system.dir	${hadoop.tmp.dir}/mapred/system
dfs.namenode.edits.toleration.length	0
mapred.task.tracker.report.address	127.0.0.1:0
mapreduce.reduce.shuffle.connect.timeout	180000
mapreduce.job.counters.max	120
dfs.datanode.readahead.bytes	4193404
mapred.healthChecker.interval	60000
mapreduce.job.complete.cancel.delegation.tokens	true
dfs.namenode.replication.work.multiplier.per.iteration	2
fs.trash.interval	0
hadoop.jetty.logs.serve.aliases	true

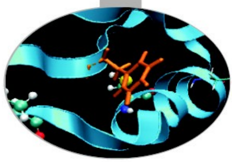
Profiling

- Like debugging, profiling a job running on a distributed system like MapReduce presents some challenges. Hadoop allows you to profile a fraction of the tasks in a job, and, as each task completes, pulls down the profile information to your machine for later analysis with standard profiling tools.
- **HPROF** is a profiling tool that comes with the JDK that, although basic, can give valuable information about a program's CPU and heap usage.

```
conf.setProfileEnabled(true);
conf.setProfileParams("-
agentlib:hprof=cpu=samples,heap=sites,depth=6," +
"force=n,thread=y,verbose=n,file=%s");
conf.setProfileTaskRange(true, "0-2");
```

https://docs.oracle.com/javase/7/docs/technotes/samples/hprof.html

SuperComputing Applications and Innovation



Profiling

- Set **mapred.task.profile** to true
- Profile a small range of maps/reduces
    - **mapred.task.profile.{maps|reduces}**
- **hprof** support is built-in
- Use mapred.task.profile.params to set options for the debugger
- Possibly *DistributedCache* for the profiler's agent

NameNode 'db75867f9e2c:9000'

Started: Sun Nov 30 11:13:51 UTC 2014
Version: 1.2.1, r1503152
Compiled: Mon Jul 22 15:23:09 PDT 2013 by mattf
Upgrades: There are no upgrades in progress.

Browse the filesystem
Namenode Logs
Go back to DFS home

Live Datanodes : 1

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Bloc
db75867f9e2c	2	In Service	18.21	0	13.43	4.78	0		26.25	

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_201411301113_0001_m_000000	100.00%	Records R/W=3586/1	30-Nov-2014 11:19:06	30-Nov-2014 11:19:11 (4sec)		16
task_201411301113_0001_m_000001	100.00%	Records R/W=3609/1	30-Nov-2014 11:19:06	30-Nov-2014 11:19:11 (4sec)		16

NameNode 'db75867f9e2c:9000'

Started:	Sun Nov 30 11:13:51 UTC 2014
Version:	1.2.1, r1503152
Compiled:	Mon Jul 22 15:23:09 PDT 2013 by mattf
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

13 files and directories, 14 blocks = 27 total. Heap Size is 72 MB / 889 MB (8%)

Configured Capacity	:	18.21 GB
DFS Used	:	28.01 KB
Non DFS Used	:	13.43 GB
DFS Remaining	:	4.78 GB
DFS Used%	:	0 %
DFS Remaining%	:	26.25 %
Live Nodes	:	1
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0

NameNode Storage:

Storage Directory	Type	State
/tmp/hadoop-root/dfs/name	IMAGE_AND_EDITS	Active

Cluster optimizations

The problem:

- Out of the box configuration not friendly
- Difficult to debug
- Performance – tuning/optimizations is a black art

Hadoop basic options

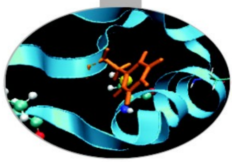
All hadoop commands are invoked by the bin/hadoop script. Running the hadoop script without any arguments prints the description for all commands.

```
Usage: hadoop [--config confdir] [COMMAND]
   [GENERIC_OPTIONS] [COMMAND_OPTIONS]
```

Hadoop has an option parsing framework that employs parsing generic options as well as running classes.

SuperComputing Applications and Innovation

Hadoop basic options



-conf <configuration file> Specify an application configuration file.

-D <property=value> Use value for given property.

-fs <local|namenode:port> Specify a namenode.

-jt <local|jobtracker:port> Specify a job tracker. Applies only to job.

-files <comma separated list of files> Specify comma separated files to be copied to the map reduce cluster. Applies only to job.

-libjars <comma seperated list of jars> Specify comma separated jar files to include in the classpath. Applies only to job.

-archives <comma separated list of archives> Specify comma separated archives to be unarchived on the compute machines. Applies only to job.

Configuration parameters

Compression *mapred.compress.map.output* → Map Output Compression

- **Default**: False
- **Pros**: Faster disk writes, lower disk space usage, lesser time spent on data transfer (from mappers to reducers).
- **Cons**: Overhead in compression at Mappers and decompression at Reducers.
- **Suggestions**: For large cluster and large jobs this property should be set true.

```
$ hadoop -Dmapred.compress.map.output=<false|true>
```

Speculative Execution

Speculative Execution *mapred.map/reduce.speculative.execution*
→ Enable/Disable task (map/reduce) speculative Execution

- **Default**: True
- **Pros**: Reduces the job time if the task progress is slow due to memory unavailability or hardware degradation.
- **Cons**: Increases the job time if the task progress is slow due to complex and large calculations. On a busy cluster speculative execution can reduce overall throughput, since redundant tasks are being executed in an attempt to bring down the execution time for a single job.
- **Suggestions**: In large jobs where average task completion time is significant (> 1 hr) due to complex and large calculations and high throughput is required the speculative execution should be set to false.

```
$ bin/hadoop jar -Dmapred.map.tasks.speculative.execution=false \
                 -Dmapred.reduce.tasks.speculative.execution=false
```

Speculative execution

- It is possible for one Map task to run more slowly than the others (perhaps due to faulty hardware, or just a very slow machine)

- It would appear that this would create a bottleneck
    - The reduce method in the Reducer cannot start until every Mapper has finished

- Hadoop uses speculative execution to mitigate against this
    - If a Mapper appears to be running significantly more slowly than the others, a new instance of the Mapper will be started on another machine, operating on the same data
        - The results of the first Mapper to finish will be used
        - Hadoop will kill off the Mapper which is still running

Number of Maps/Reducers

Number of Maps/Reducers

mapred.tasktracker.map/reduce.tasks.maximum → Maximum tasks (map/reduce) for a tasktracker

- **Default**: 2
- **Suggestions**: Recommended range - (cores_per_node)/2 to 2x(cores_per_node), especially for large clusters. This value should be set according to the hardware specification of cluster nodes and resource requirements of tasks (map/reduce).

File block size

File block size *dfs.block.size →* File system block size

- **Default**: 67108864 (bytes)
- **Suggestions**:
  - Small cluster and large data set: default block size will create a large number of map tasks. e.g. Input data size = 160 GB and dfs.block.size = 64 MB then the minimum no. of maps= (160*1024)/64 = 2560 maps.
  - If dfs.block.size = 128 MB minimum no. of maps= (160*1024)/128 = 1280 maps.
  - If dfs.block.size = 256 MB minimum no. of maps= (160*1024)/256 = 640 maps.
  - In a small cluster (6-10 nodes) the map task creation overhead is considerable. So dfs.block.size should be large in this case but small enough to utilize all the cluster resources. The block size should be set according to size of the cluster, map task complexity, map task capacity of cluster and average size of input files.

Sort size

Sort size *<u>io.sort.mb → </u>*Buffer size (MBs) for sorting

- **Default**: 100
- **Suggestions**: For Large jobs (the jobs in which map output is very large), this value should be increased keeping in mind that it will increase the memory required by each map task. So the increment in this value should be according to the available memory at the node. Greater the value of io.sort.mb, lesser will be the spills to the disk, saving write to the disk

Sort factor

Sort factor *io.sort.factor* → Stream merge factor

- **Default**: 10
- **Suggestions**: For Large jobs (the jobs in which map output is very large and number of maps are also large) which have large number of spills to disk, value of this property should be increased. The number of input streams (files) to be merged at once in the map/reduce tasks, as specified by io.sort.factor, should be set to a sufficiently large value (for example, 100) to minimize disk accesses. Increment in io.sort.factor, benefits in merging at reducers since the last batch of streams (equal to io.sort.factor) are sent to the reduce function without merging, thus saving time in merging.

JVM Reuse

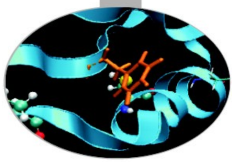
JVM reuse *mapred.job.reuse.jvm.num.tasks* → Reuse single JVM

- **Default**: 1
- **Suggestions**: The minimum overhead of JVM creation for each task is around 1 second. So for the tasks which live for seconds or a few minutes and have lengthy initialization, this value can be increased to gain performance.

Reduce parallel copies

Reduce parallel copies *mapred.reduce.parallel.copies* → Threads for parallel copy at reducer

- **Default**: 5
- **Description**: The number of threads used to copy map outputs to the reducer.
- **Suggestions**: For Large jobs (the jobs in which map output is very large), value of this property can be increased keeping in mind that it will increase the total CPU usage.

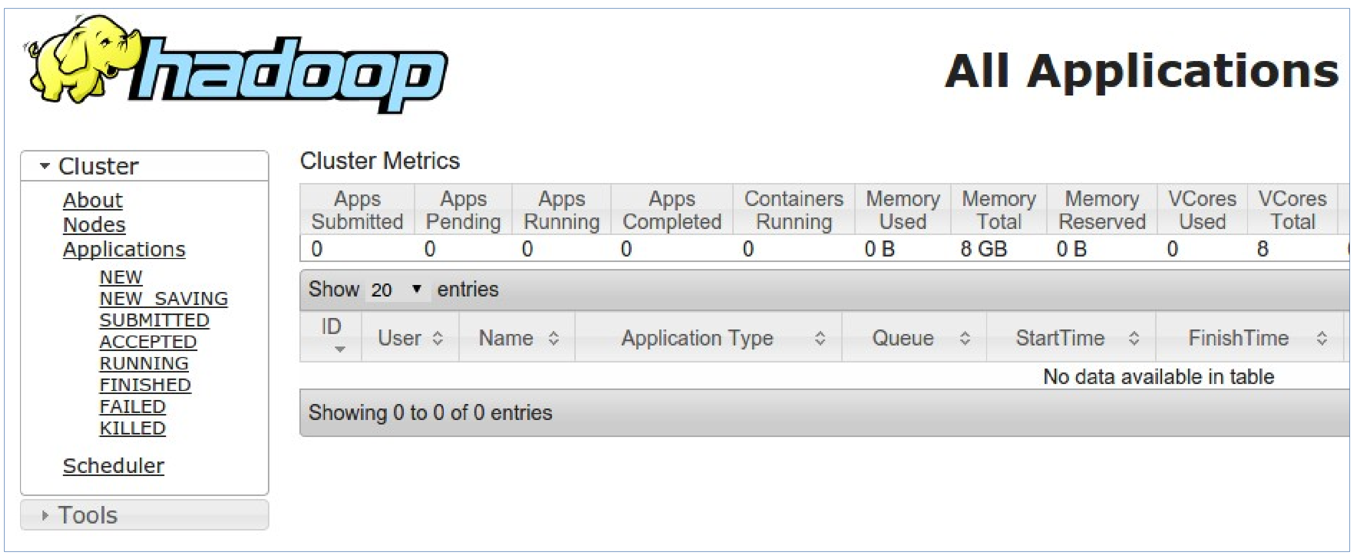


Map Reduce Limitations

Exercise

SPARK Environment

```
$ docker run -v … -p 127.0.0.1:8088:8088 -p
127.0.0.1:8042:8042 -i -t cineca/hadoop-spark:1.1.0
/etc/bootstrap.sh -bash
```




```
http://127.0.0.1:8088
```

SCAI
SuperComputing Applications and Innovation

SPARK shell

```
root@555b23de51e8: /course-exercises
odify permissions: Set(root, )
14/12/12 08:53:37 INFO spark.HttpServer: Starting HTTP Server
14/12/12 08:53:38 INFO server.Server: jetty-8.y.z-SNAPSHOT
14/12/12 08:53:38 INFO server.AbstractConnector: Started SocketConnector@0.0.0.0
:55131
14/12/12 08:53:38 INFO util.Utils: Successfully started service 'HTTP class serv
er' on port 55131.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.1.0
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_72)
Type in expressions to have them evaluated.
Type :help for more information.
14/12/12 08:53:47 INFO spark.SecurityManager: Changing view acls to: root,
14/12/12 08:53:47 INFO spark.SecurityManager: Changing modify acls to: root,
14/12/12 08:53:47 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disab
led; users with view permissions: Set(root, ); users with modify permissions: Set(root, )
14/12/12 08:53:53 INFO slf4j.Slf4jLogger: Slf4jLogger started
14/12/12 08:53:53 INFO Remoting: Starting remoting
14/12/12 08:53:53 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriver@55
5b23de51e8:38937]
14/12/12 08:53:53 INFO Remoting: Remoting now listens on addresses: [akka.tcp://sparkDriver@555b23de5
1e8:38937]
```

```
$ spark-shell
```

PRACE

CINECA

SPARK Shell (using Scala)

```
$ hadoop fs -put ../data/txt/divine_comedy.txt
/spark/divine_comedy.txt

$ spark-shell
$ scala> val textFile = sc.textFile("/spark/divine_comedy.txt")
// create a Resilient Distributed Dataset
$ scala> textFile.count() // Number of items in this RDD
$ scala> textFile.first() // First item in this RDD
$ scala> val linesWithCanto = textFile.filter(line =>
line.contains("Canto"))

$ scala> textFile.filter(line =>
line.contains("Canto")).count()
$ scala> linesWithSpark.cache()
$ scala> linesWithSpark.count()
```

SuperComputing Applications and Innovation

SPARK Exercise

```
import re
import sys

from pyspark import SparkContext

#function to extract the data from the line
#based on position and filter out the invalid records
def extractData(line):
    val = line.strip()
    (year, temp, q) = (val[15:19], val[87:92], val[92:93])
    if (temp != "+9999" and re.match("[01459]", q)):
        return [(year, temp)]
    else:
        return []

#Create Spark Context with the master details and the application name

sc = SparkContext(appName="PythonMaxTemp")

#Create an RDD from the input data in HDFS
weatherData = sc.textFile(sys.argv[1], 1)

#Transform the data to extract/filter and then find the max temperature
max_temperature_per_year = weatherData.flatMap(extractData).reduceByKey(lambda a,b : a if int(a) > int(b) else
b)

#Save the RDD back into HDFS
max_temperature_per_year.saveAsTextFile("output")
```

course-exercises/spark/max_temp.py

spark/max_temp.py

```
$ hadoop fs -put ../data/spark/1902 /spark/1902

$ spark-submit --master yarn-client max_temp.py
/spark/1902

$ hadoop fs -get /user/root/output/part-00000
```

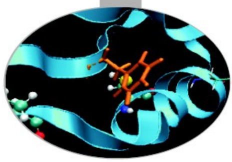
SPARK execution

YARN-client mode

In yarn-client mode, the driver runs in the client process, and the application master is only used for requesting resources from YARN.

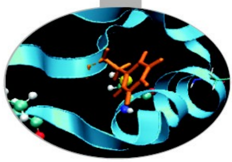
YARN-cluster mode

In yarn-cluster mode, the Spark driver runs inside an application master process which is managed by YARN on the cluster, and the client can go away after initiating the application. This mode is not available for Python.



SPARK vs Map Reduce

Criteria	Map Reduce	Spark
Conciseness	Plain MR has a lot of boiler plate	Almost no boilerplate
Performance	High latency	very fast compared to MR
Testability	Possible via libraries, but non trivial	Very much easy
Iterative processing	Non trivial	straight forward
Exploration of data	Not possible easily	Spark shell allows quick and easy data exploration
SQL like interface	Via Hive	Build in as SparkSQL
Fault Tolerance	Inheranlty able to handle fault tolerance via persisting the results of each of phases	Exploits immutability of RDD to enable fault tolerance
Eco system	lots of tools available but integration is not quite seamless, requiring lot of effort for their seamless integration	Unifies lot of interfaces like SQL, stream processing etc into single abstraction of RDD
In memory computations	not possible	possible



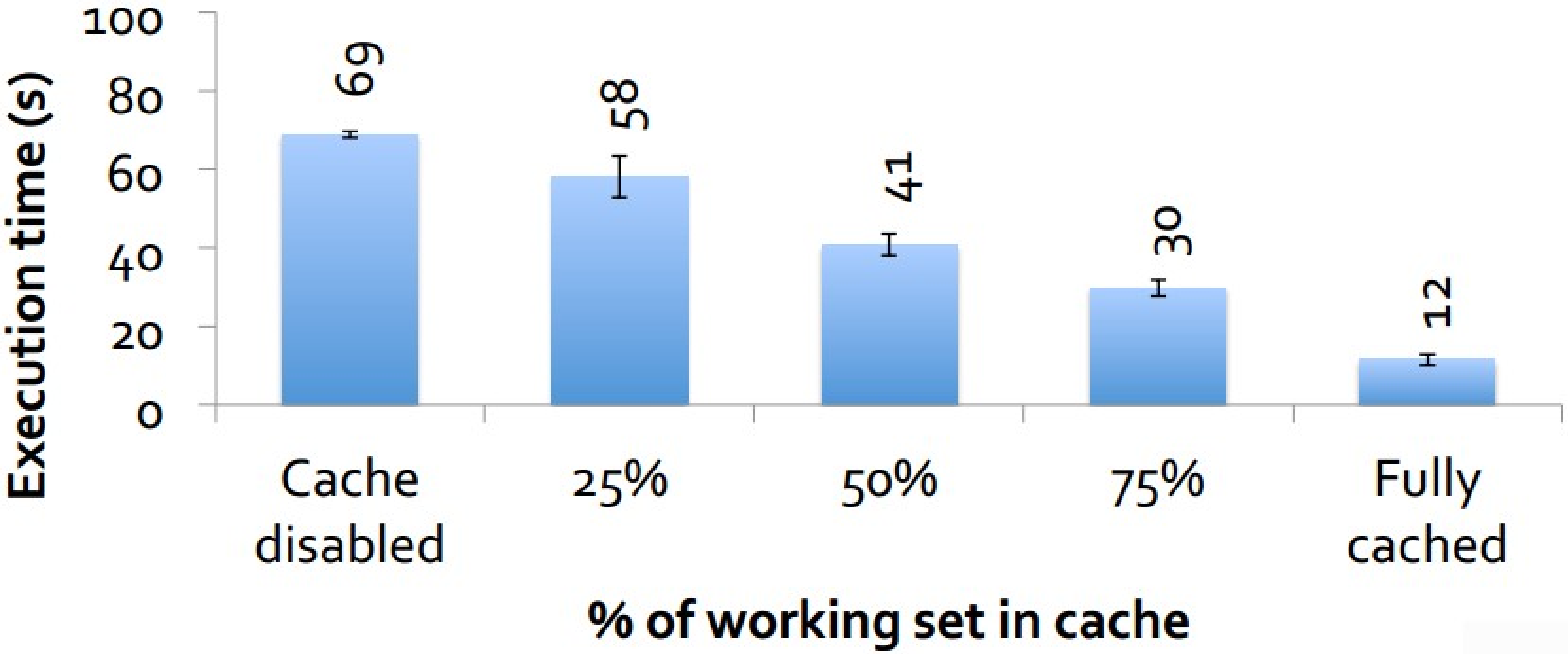
SPARK Performance

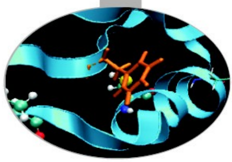
	Hadoop World Record	Spark 100 TB *	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark Daytona Rules	Yes	Yes	No
Environment	dedicated data center	EC2 (i2.8xlarge)	EC2 (i2.8xlarge)

* not an official sort benchmark record

http://databricks.com/blog/2014/10/10/spark-petabyte-sort.html

SPARK caching performance





What do we do when there is too much data to process?

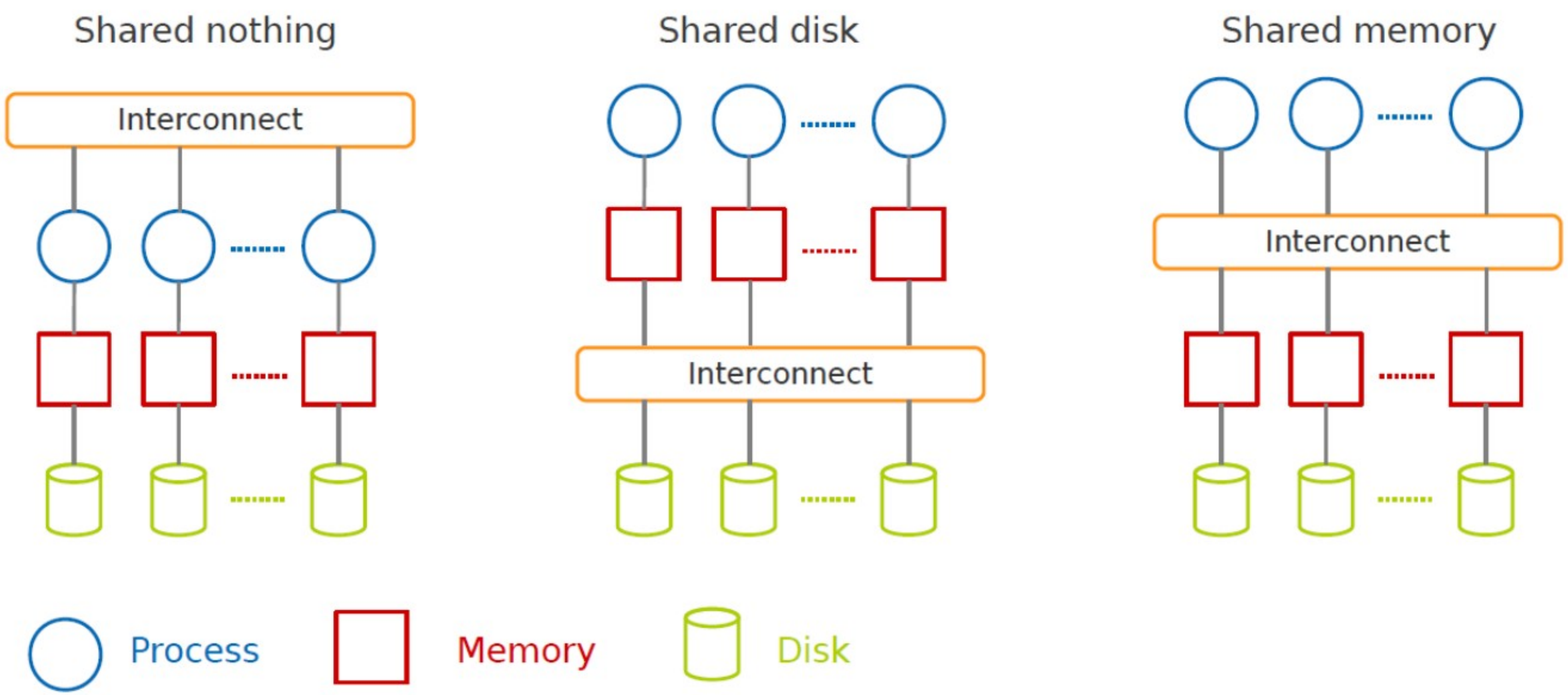
Scale Up vs. Scale Out (1/2)

- Scale up or scale vertically:
  - adding resources to a single node in a system.
- Scale out or scale horizontally:
  - adding more nodes to a system.

Scale Up vs. Scale Out (2/2)

- Scale up:
  - more expensive than scaling out.
- Scale out:
  - more challenging for fault tolerance and software development.

Taxonomy of Parallel Architectures

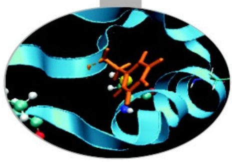


DeWitt, D. and Gray, J. “Parallel database systems: the future of high performance database systems”. ACM Communications, 35(6), 85-98, 1992.

Different classes of applications

- **Map Reduce/Hadoop**
  - A shared nothing architecture for processing large data sets with a distributed algorithm on clusters.
- **MPI (Message Passing Interface)**
  - A shared disk infrastructure for processing large data sets with a parallel algorithm on clusters
- **OpenMP (Open MultiProcessing)**
  - A shared memory infrastructure for processing large data sets with a parallel algorithm on a node

Programming Models: What is MPI?



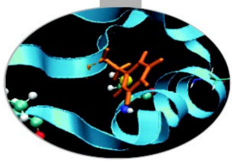
- **Message Passing Interface (MPI)**
  - World's most popular distributed API
  - MPI is "de facto standard" in scientific computing
  - C and FORTRAN, ver. 2 in 1997

- **What is MPI good for?**
  - Abstracts away common network communications
  - Allows lots of control without bookkeeping
  - Freedom and flexibility come with complexity
    - 300 subroutines, but serious programs with fewer than 10

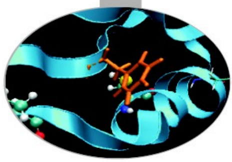
- **Basics:**
  - One executable run on every node
  - Each node process has a rank ID number assigned
  - Call API functions to send messages

http://www.mpi-forum.org/
http://forum.stanford.edu/events/2007/plenary/slides/Olukotun.ppt
http://www.tbray.org/ongoing/When/200x/2006/05/24/On-Grids

Challenges with MPI



- **Deadlock is possible…**
  - Blocking communication can cause deadlock
    - "crossed" calls when trading information
    - example:
    - `Proc1: MPI_Receive(Proc2, A);  MPI_Send(Proc2, B);`
    - `Proc2: MPI_Receive(Proc1, B);  MPI_Send(Proc1, A);`
    - There are some solutions - MPI_SendRecv()
- **Large overhead from comm. mismanagement**
  - Time spent blocking is wasted cycles
  - Can overlap computation with non-blocking comm.
- **Load imbalance is possible! Dead machines?**
- **Things are starting to look hard to code!**



Are emerging data analytics techniques the new El Dorado?

SuperComputing Applications and Innovation

Where and When using Hadoop

Where

- Batch data processing, not real-time
- Highly parallel data intensive distributed applications
- Very large production deployments

When

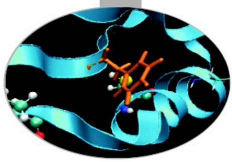
- Process lots of unstructured data
- When your processing can easily be made parallel
- Running batch jobs is acceptable
- When you have access to lots of cheap hardware

Advantages/Disadvantages

- **Now it's easy to program for many CPUs**
  - Communication management effectively gone
    - I/O scheduling done for us
  - Fault tolerance, monitoring
    - machine failures, suddenly-slow machines, etc are handled
  - Can be much easier to design and program!

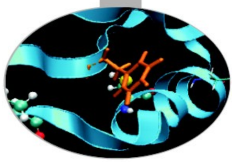
- **But … it further restricts solvable problems**
  - Might be hard to express problem in MapReduce
  - Data parallelism is key
  - Need to be able to break up a problem by data chunks
  - MapReduce is closed-source (to Google) C++
  - Hadoop is open-source Java-based rewrite

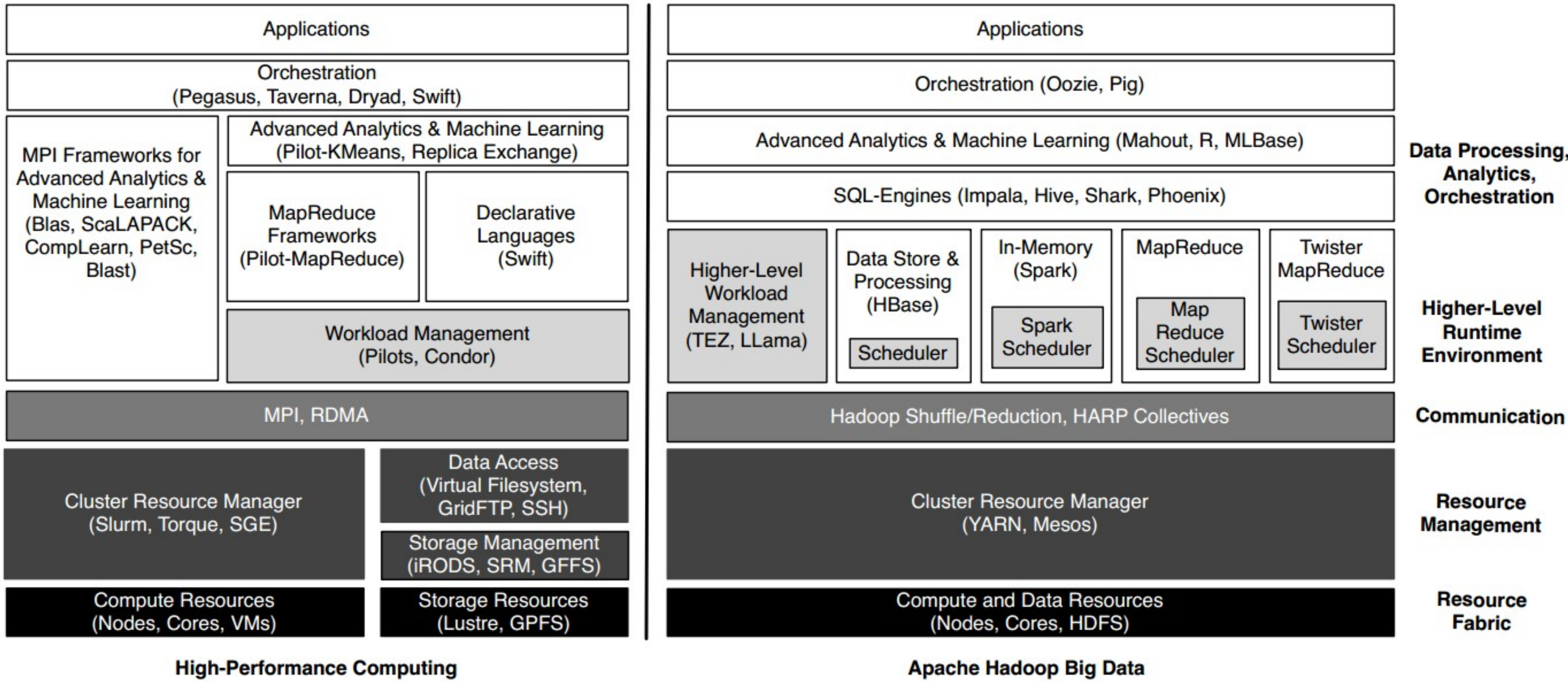
What if



- If you have access to a Hadoop cluster and you want a quick-and-dirty job…
  - *Hadoop Streaming*
- If you don’t have access to Hadoop and want to try stuff out…
  - *MrJob*
- If you’re heavily using AWS…
  - *MrJob*
- If you want to work interactively…
  - *PySpark*
- If you want to do in-memory analytics…
  - *PySpark*
- If you want to do anything…*
  - *PySpark*
- If you want ease of Python with high performance
  - *Impala + Numba*

HPC vs HPDA




Applications
Orchestration (Pegasus, Taverna, Dryad, Swift)
MPI Frameworks for Advanced Analytics & Machine Learning (Blas, ScaLAPACK, CompLearn, PetSc, Blast)
Advanced Analytics & Machine Learning (Pilot-KMeans, Replica Exchange)
MapReduce Frameworks (Pilot-MapReduce)
Declarative Languages (Swift)
Workload Management (Pilots, Condor)
MPI, RDMA
Cluster Resource Manager (Slurm, Torque, SGE)
Data Access (Virtual Filesystem, GridFTP, SSH)
Storage Management (iRODS, SRM, GFFS)
Compute Resources (Nodes, Cores, VMs)
Storage Resources (Lustre, GPFS)
High-Performance Computing
Applications
Orchestration (Oozie, Pig)
Advanced Analytics & Machine Learning (Mahout, R, MLBase)
SQL-Engines (Impala, Hive, Shark, Phoenix)
Higher-Level Workload Management (TEZ, LLama)
Data Store & Processing (HBase)
Scheduler
In-Memory (Spark)
Spark Scheduler
MapReduce
Map Reduce Scheduler
Twister MapReduce
Twister Scheduler
Hadoop Shuffle/Reduction, HARP Collectives
Cluster Resource Manager (YARN, Mesos)
Compute and Data Resources (Nodes, Cores, HDFS)
Apache Hadoop Big Data
Data Processing, Analytics, Orchestration
Higher-Level Runtime Environment
Communication
Resource Management
Resource Fabric

Parallel Computing Model

MapReduce can be classified as a SIMD (single-instruction, multiple-data) problem.

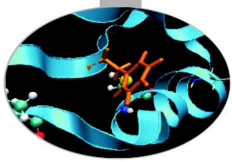
- Indeed, the map step is highly scalable because the same instructions are carried out over all data. Parallelism arises by breaking the data into independent parts with no forward or backward dependencies (side effects) within a Map step; that is, the Map step may not change any data (even its own).
- The reducer step is similar, in that it applies the same reduction process to a different set of data (the results of the Map step).
- In general, the MapReduce model provides a functional, rather than procedural, programing model. Similar to a functional language, MapReduce cannot change the input data as part of the mapper or reducer process, which is usually a large file. Such restrictions can at first be seen as inefficient; however, the lack of side effects allows for easy scalability and redundancy.

An HPC cluster, on the other hand, can run SIMD and MIMD (multiple-instruction, multiple-data) jobs.

- The programmer determines how to execute the parallel algorithm. Users, however, are not restricted when creating their own MapReduce application within the framework of a typical HPC cluster.

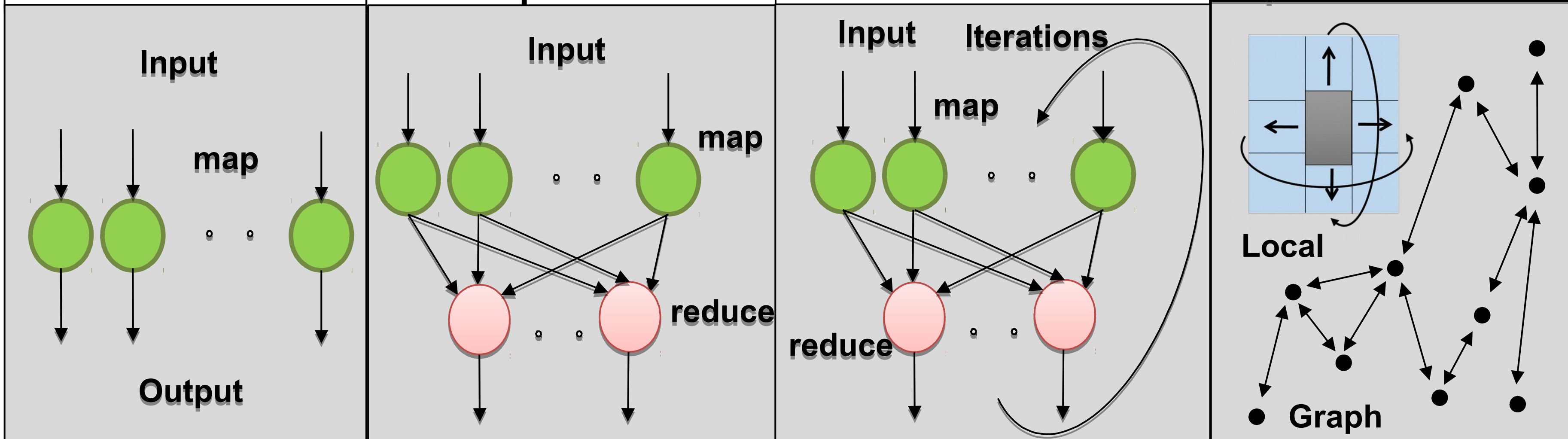
A Tale of Two Data-Intensive Paradigs: Applications, Abstractions, and Architectures
Shantenu Jha , Judy Qiu, Andre Luckow , Pradeep Mantha , Geoffrey C.Fox

Big Data Needs Big Solutions



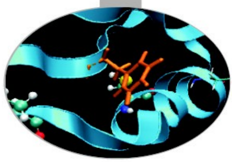
- Without a doubt, Hadoop is useful when analyzing very large data files.
- HPC has no shortage of “big data” files
- Provided your problem fits into the MapReduce framework, Hadoop is a powerful way to operate on staggeringly large data sets. Because both the Map and Reduce steps are user defined, highly complex operations can be encapsulated in these steps.
- The growth of Hadoop and the hardware on which it runs has been increasing. Certainly it can be seen as a subset of HPC, offering a single yet powerful algorithm that has been optimized for a large number of commodity servers.

(1) Map Only	(2) Classic MapReduce	(3) Iterative Map Reduce or Map-Collective	(4) Point to Point or Map-Communication



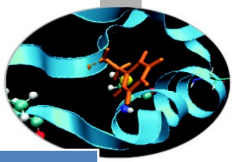
BLAST Analysis Local Machine Learning Pleasingly Parallel	High Energy Physics (HEP) Histograms Distributed search Recommender Engines	Expectation maximization Clustering e.g. K-means Linear Algebra, PageRank	Classic MPI PDE Solvers and Particle Dynamics Graph Problems
MapReduce and Iterative Extensions (Spark, Twister)			MPI, Giraph
Integrated Systems such as Hadoop + Harp with Compute and Communication model separated			

Correspond to first 4 of Identified Architectures



The PICO system

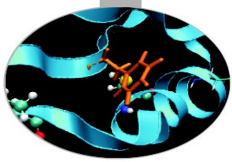
The PICO system



	Total Nodes	CPU	Cores per Nodes	Memory (RAM)	Notes
Compute login node	66	Intel Xeon E5 2670 v2 @2.5Ghz	20	128 GB	
Visualization node	2	Intel Xeon E5 2670 v2 @ 2.5Ghz	20	128 GB	2 GPU Nvidia K40
Big Mem node	2	Intel Xeon E5 2650 v2 @ 2.6 Ghz	16	512 GB	1 GPU Nvidia K20
BigInsight node	4	Intel Xeon E5 2650 v2 @ 2.6 Ghz	16	64 GB	32TB of local disk
SSD Storage					40 TB

http://www.hpc.cineca.it/hardware/pico

PICO: how to log in



- Establish a ssh connection

  `ssh <username>@login.pico.cineca.it`

- Notes:
  - **ssh** available on all linux distros
  - **Putty** (free) or **Tectia** ssh on Windows
  - *secure shell plugin* for Google Chrome!
  - login nodes are swapped to keep the load balanced
  - important messages can be found in the *message of the day*

SCAI
SuperComputing Applications and Innovation

Working environment

$HOME:

- Permanent, backed-up, and local to PICO.
- For source code or important input files.

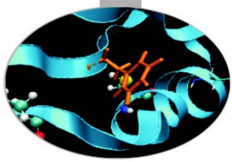
$CINECA_SCRATCH:

- Large, parallel filesystem (GPFS).
- No quota. Run your simulations and calculations here.

- use the command `cindata` command to get info on your disk occupation

http://www.hpc.cineca.it/content/data-storage-and-filesystems-0

"module", my best friend

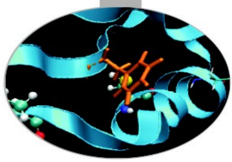


- **All the optional software on the system is made available through the "module" system**
  - provides a way to rationalize software and its environment variables

- Modules are divided in 2 *profiles*
  - **profile/base** (stable and tested modules)
  - **profile/advanced** (software not yet tested or not well optimized)

- Each profile is divided in 4 categories
  - **compilers** (GNU, intel, openmpi)
  - **libraries** (e.g. LAPACK, BLAS, FFTW, ...)
  - **tools** (e.g. Hadoop, GNU make, VNC, ...)
  - **applications** (software for chemistry, physics, ...)

Modules



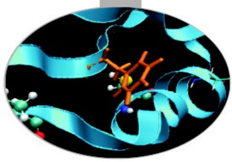
- CINECA’s work environment is organized in modules, a set of installed libraries, tools and applications available for all users.

- “loading” a module means that a series of (useful) shell environment variables will be set

- E.g. after a module is loaded, an environment variable of the form “<MODULENAME>_HOME” is set

```
[amarani0@fen07 ~]$ module load namd
[amarani0@fen07 ~]$ ls $NAMD_HOME
backup  flipbinpdb  flipdcd  namd2  namd2_plumed  namd2_remd  psfgen  sortreplicas
```

Module commands



COMMAND	DESCRIPTION
module avail	list all the available modules
module load <module_name(s)>	load module <module_name>
module list	list currently loaded modules
module purge	unload all the loaded modules
module unload <module_name>	unload module <module_name>
module help <module_name>	print out the help (hints)
module show <module_name>	print the env. variables set when loading the module

Launching a Job

- Now that we have our executable, it’s time to learn how to prepare a job for its execution
- PICO uses **PBS scheduler**.
- The job script scheme is:

```
#!/bin/bash
#PBS keywords

variables environment

execution line
```

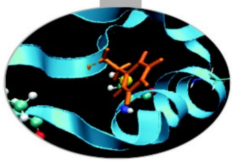
PBS keywords

```
#PBS -N jobname # name of the job
#PBS -o job.out   # redirect stdout (output file)
#PBS -e job.err   # redirect stderr (error file)
#PBS -l select=1:ncpus=20::mem=96gb   # resources
#PBS -l walltime=1:00:00      # hh:mm:ss
#PBS -q <queue-name>  # chosen queue
#PBS -A <my_account>    # name of the account
```

select = number of chunk requested

ncpus = number of cpus per chunk requested

mem = RAM memory per chunk



PBS keyword - resource

Memory per node:

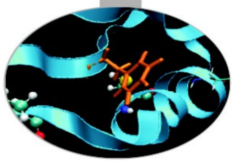
- The default memory is 1 GB per node (for the classes debug, parallel and longpar).
- The user can specify the requested memory up to 128 GB, on 58 nodes

```
#PBS -l select=NN:ncpus=CC:mem=128GB
```

PBS job script – Serial using 1 GPU

```
#!/bin/bash
#PBS -l walltime=30:00
#PBS -l select=1:ncpus=1
#PBS -o job.out
#PBS -e job.err
#PBS -q debug
#PBS -A train_cmda2014

cd $PBS_O_WORKDIR

./myProgram
```

PBS Commands



qsub

qsub <job_script>

Your job will be submitted to the PBS scheduler and executed when there will be nodes available (according to your priority and the queue you requested)

qstat

qstat

Shows the list of all your scheduled jobs, along with their status (idle, running, closing, …) Also, shows you the job id required for other qstat options

PBS Commands

qstat

qstat -f <job_id>

Provides a long list of informations for the job requested.

In particular, if your job isn't running yet, you'll be notified about its estimated start time or, if you made an error on the job script, you will learn that the job won't ever start

qdel

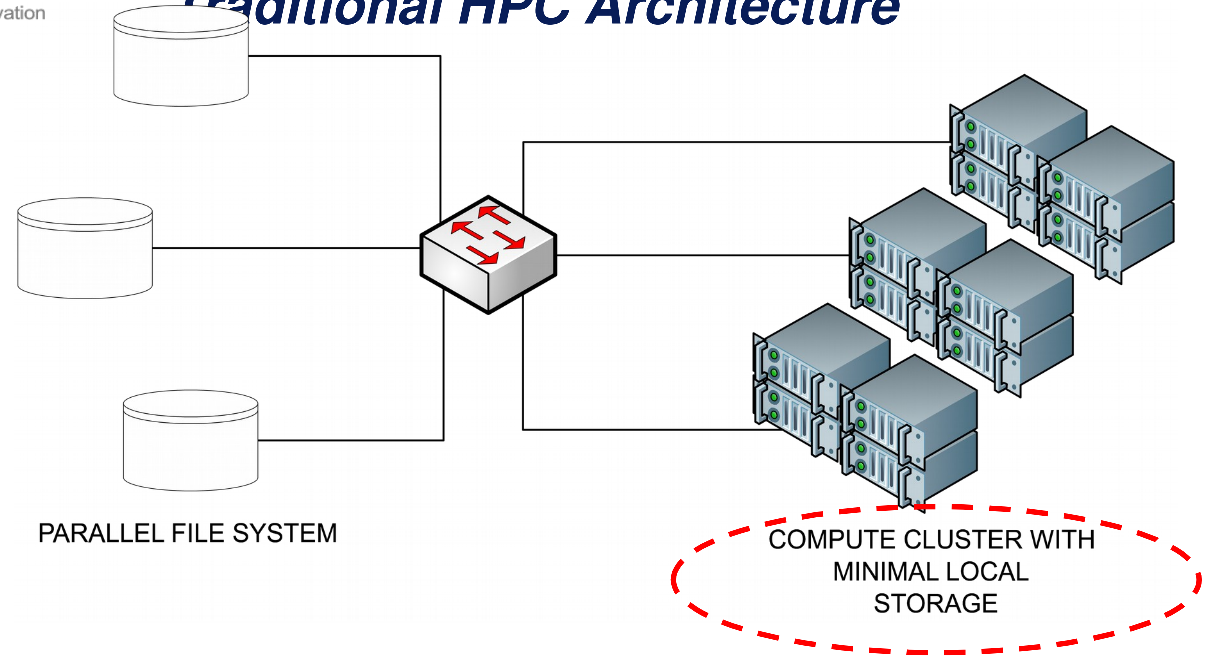
qdel <job_id>

Removes the job from the scheduled jobs by killing it

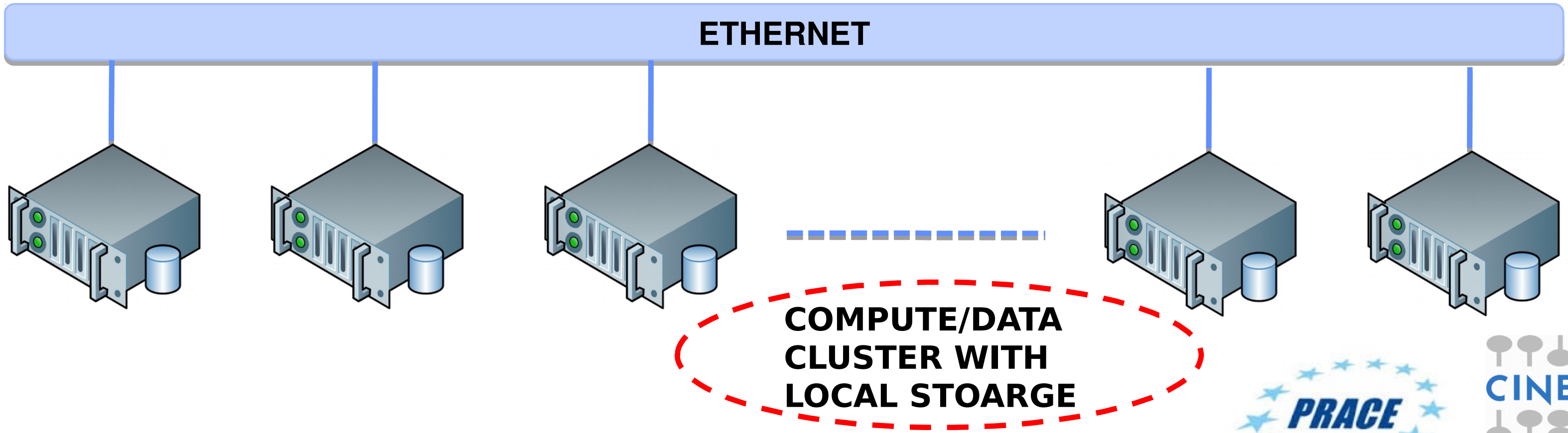
SuperComputing Applications and Innovation

Hadoop on PICO

Traditional HPC Architecture



Shared-nothing (MapReduce-style) Architectures



PBS Script

```
#!/bin/bash
#PBS -A <account>
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=20:mem=96GB
#PBS -q parallel

## Environment configuration
module load profile/advanced hadoop/1.2.1
# Configure a new HADOOP instance using PBS job information
$MYHADOOP_HOME/bin/myhadoop-configure.sh -c $HADOOP_CONF_DIR
# Start the Datanode, Namenode, and the Job Scheduler
$HADOOP_HOME/bin/start-all.sh
#####################################

# Your job goes here

# Stop HADOOP services
$MYHADOOP_HOME/bin/myhadoop-shutdown.sh
```

Sample execution

- Login on PICO
  - **`ssh login.pico.cineca.it -l <username>`**
- Download source codes within **$HOME** or **$CINECA_SCRATCH**
- Change the selected PBS script accordingly to the destination directory
- ***`qsub $HOME/course-exercises/pbs/mrjob/wordcount/word-count.hadooop.pbs`***
- ***`qstat`***

Sample execution (cont.)

Output:

- word-count.hado.o3041 **// std output**
- word-count.hado.o3042 **// std error**

Credits

- **Geoffrey C. Fo**x – Indiana University
- **Hanspeter Pfister and Joe Blitzstein** – Harvard University
- **Borja Sotomayor** – University of Chicago
- **Glenn K. Lockwood** – High-Performance and Data-Intensive Computing San Francisco Bay Area