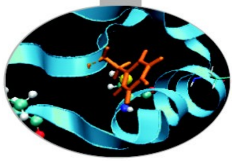
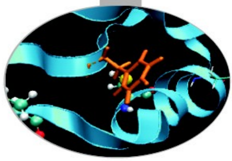


Welcome!

Tool and techniques for massive data processing

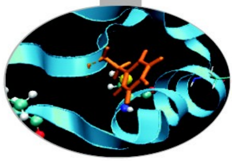
Giuseppe Fiameni
April 12th 2016

Why this course?



- 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 common technologies, although are the only viable solutions in many cases, have a steep learning curve which limits their wide adoption

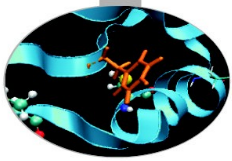
Goals



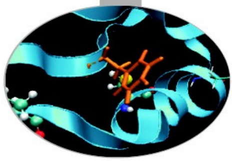
During this course, you will learn:

- the trends and challenges surrounding the BigData definition
- how the most relevant technologies and methods in this domain work
  - *Map-Reduce*
  - *Apache Hadoop*
  - *Apache Spark, Apache SparkSQL*
  - *Apache Spark MLLIB (Machine Learning Library)*
- how to structure and program your code using Python/Scala

General notes



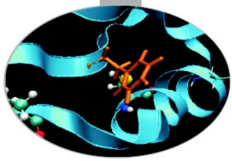
- Each participant will receive personal credentials to access Cineca resources for a limited amount of time
    - requests for extensions are possible
- **There is one workstation every two participants**:
    - **U**: corsi **P**: corsi_2013!
- **Course web site**:
    - https://goo.gl/Pq03qz
- Feel free to ask questions at any time → do not be shy!



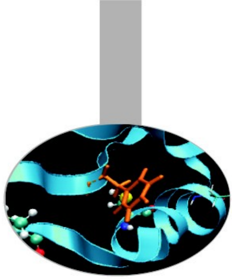
	Thursday, April 14	
09.00 - 09.30	Apache SparkR	*Giorgio Pedrazzi* *Giovanni Simonini* *Song Zhu***
09.30 - 10.00	Apache Spark Exercises	*Giovanni Simonini* *Giorgio Pedrazzi* *Song Zhu*
10.00 - 12.30	Apache Spark Exercises	*Giovanni Simonini* *Giorgio Pedrazzi* *Song Zhu*
12.30 - 14.00	**Lunch Break**	
14.00 - 15.30	Apache Spark MLLib	*Giorgio Pedrazzi* *Song Zhu*
15.30 - 17.30	Apache Spark MLLib	*Giorgio Pedrazzi* *Song Zhu*

*Giuseppe Fiameni, Paolo D'Onorio De Meo, Giorgio Pedrazzi - **CINECA**

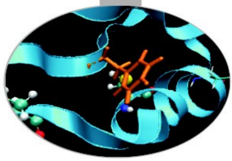
Giovanni Simoni, Song Zhu - **University of Modena and Reggio Emilia



	Tuesday, April 12	
09.00 - 09.30	Registration	
09.30 - 09.40	Introduction/logistics/agenda	*Giuseppe Fiameni**
09.40 - 10.30	Big Data and HPC	*Giuseppe Fiameni*
10.30 - 11.00	**Coffee break**	
11.00 - 11.40	Brief introduction to Map-Reduce and Hadoop (Hadoop, HDFS, Yarn)	*Giuseppe Fiameni*
11.40 - 12.30	Set-up of the environment	*Paolo D'Onorio De Meo** *Giuseppe Fiameni*
12.30 - 14.00	**Lunch break**	
14.00 - 15.00	Brief introduction to iPython MrJob/Hadoop Exercies	*Paolo D'Onorio De Meo* *Giuseppe Fiameni*
15.00 - 15.30	**Coffee Break**	
15.30 - 17.30	MrJob/Hadoop Exercises	*Paolo D'Onorio De Meo* *Giuseppe Fiameni*
17.30	**End of the first day**	



	Wednesday, April 13	
09.30 - 11.00	MrJob/Hadoop Exercises	*Paolo D'Onorio De Meo* *Giuseppe Fiameni*
11.00 - 11.15	**Coffee break**	
11.15 - 12.30	MrJob/Hadoop Exercises (NGS Use Case)	*Paolo D'Onorio De Meo* *Giuseppe Fiameni*
12.30 - 14.00	**Lunch break**	
14.00 - 15.00	Apache Spark	*Giovanni Simonini*** *Giorgio Pedrazzi**
15.00 - 16.00	Apache Spark Examples	*Giovanni Simonini* *Giorgio Pedrazzi*
16.00 - 16.30	**Coffee break**	
16.30 - 17.30	Apache SparkSQL	*Giovanni Simonini* *Giorgio Pedrazzi*
17.30	**End of the second day**	

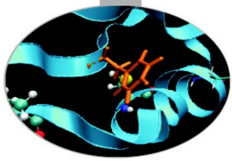


	Thursday, April 14	
09.00 - 09.30	Apache SparkR	*Giorgio Pedrazzi* *Giovanni Simonini* *Song Zhu***
09.30 - 10.00	Apache Spark Exercises	*Giovanni Simonini* *Giorgio Pedrazzi* *Song Zhu*
10.00 - 12.30	Apache Spark Exercises	*Giovanni Simonini* *Giorgio Pedrazzi* *Song Zhu*
12.30 - 14.00	**Lunch Break**	
14.00 - 15.30	Apache Spark MLLib	*Giorgio Pedrazzi* *Song Zhu*
15.30 - 17.30	Apache Spark MLLib	*Giorgio Pedrazzi* *Song Zhu*

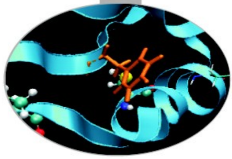
*Giuseppe Fiameni, Paolo D'Onorio De Meo, Giorgio Pedrazzi - **CINECA**

Giovanni Simoni, Song Zhu - **University of Modena and Reggio Emilia

Quick poll

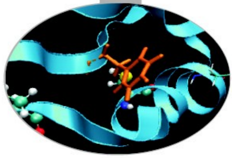


- How many of you already know part of the mentioned Big Data technologies?
- How many of you know any HPC methods (MPI, OpenMP, etc.)?
- How many of you know Python?
- Scala?



Before starting

Size of computational applications



Computational Dimension:

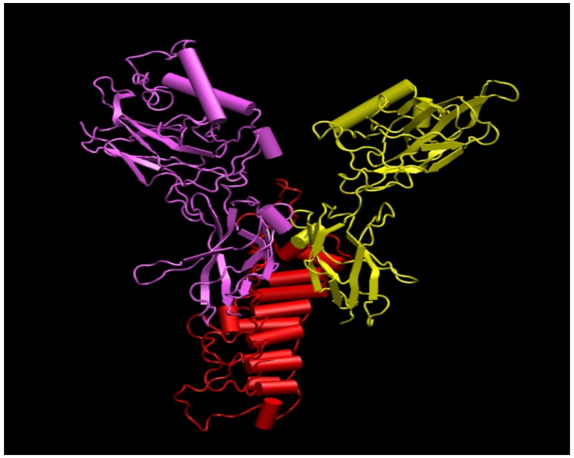
number of operations needed to solve the problem, in general is a function of the size of the involved data structures (n, n^2 , n^3 , n log n, etc.)

flop - Floating point operations

indicates an arithmetic floating point operation.

flop/s - Floating points operations per second

is a unit to measure the speed of a computer.

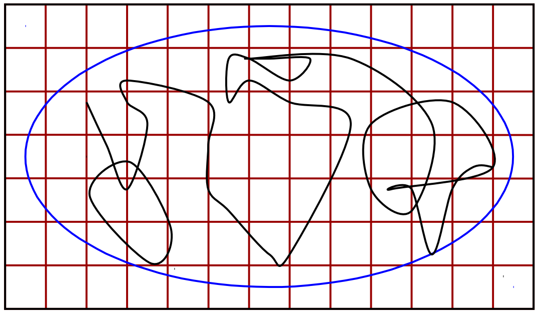


Computational problems today: 10^{15} – 10^{22} flop

One year has about 3 x 10^7 seconds!

Most powerful computers today have reach a sustained performance is of the order of Tflop/s - Pflop/s (10^{12} - 10^{15} flop/s).

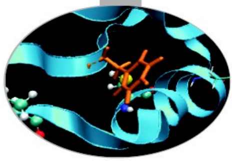
Example: Weather Prediction



Forecasts on a global scale:

- **3D Grid to represent the Earth**
  - Earth's circumference: ≅ 40000 km
  - radius: ≅ 6370 km
  - Earth's surface: ≅ $4\pi r^2$ ≅ $5 \bullet 10^8$ km^2
- **6 variables:**
  - temperature
  - pressure
  - humidity
  - wind speed in the 3 Cartesian directions
- cells of 1 km on each side
- 100 slices to see how the variables evolve on the different levels of the atmosphere
- a 30 seconds time step is required for the simulation with such resolution
- Each cell requires about 1000 operations per time step (Navier-Stokes turbulence and various phenomena)

Example: Weather Prediction / 1



Grid: $5 \bullet 10^{8} \bullet 100 = 5 \bullet 10^{10}$ cells

- each cell is represented with 8 Byte
- Memory space:
  - ➔ (6 var)•(8 Byte)•($5 \bullet 10^{10}$ cells) $\cong 2 \bullet 10^{12}$ Byte = 2TB

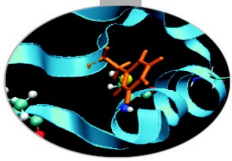
A 24 hours forecast needs:

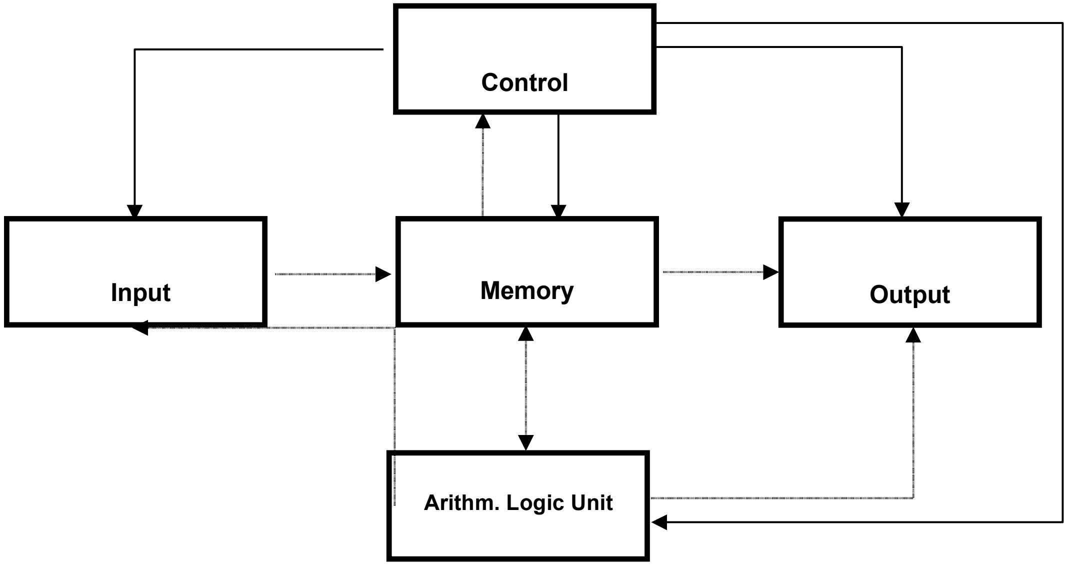
- ➔ $24 \bullet 60 \bullet 2 \cong 3 \bullet 10^{3}$ time-step
- ➔ ($5 \bullet 10^{10}$ cells) • (10^{3} oper.) • ($3 \bullet 10^{3}$ time-steps) = $1.5 \bullet 10^{17}$ operations !

A computer with a power of 1Tflop/s will take $1.5 \bullet 10^{5}$ sec.

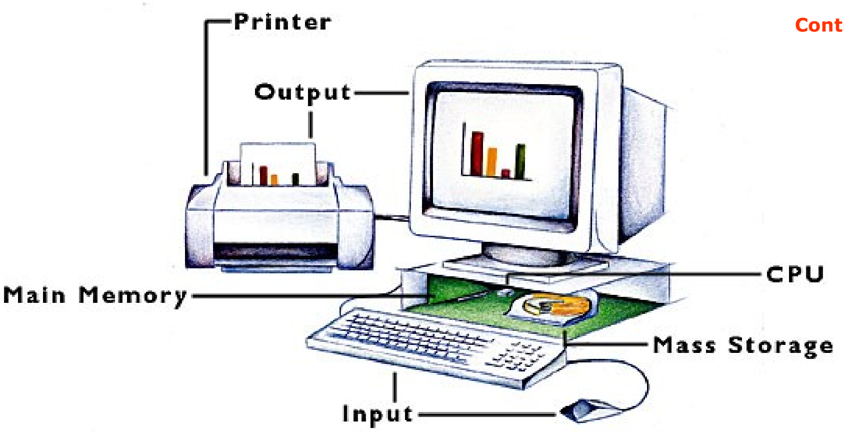
➔ **24 hours forecast will need 2days to run ... but we shall obtain a very accurate forecast**

Von Neumann Model





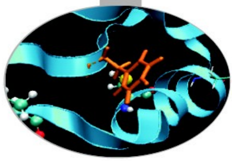
Von Neumann Model of Computer Architecture



Instructions are processed sequentially

- A single instruction is loaded from memory (fetch) and decoded
- Compute the addresses of operands
- Fetch the operands from memory;
- Execute the instruction ;
- Write the result in memory (store).

Speed of Processors: Clock Cycle and Frequency



The *clock cycle* τ is defined as the time between two adjacent pulses of oscillator that sets the time of the processor.

The number of these pulses per second is known as clock speed or clock frequency, generally measured in GHz (gigahertz, or billions of pulses per second).

The clock cycle controls the synchronization of operations in a computer: All the operations inside the processor last a multiple of τ.

Processor	τ (ns)	freq (MHz)
CDC 6600	100	10
Cyber 76	27.5	36,3
IBM ES 9000	9	111
Cray Y-MP C90	4.1	244
Intel i860	20	50
PC Pentium	< 0.5	> 2 GHz
Power PC	1.17	850
IBM Power 5	0.52	1.9 GHz
IBM Power 6	0.21	4.7 GHz

Increasing the clock frequency:

The **speed of light** sets an upper limit to the speed with which electronic components can operate .

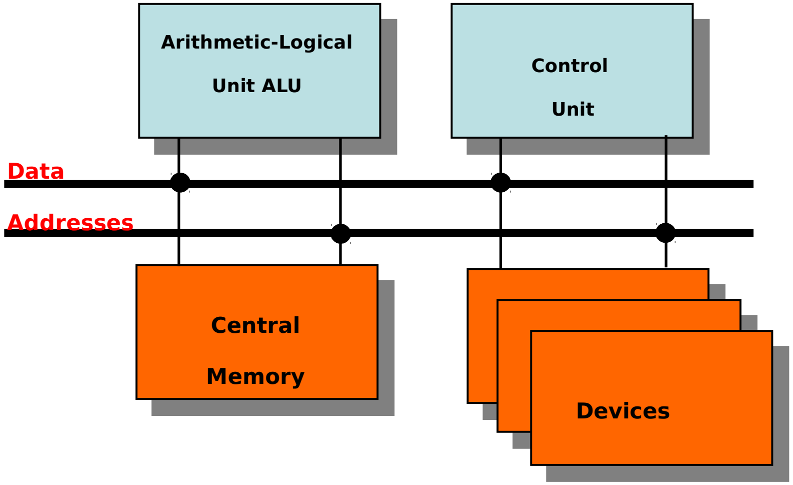
Propagation velocity of a signal in a vacuum: **300.000 Km/s = 30 cm/ns**

Heat dissipation problems inside the processor. Also Quantum tunelling expected to become important.

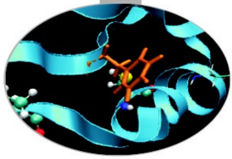
Other factors that affect Performance

In addition to processor power, other factors affect the performance of computers:

- Size of memory
- Bandwidth between processor and memory
- Bandwidth toward the I/O system
- Size and bandwidth of the cache
- Latency between processor, memory, and I/O system

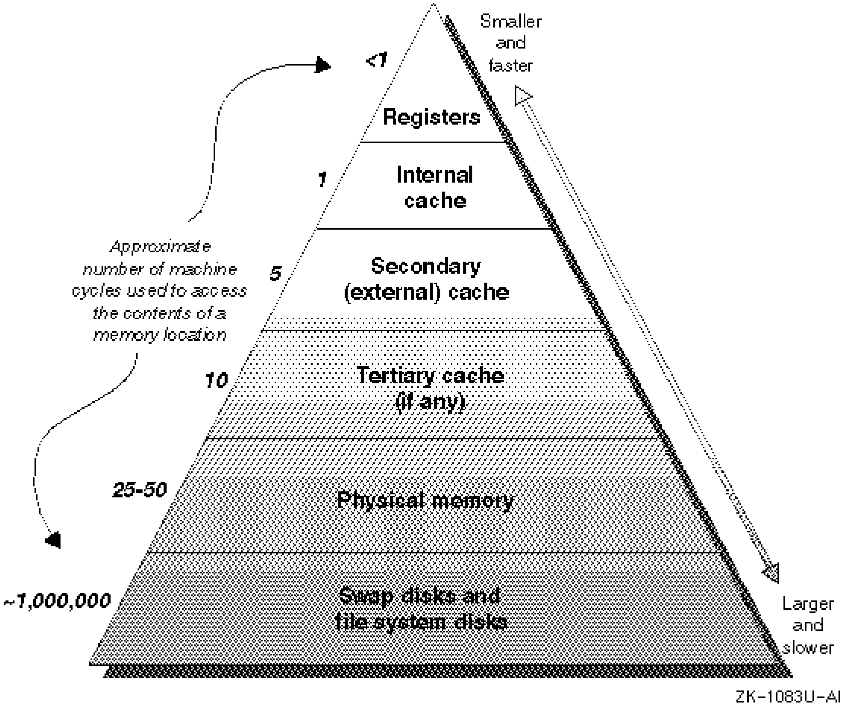


Memory hierarchies



Time to run code = clock cycles running code + clock cycles waiting for memory

Memory access time: the *time* required by the processor to *access* data or to write data from / to *memory*



The hierarchy exists because :

- fast memory is expensive and small
- slow memory is cheap and big

Latency

- how long do I have to wait for the data?
- (cannot do anything while waiting)

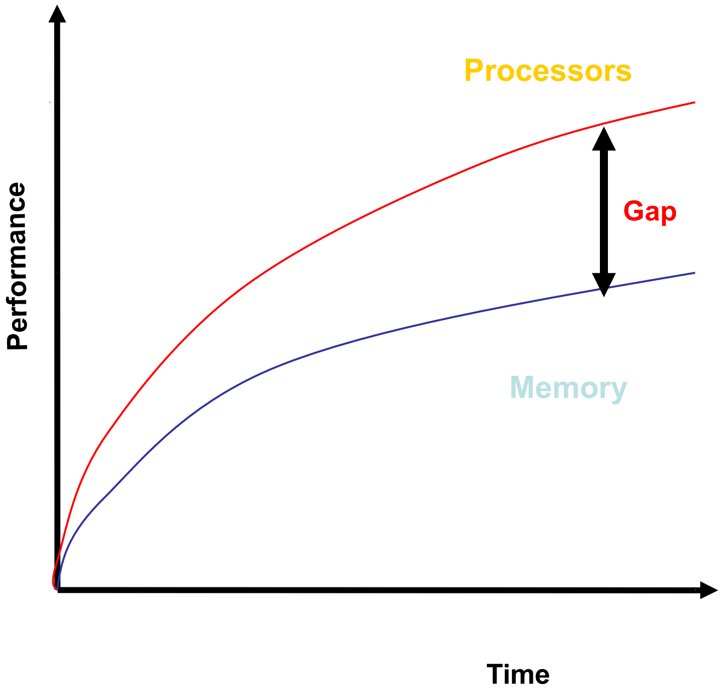
Throughput

- how many bytes/second. but not important if waiting.

Total time = latency + (amount of data / throughput)

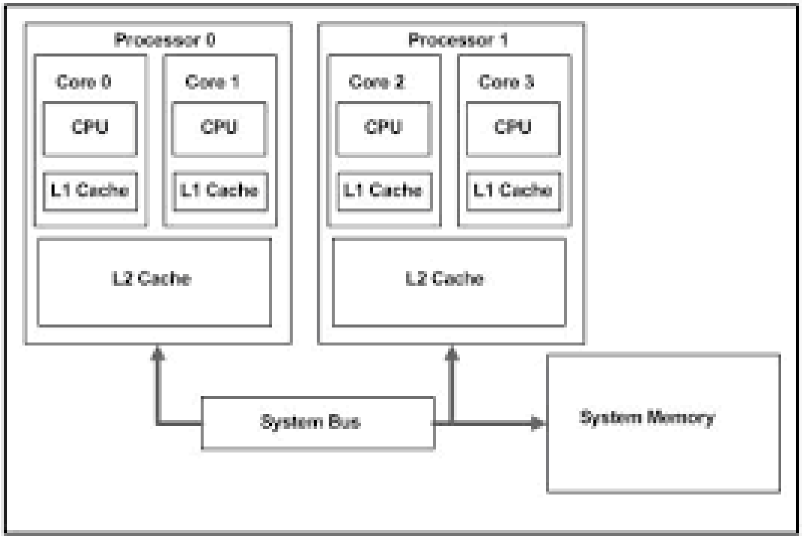
Memory access

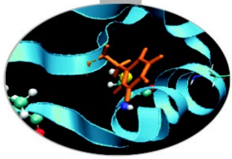
- Important problem for the performance of any computer is access to main memory. Fast processors are useless if memory access is slow!
- Over the years the difference in speed between processors and main memory has been growing.



Cache Memory

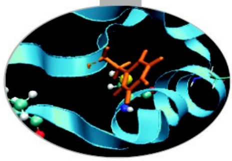
- High speed, small size memory used as a buffer between the main memory and the processor. When used correctly, reduces the time spent waiting for data from main memory.
- Present as various "levels" (e.g. L1, L2, L3, etc) according to proximity to the functional units of the processor.
- Cache efficiency depends on the locality of the data references:
  - *Temporal locality* refers to the re-use of data within relatively small time frame.
  - *Spatial locality refers to the use of data within close storage locations (e.g. one dimensional array).*
- *Cache can contain Data, Instructions or both.*



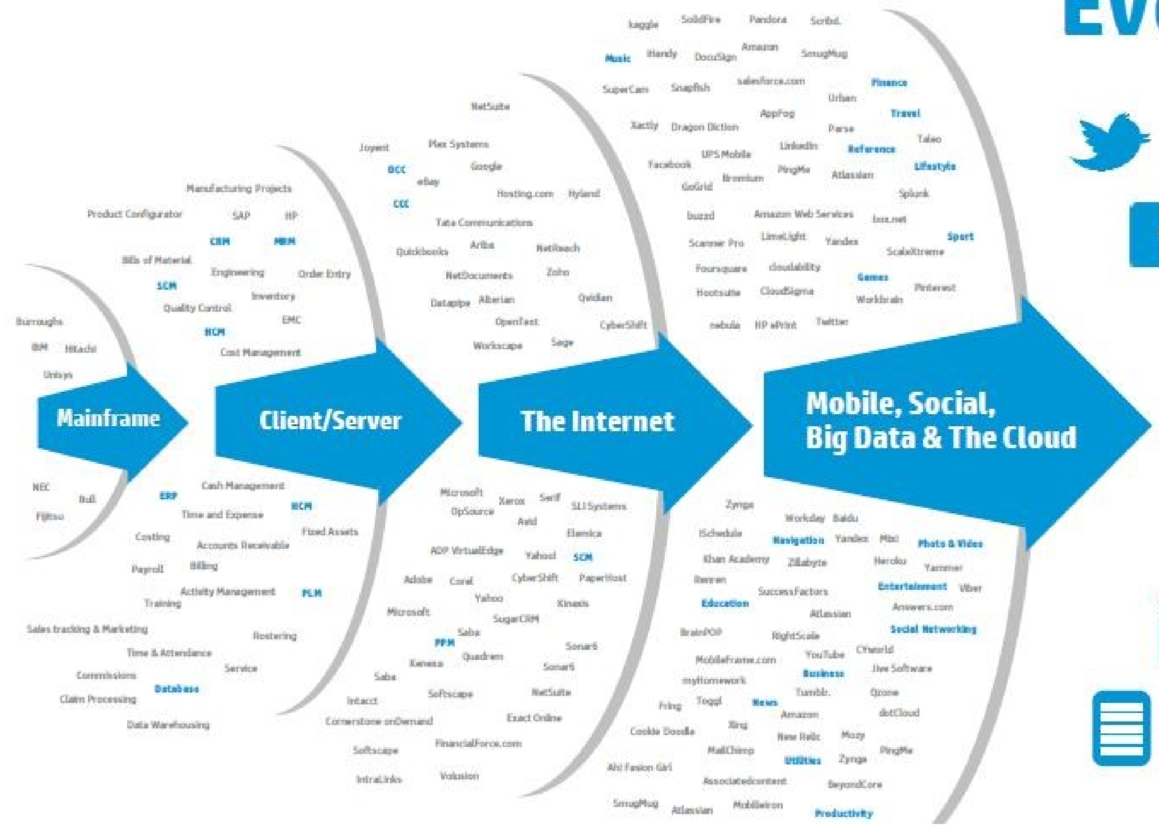


Aspects of parallelism

- It has been recognized for a long time that constant performance improvements cannot be obtained just by increasing factors such as processor clock speed – parallelism is needed.
- Parallelism can be present at many levels:
  - Functional parallelism within the CPU
  - Pipelining and vectorization
  - Multi-processor and multi-core
  - Accelerators
  - Parallel I/O



A new style of IT emerging



Every 60 seconds

98,000+ tweets

695,000 status updates

11million instant messages

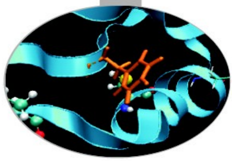
698,445 Google searches

168 million+ emails sent

1,820TB of data created

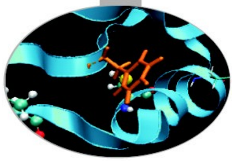
217 new mobile web users

Why?



- **Overwhelming amounts of data generated** by all kinds of devices, networks and programs, *e.g. sensors, mobile devices, internet, social networks, computer simulations, satellites, radiotelescopes, LHC, etc.*
- **Increasing storage capacity**
  - ✓ *Storage capacity has doubled every 3 years since 1980 with prices steadily going down*
  - ✓ *1,8 zetabytes: an estimation for the data stored by humankind in 2011 (Digital Universe study of International Data Corporation)*
- **Massive data can produce high-value information and knowledge**
- **Critical for data analysis, decision support, forecasting, business intelligence, research, (data-intensive) science, etc.**

Big Data



A buzz word!

- With different meanings depending on your perspective - e.g. 100 TBs is big for a transaction processing system, but small for a world-wide search engine

A simple “definition” (Wikipedia)

- ***Consists of data sets that grow so large that they become awkward to work with using on-hand database management tools***
  - Difficulties: capture, storage, search, sharing, analytics, visualizing

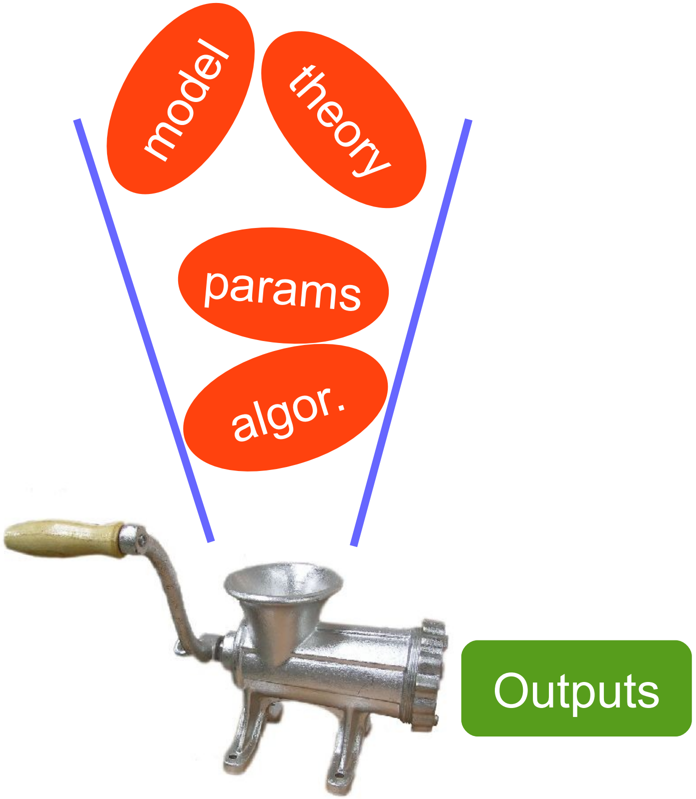
How big is big?

- Moving target: terabyte (1012 bytes), petabyte (1015 bytes), exabyte (1018), zetabyte (1021)

Scale is only one dimension of the problem!

Operate without models

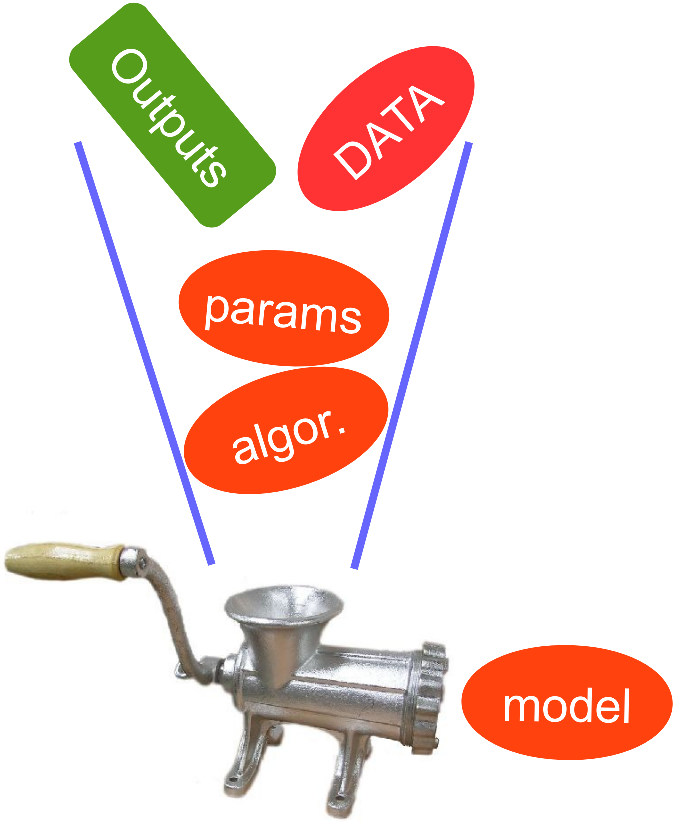
forward problem



Before...

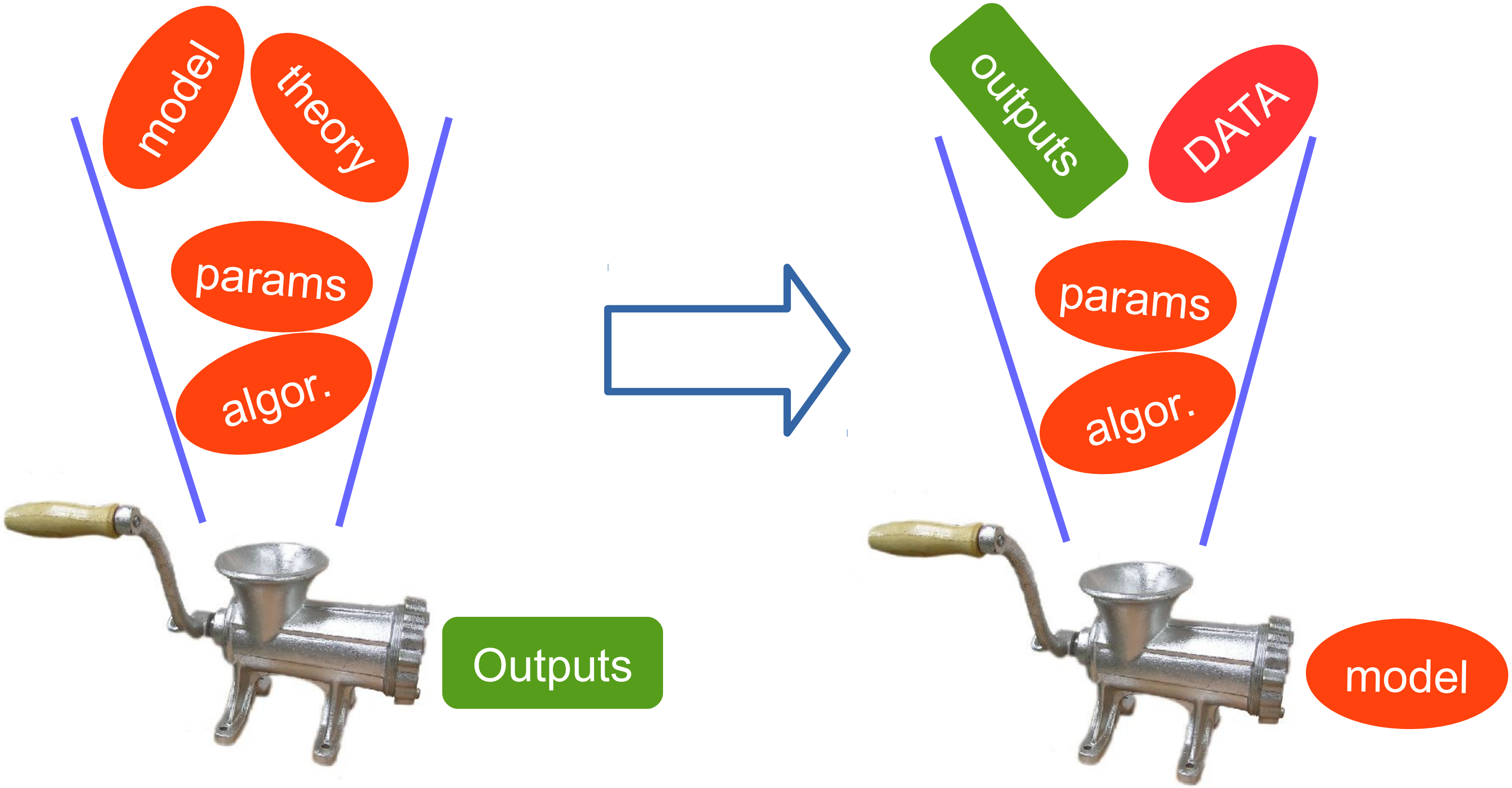
Operate without models

inverse problem



Now, future...

Operate without models (Big Data)



model

theory

params

algor.

Outputs

outputs

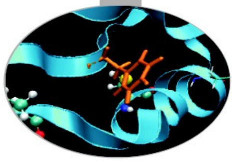
DATA

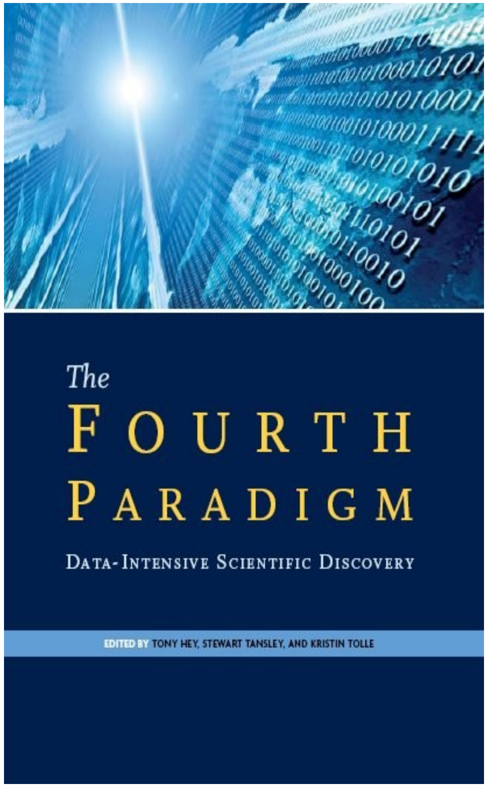
params

algor.

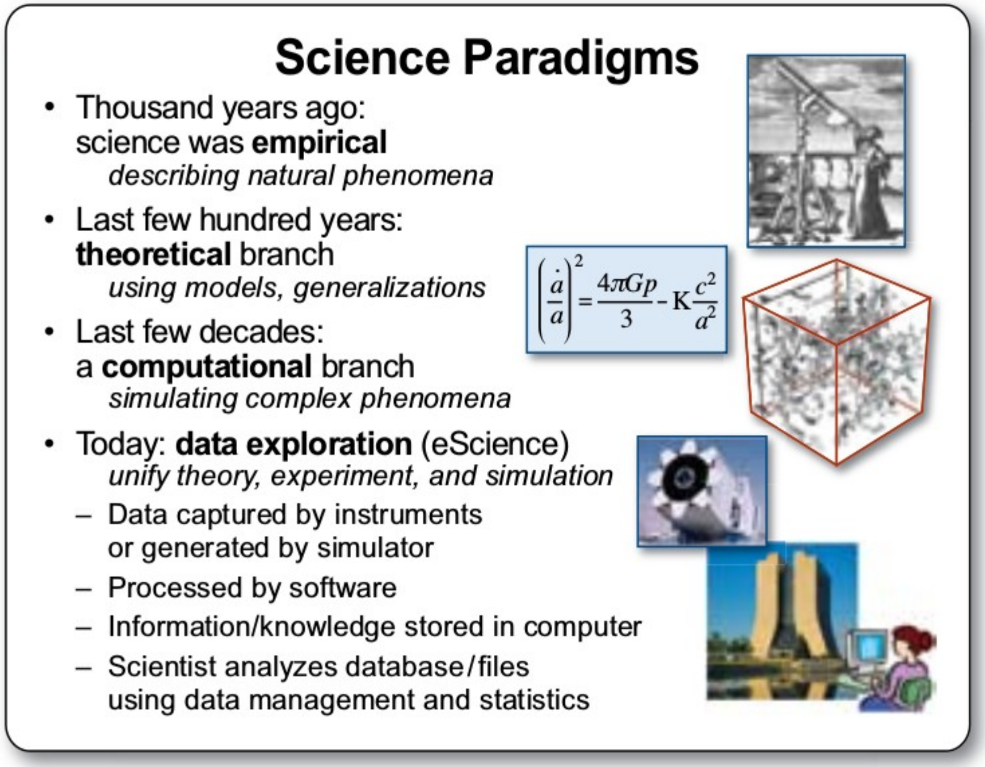
model

Where all began...a decade ago!





The
FOURTH
PARADIGM
DATA-INTENSIVE SCIENTIFIC DISCOVERY
EDITED BY TONY HEY, STEWART TANSLEY, AND KRISTIN TOLLE



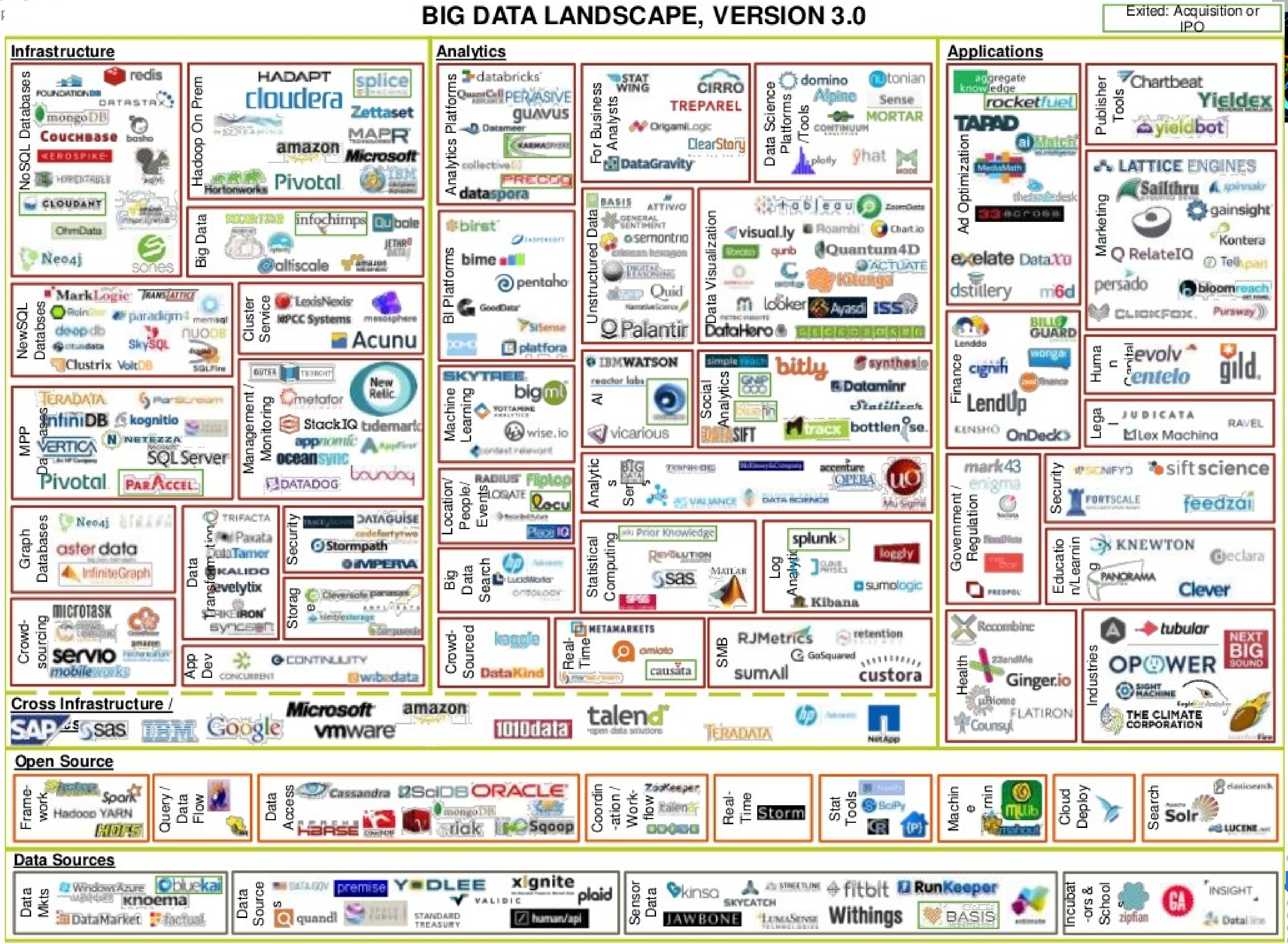
Science Paradigms

- Thousand years ago:
  science was **empirical**
  describing natural phenomena
- Last few hundred years:
  theoretical branch
  using models, generalizations
- Last few decades:
  a **computational** branch
  simulating complex phenomena
- Today: **data exploration** (eScience)
  unify theory, experiment, and simulation
  – Data captured by instruments
  or generated by simulator
  – Processed by software
  – Information/knowledge stored in computer
  – Scientist analyzes database/files
  using data management and statistics

$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{4\pi Gp}{3} - K\frac{c^2}{a^2}$$

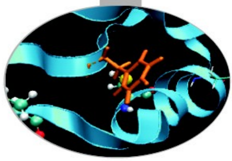
http://research.microsoft.com/en-us/collaboration/fourthparadigm/4th_paradigm_book_complete_lr.pdf

BIG DATA LANDSCAPE, VERSION 3.0

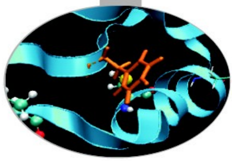


© Matt Turck (@mattturck), Sutian Dong (@sutiandong) & FirstMark Capital (@firstmarkcap)

Dimensions of the problem

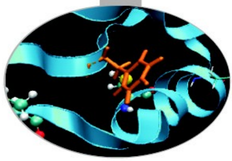


- **Volume**
  - Refers to massive amounts of data
  - Makes it hard to store and manage, but also to analyze (big analytics)
- **Velocity**
  - Continuous data streams are being captured (e.g. from sensors or mobile devices) and produced
  - Makes it hard to perform online processing
- **Variety**
  - Different data formats (sequences, graphs, arrays, …), different semantics, uncertain data (because of data capture), multiscale data (with lots of dimensions)
  - Makes it hard to integrate and analyze



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

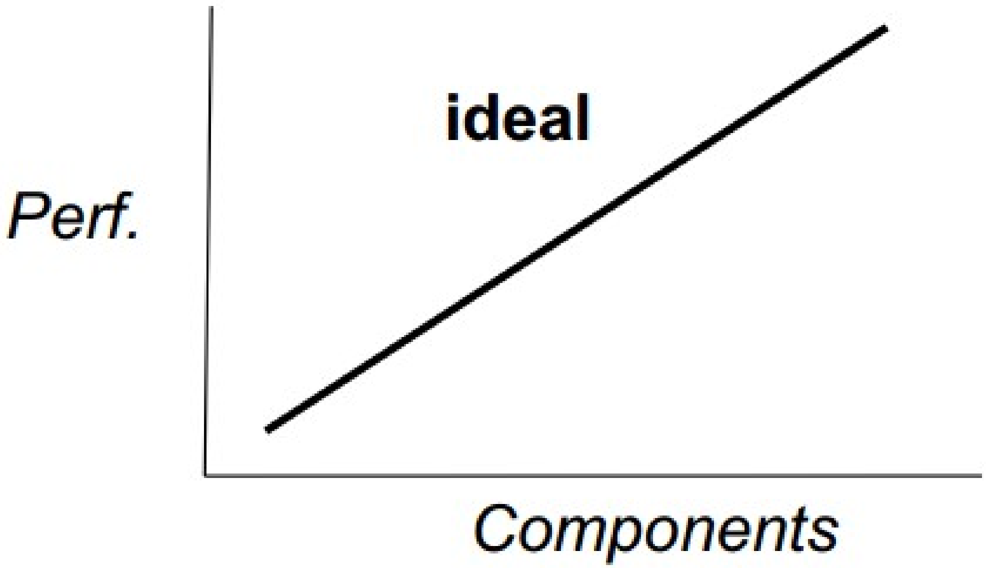
Parallel data processing!



- **Exploit a massively parallel computer**
  - A computer that interconnects lots of CPUs, RAM and disk units
- **To obtain**
  - High performance through data-based parallelism
    - ➔ *High throughput for transaction-oriented (OLTP) loads*
    - ➔ *Low response time for decision-support (OLAP) queries*
  - *High availability and reliability through data replication*
  - Extensibility with the ideal goals
    - ➔ *Linear speed-up*
    - ➔ *Linear scale-up*

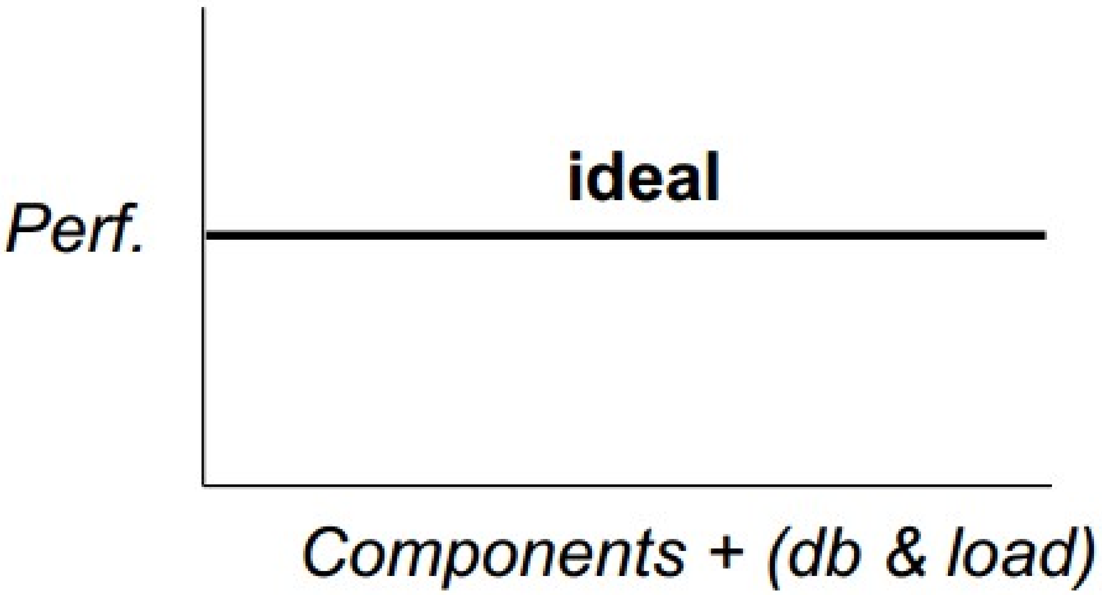
Speed-up

- Linear increase in performance for a constant database size and load, and proportional increase of the system components (CPU, memory, disk)



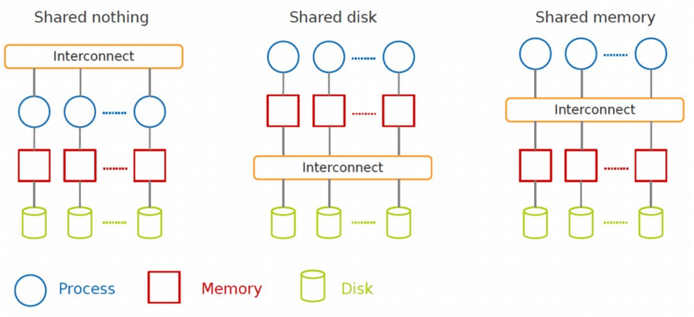
Scale-up

- Sustained performance for a linear increase of database size and load, and proportional increase of components



Parallel Architectures for Data Processing

- Three main alternatives, depending on how processors,memory and disk are interconnected
  - ***Shared-memory computer***
  - ***Shared-disk cluster***
  - ***Shared-nothing cluster***



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

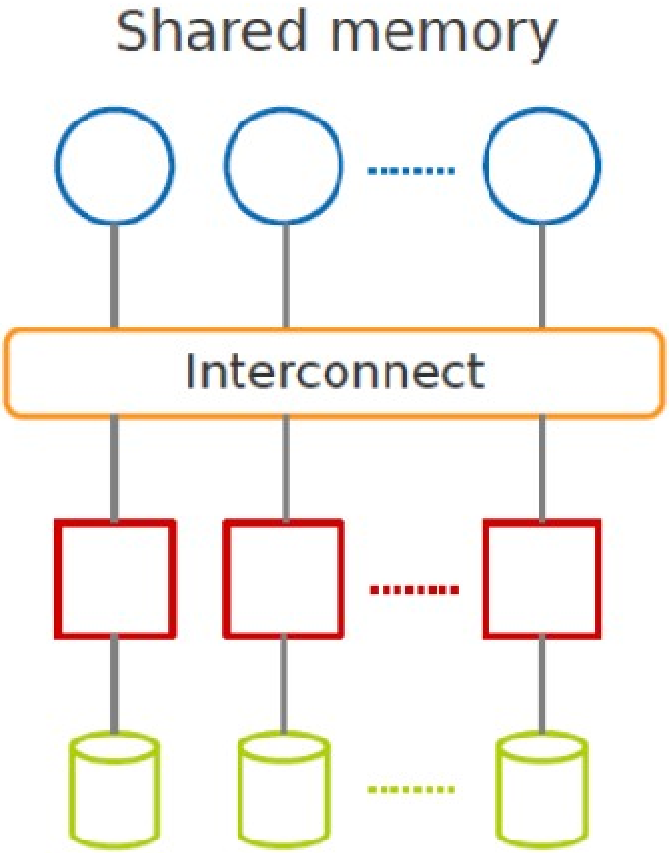
Shared Memory

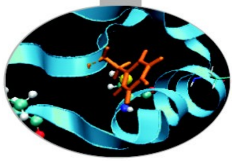
- All memory and disk are shared
  - Symmetric Multiprocessor (SMP)
  - Recent: Non Uniform Memory

<u>+ Simple for apps, fast com., load balancing</u>

- Complex interconnect limits extensibility, cost

- For write-intensive workloads, not for big data





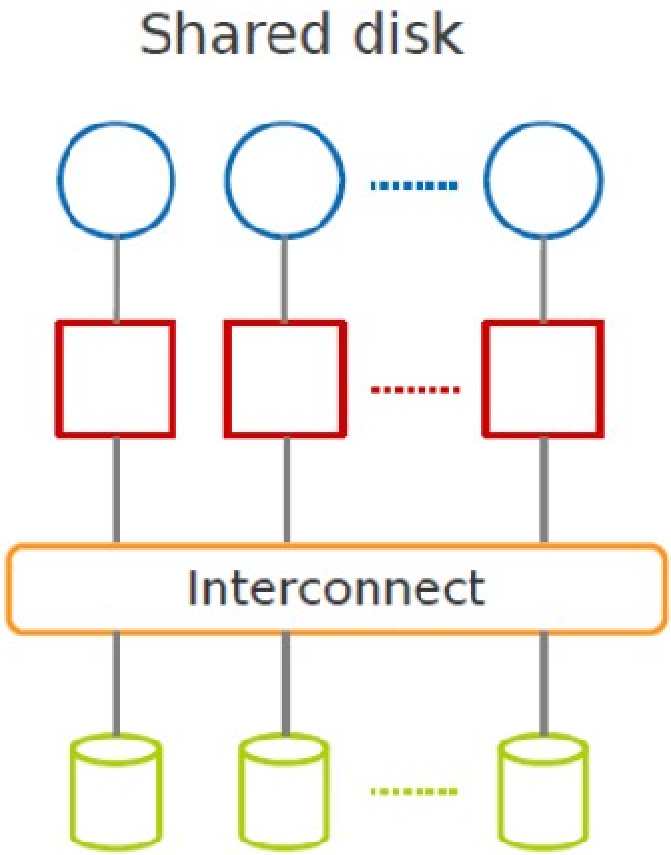
Shared Disk

- Disk is shared, memory is private
  - Storage Area Network (SAN) to interconnect memory and disk (block level)
  - Needs distributed lock manager (DLM) for cache coherence

+ Simple for apps, extensibility

- Complex DLM, cost

- For write-intensive workloads or big data

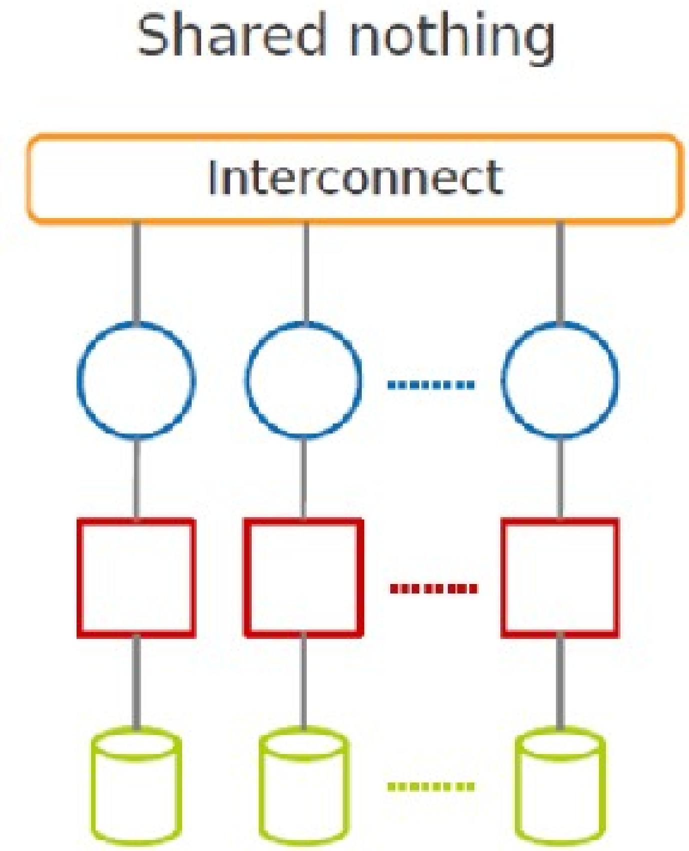


Shared Nothing

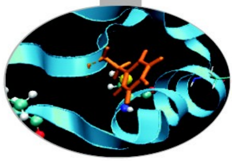
- No sharing of memory or disk across nodes
  - No need for DLM
  - But needs data partitioning

  + highest extensibility, cost

  - updates, distributed trans
- For **big data** (read intensive)



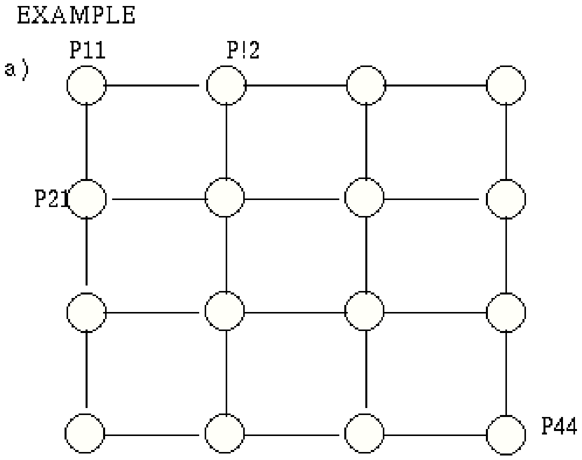
Example networks



Some variations of the mesh model have wrap-around type connections between the nodes to the edges of the mesh (torus topology).

The Cray T3E adopts a 3D torus topology

IBM BG/Q adopts a 5D torus topology



2D mesh of width 4 with

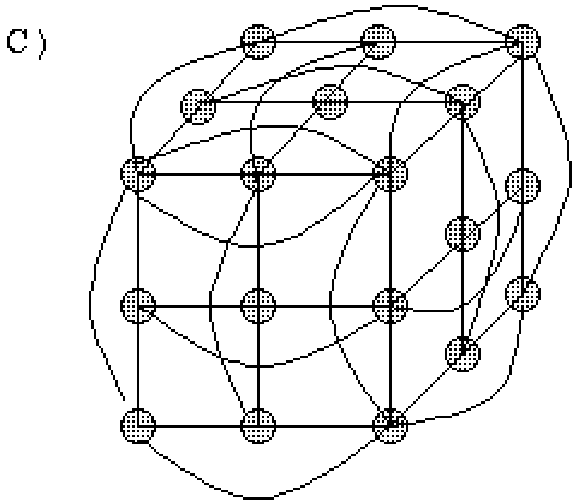
no wraparound connections

on edge or corner nodes

corner nodes have degree 2

edge nodes have degree 3

MESH Topology



k = 3 w = 3

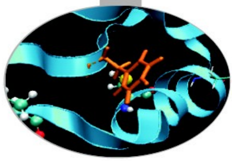
i.e. 3 ^ 3 = 27 nodes

with wraparound connections

all nodes have degree 6 (2k)

Toroidal Topology

Commodity Interconnects



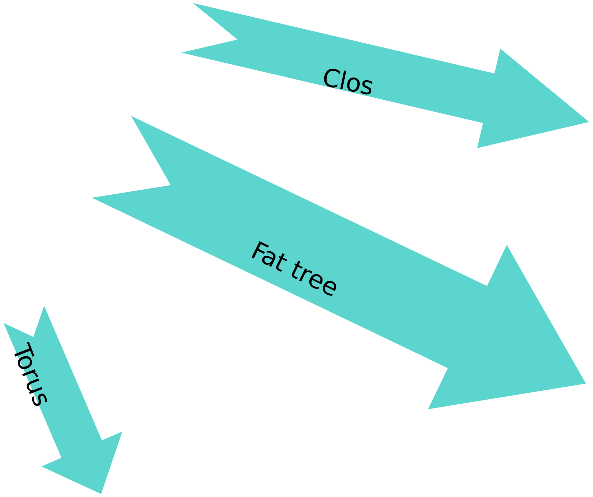
Gig Ethernet

Myrinet

Infiniband

QsNet

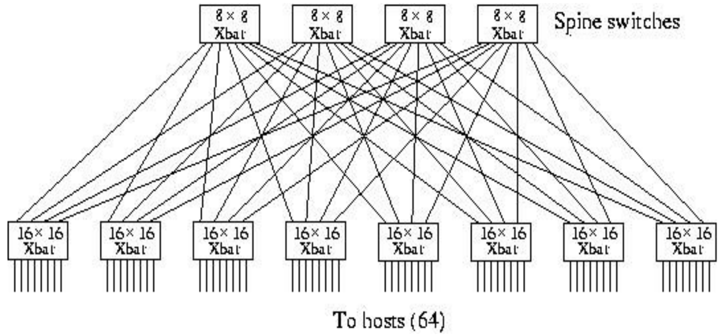
SCI

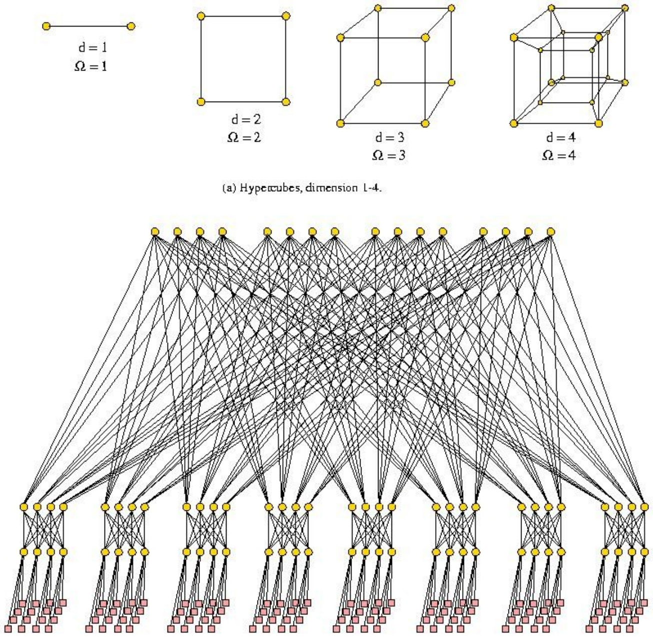


Clos

Fat tree

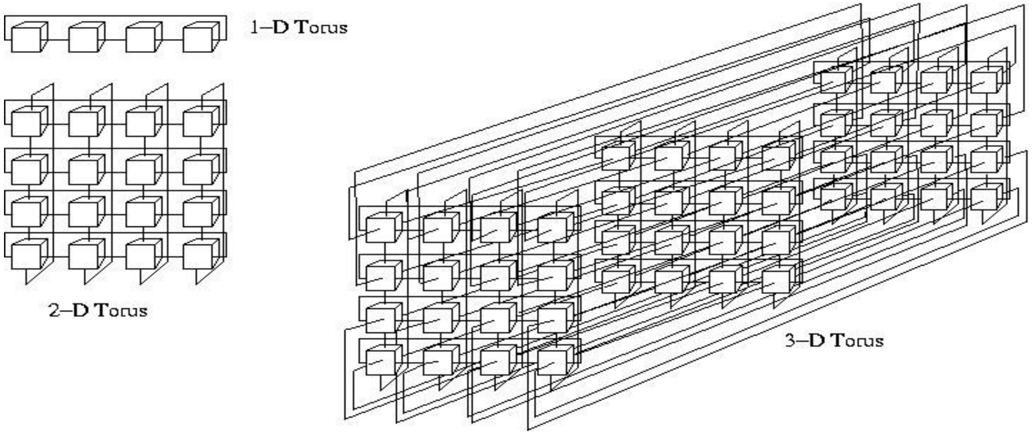
Torus



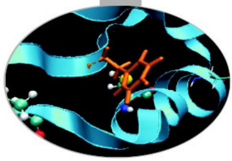


(a) Hypercubes, dimension 1-4.

(b) A 128-way fat tree.



Simple Model for Parallel Data



Shared-nothing architecture

- The most general and scalable

Set-oriented

- Each dataset **D** is represented by a table of rows

Key-value

- Each row is represented by a <key, value> pair where
  - *Key uniquely identifies the value in D*
  - *Value is a list of (attribute name : attribute value)*

Can represent structured (relational) data or NoSQL data

- But graph is another story (see Pregel, DEX or Spark)

Examples

- *<row-id#5, (part-id:5, part-name:iphone5, supplier:Apple)>*
- *<doc-id#10, (content:<html> html text … </html>)>*
- *<akeyword, (doc-id:id1, doc-id:id2, doc-id:id10)>*

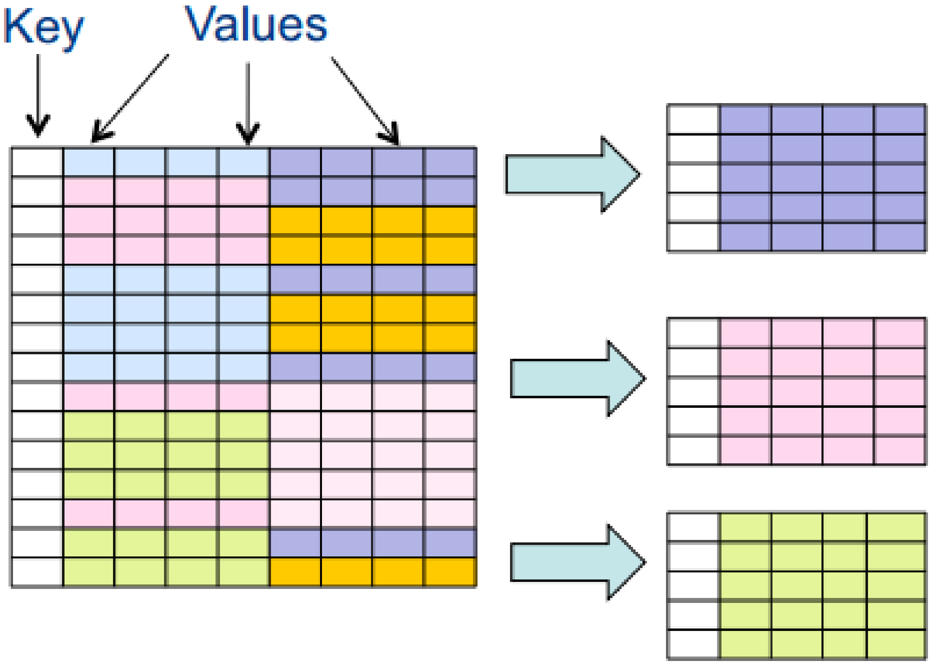
Data Partitioning

Vertical partitioning

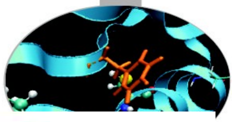
- Basis for column stores (e.g. MonetDB): efficient for OLAP queries
- Easy to compress, e.g. using Bloom filters

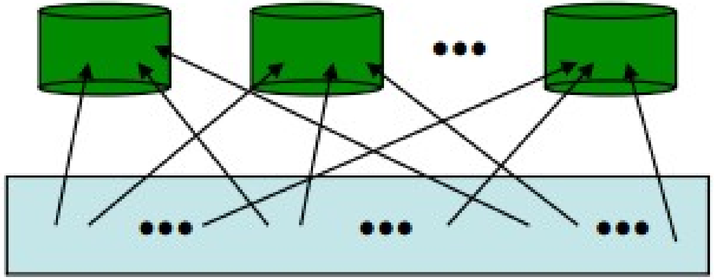
Horizontal partitioning (sharding)

- Shards can be stored (and replicated) at different nodes



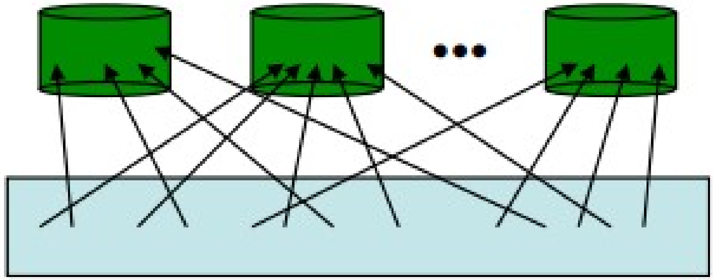
Sharding Schemes





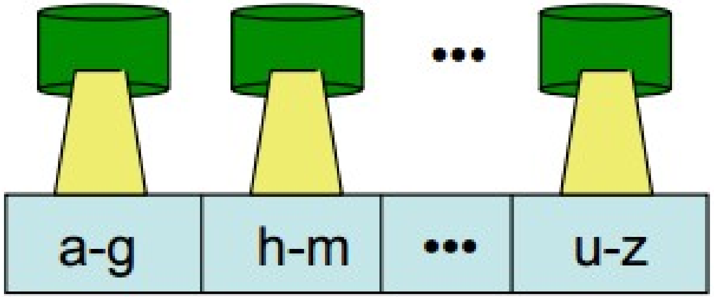
Round-Robin

- *i*th row to node (*i mod n*)
- perfect balancing
- but full scan only



Hashing

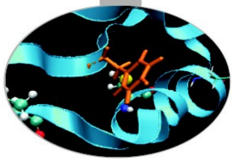
- (k,v) to node h(k)
- exact-match queries
- but problem with skew



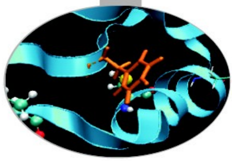
Range

- (k,v) to node that holds k's interval
- exact-match and range queries
- deals with skew

Different classes of applications

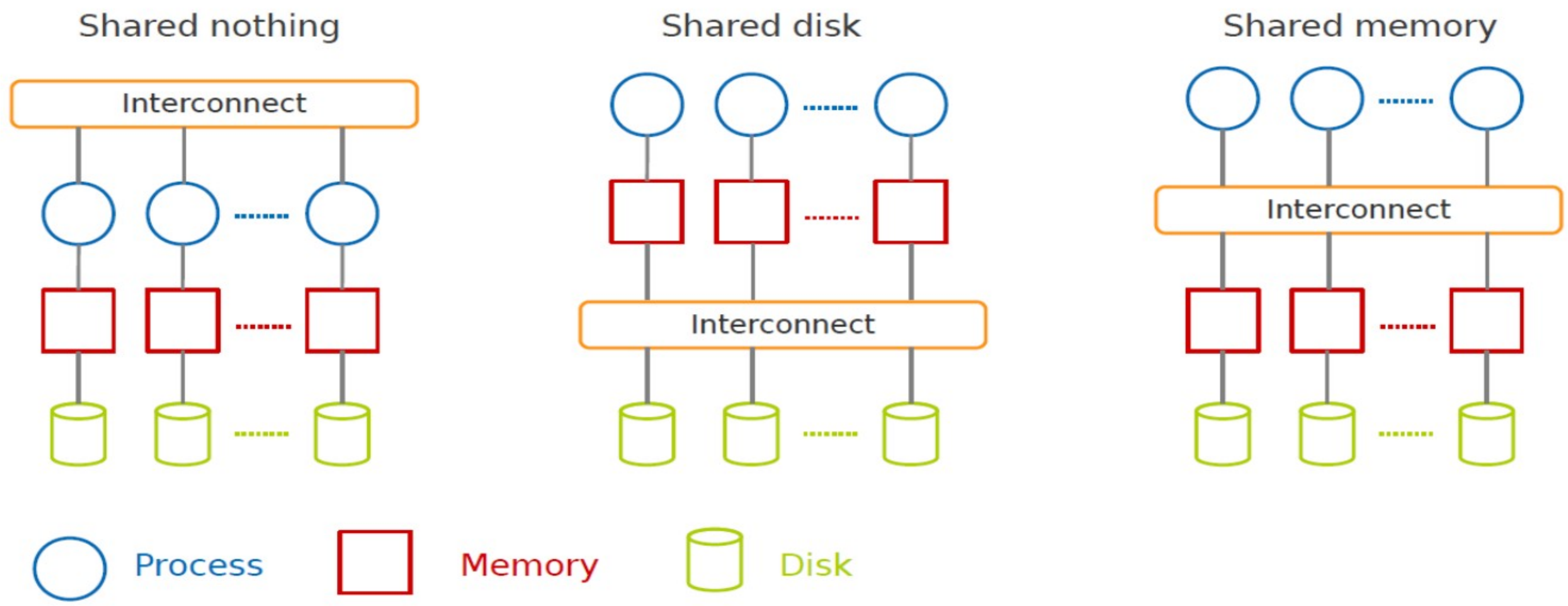


- **MPI (Message Passing Interface)**
  - A shared disk infrastructure for processing large data sets with a parallel algorithm on clusters
- **OpenMP (Open MultiProcessing)**
  - A shared memory infrastructure for processing large data sets with a parallel algorithm on a node
- **Map Reduce (Hadoop)**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

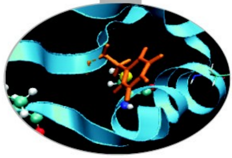


Parallel Architectures

Map Reduce/Hadoop **MPI** **OpenMP**



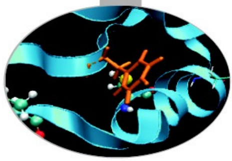
Programming Models: What is MPI?



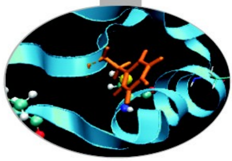
- **Message Passing Interface (MPI)**
  - World's most popular distributed API
  - MPI is "de facto standard" in scientific computing
  - C and FORTRAN, ver. 2 in 1997
- **What is MPI good for?**
  - Abstracts away common network communications
  - Allows lots of control without bookkeeping
  - Freedom 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

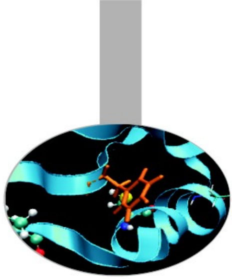


- **Deadlock is possible…**
  - Blocking communication can cause deadlock
    - "crossed" calls when trading information
    - example:
    - `Proc1: MPI_Receive(Proc2, A); MPI_Send(Proc2, B);`
    - `Proc2: MPI_Receive(Proc1, B); MPI_Send(Proc1, A);`
    - There are some solutions - MPI_SendRecv()
- **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!**



Brief recap

Map Reduce flow



- Programmers must specify:

  map (k, v) → list(<k’, v’>)

  reduce (k’, list(v’)) → <k’’, v’’>

  - All values with the same key are reduced together

- Optionally, also:

  partition (k’, number of partitions) → partition for k’

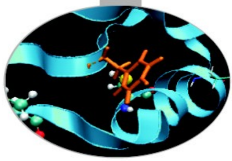
  - Often a simple hash of the key, e.g., hash(k’) mod n
  - Divides up key space for parallel reduce operations

  combine (k’, v’) → <k’, v’>*

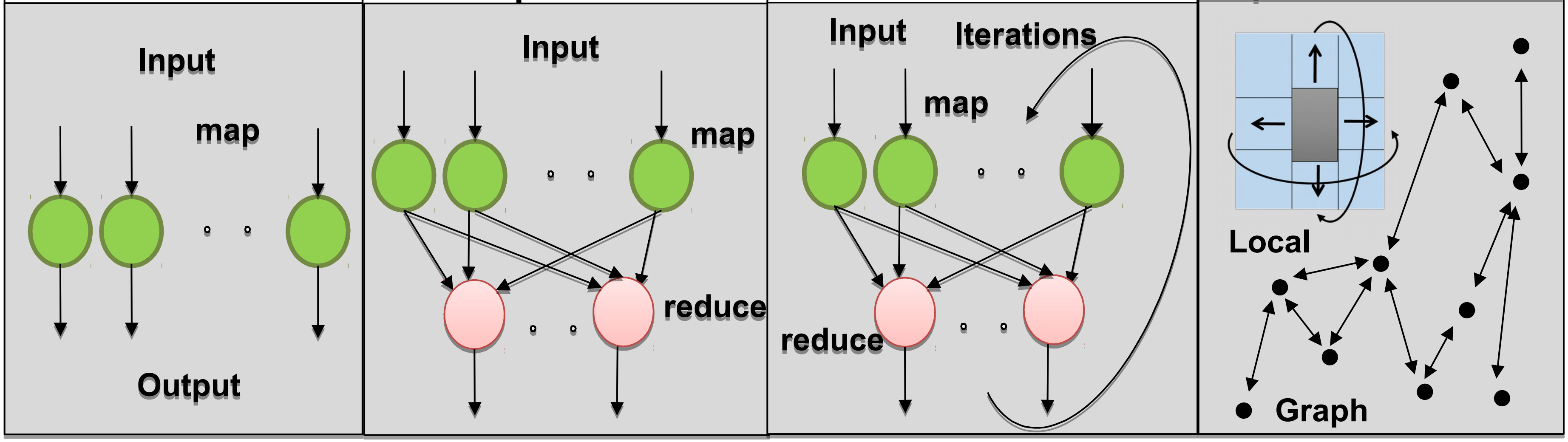
  - Mini-reducers that run in memory after the map phase
  - Used as an optimization to reduce network traffic

- The execution framework handles everything else…

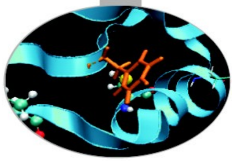
“Everything Else”



- The execution framework handles everything else…
  - **Scheduling**: assigns workers to map and reduce tasks
  - **“Data distribution”**: moves processes to data
  - **Synchronization**: gathers, sorts, and shuffles intermediate data
  - **Errors and faults**: detects worker failures and restarts
- Limited control over data and execution flow
  - All algorithms must expressed in m, r, c, p
- You don’t know:
  - Where mappers and reducers run
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which intermediate key a particular reducer is processing

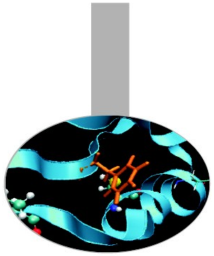


(1) Map Only	(2) Classic MapReduce	(3) Iterative Map Reduce or Map-Collective	(4) Point to Point or Map-Communication
BLAST Analysis Local Machine Learning Pleasingly Parallel	High Energy Physics (HEP) Histograms Distributed search Recommender Engines	Expectation maximization Clustering e.g. K-means 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			



Are emerging data analytics techniques the new El Dorado?

Where and When using Apache Hadoop



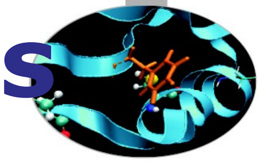
Where

- 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

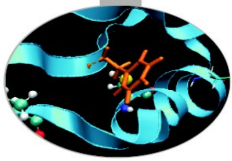
Advantages/Disadvantages



- **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

Parallel Computing Model



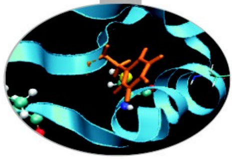
MapReduce can be classified as a SIMD (single-instruction, multiple-data) problem.

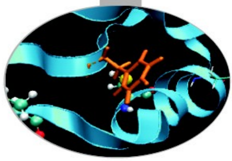
- ✔ Indeed, the map step is highly scalable because the same instructions are carried out over all data. Parallelism arises by breaking the data into independent parts with no forward or backward dependencies (side effects) within a Map step; that is, the Map step may not change any data (even its own).
- ✔ 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.

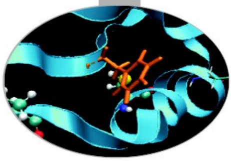
A Tale of Two Data-Intensive Paradigs: Applications, Abstractions, and Architectures
Shantenu Jha , Judy Qiu, Andre Luckow , Pradeep Mantha , Geoffrey C.Fox





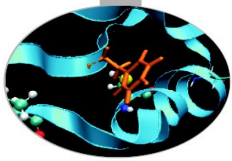
When to use Apache Hadoop

When to use Apache Hadoop



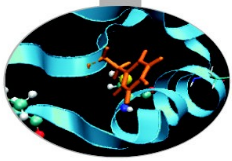
- **Your Data Sets Are Really Big**
  - *Don’t even think about Hadoop if the data you want to process is measured in MBs or GBs.* If the data driving the main problem you are hoping to use Hadoop to solve is measured in GBs, save yourself the hassle and use Excel, a SQL BI tool on Postgres, or some 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 MapReduce



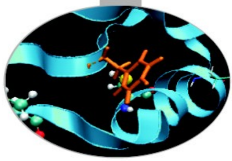
- **You Find Yourself Throwing Away Perfectly Good Data**
  - One of the great things about Hadoop is its capability to store petabytes of data. If you find that you are throwing away potentially valuable data because its costs too much to archive, you may find that setting up a Hadoop cluster allows 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 MapReduce



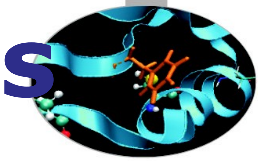
- **You Need Answers in a Hurry**
  - Hadoop is probably not the ideal solution if you need really fast access to data. The various SQL engines for Hadoop have made big strides in the past year, and will likely continue to improve. But if you’re using Map-Reduce to crunch your data, expect to wait days 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 MapReduce

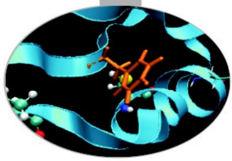


- **You Require Random, Interactive Access to Data**
  - The pushback from the limitations of the batch-oriented MapReduce paradigm in early Hadoop led the community to improve SQL performance and boost its capability to serve interactive queries against random data. While SQL on Hadoop is getting better, 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

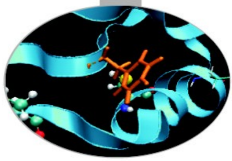


- **Now it’s easy to program for many CPUs**
  - Communication management effectively gone
    - I/O scheduling done for us
  - Fault tolerance, monitoring
    - machine failures, suddenly-slow machines, etc are handled
  - Can be much easier to design and program!
  - Can cascade several (many?) Map-Reduce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



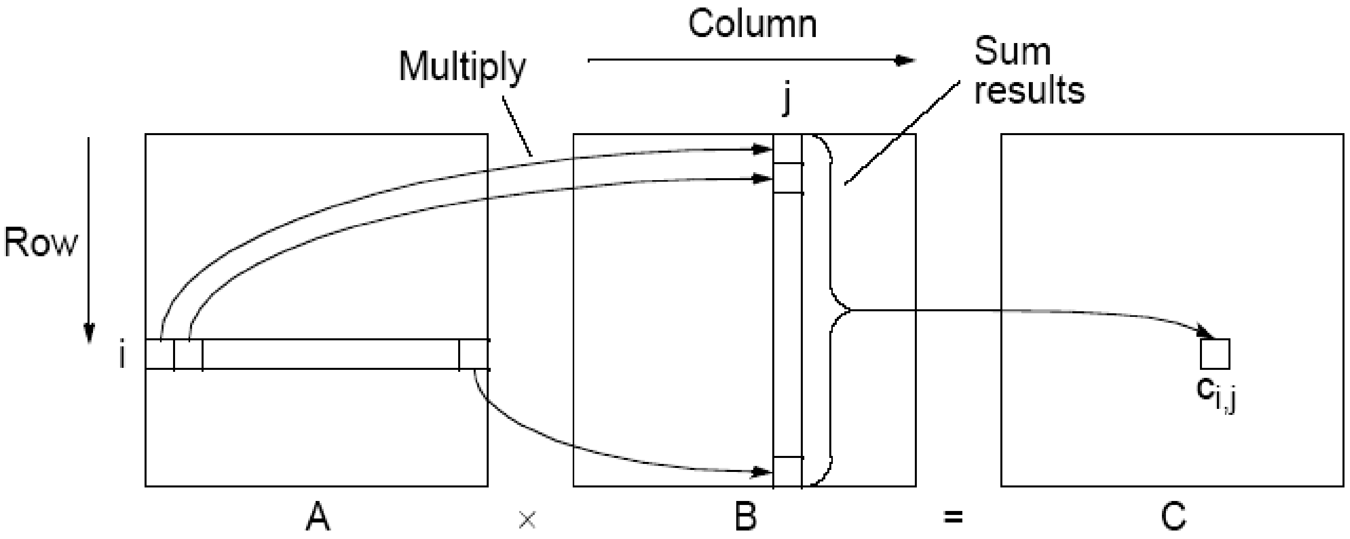
Matrix-matrix product v1

Matrix-matrix product



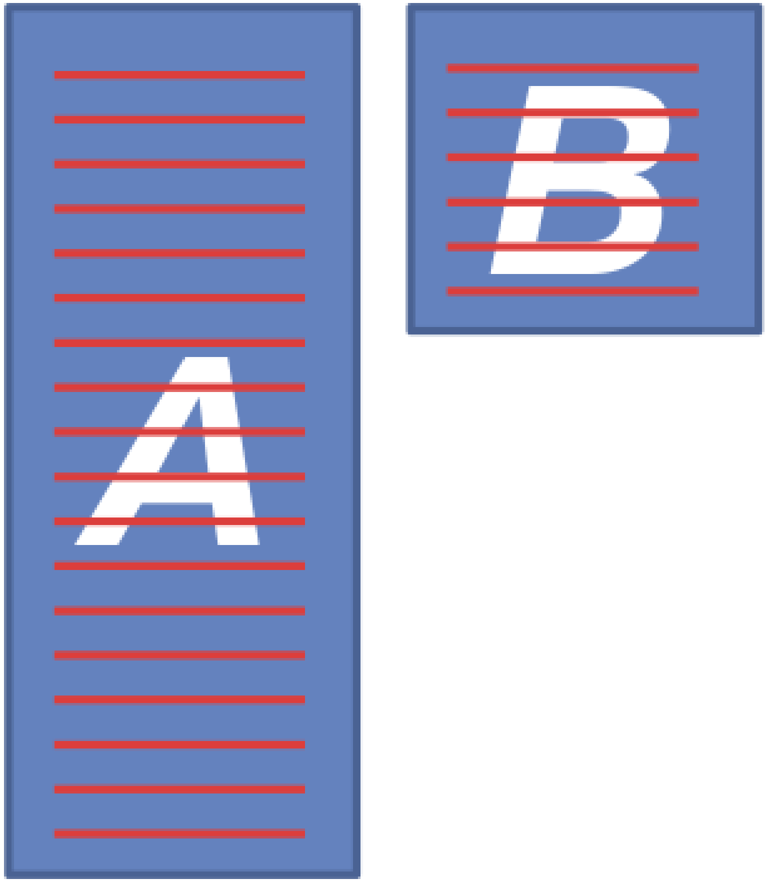
- Basic matrix multiplication on a 2-D grid
- Matrix multiplication is an important application in HPC and appears in many areas (linear algebra)
- **C = A * B** where A, B, and C are matrices (two-dimensional arrays)
- A restricted case is when B has only one column, matrix-vector product, which appears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

C = A x B



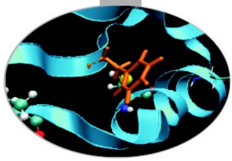
$$c_{i,\,j} = \sum_{k=0}^{l-1} a_{i,k} b_{k,j}$$

Matrix-matrix product



$$\mathbf{AB} = \mathbf{C}$$

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



Matrix-matrix product

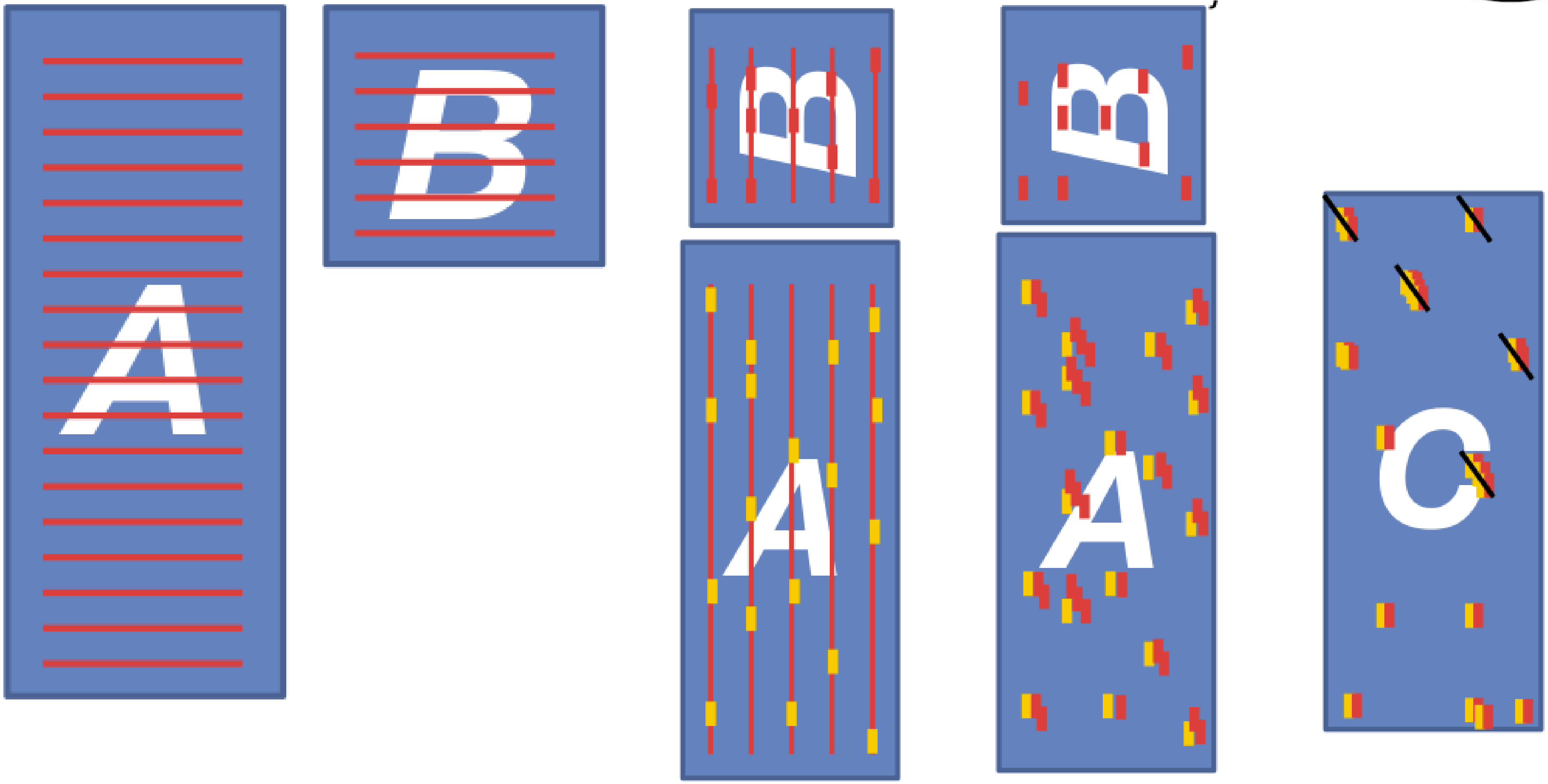
A is stored by row (`$ head data/mat/smat_10x5_A`)

```
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



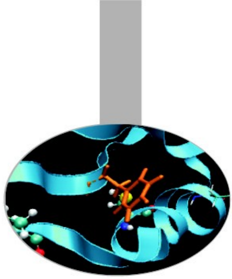
$$\boldsymbol{AB} = \boldsymbol{C}$$

$$C_{ij} = \sum_{k} A_{ik} B_{kj}$$

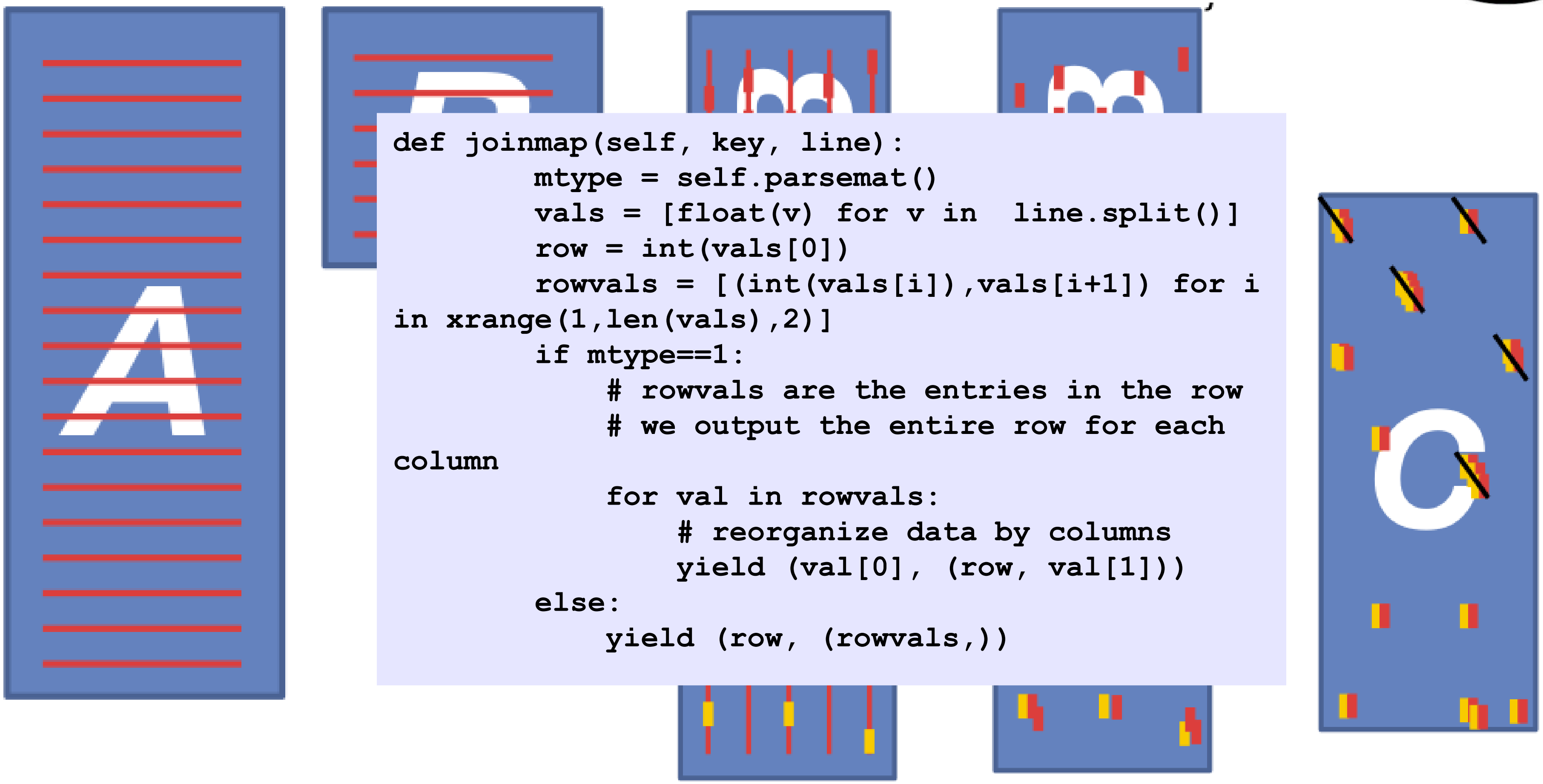
Map 1
Align on
columns

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

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



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,))
```

$$\boldsymbol{AB = C}$$

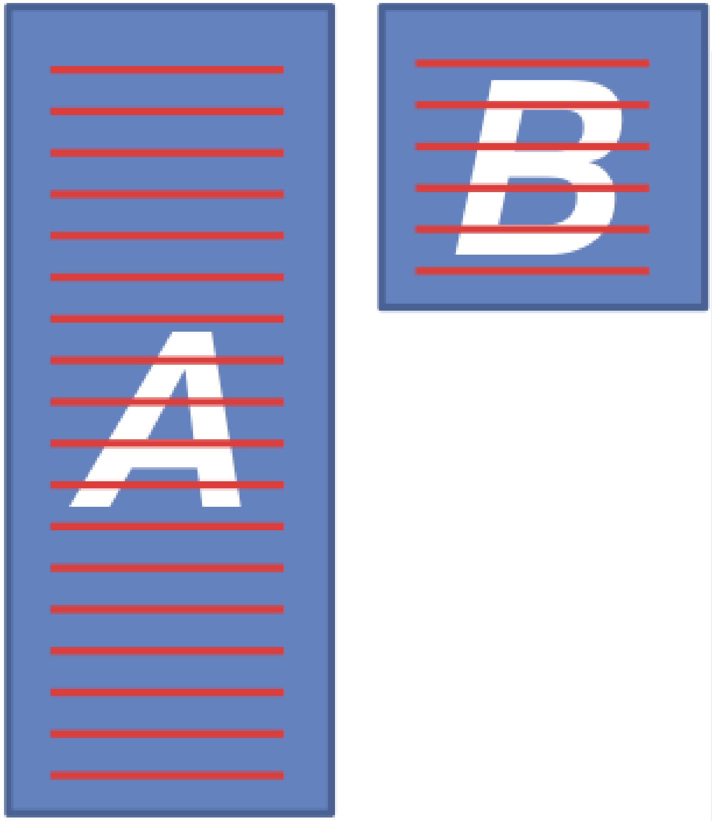
$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Map 1
Align on columns

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

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

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)
```

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

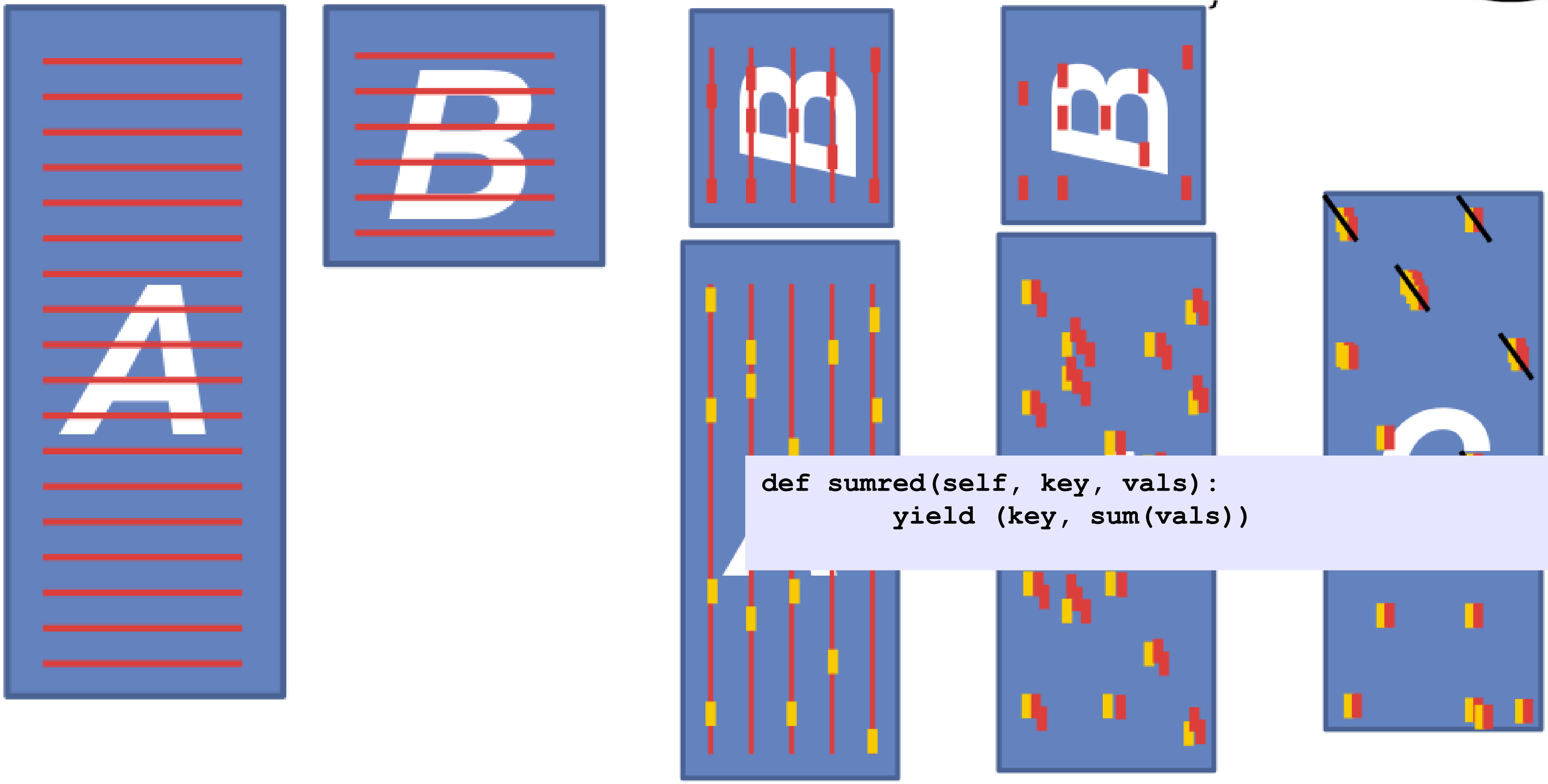
$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Map 1
Align on columns

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

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

Sumred



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

Map 1
Align on columns

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

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

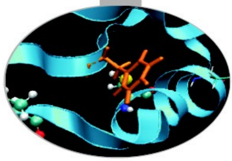
$$\boldsymbol{AB} = \boldsymbol{C}$$

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

sparse_matmat.py

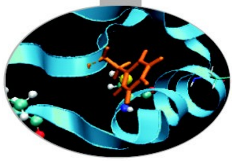
```
from mrjob.job import MRJob
from mrjob.compat import get_jobconf_value
import itertools
import sys

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()
```



Matrix-matrix product v2

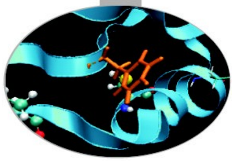
Matrix-matrix product v2



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

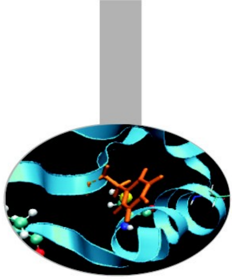
The Reduce Function:

- 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

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()
```

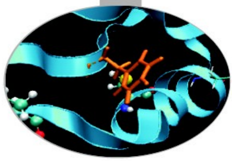


Matrix-matrix product v2

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
```

Log based debug

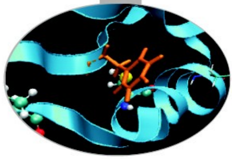


- Python

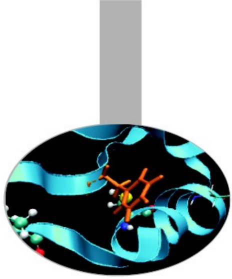
```
sys.stderr(out).write("REDUCER INPUT: ({0},{1})\n".format(j,
values))
```

- Java

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



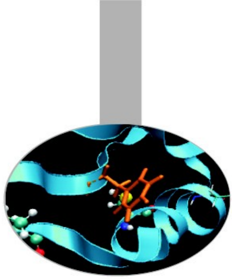
Debugging



Debug mechanisms

- The Web Interface
- Runtime monitor
- Log based debug

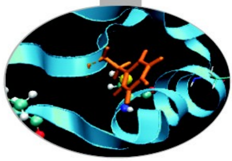
The Web User Interface

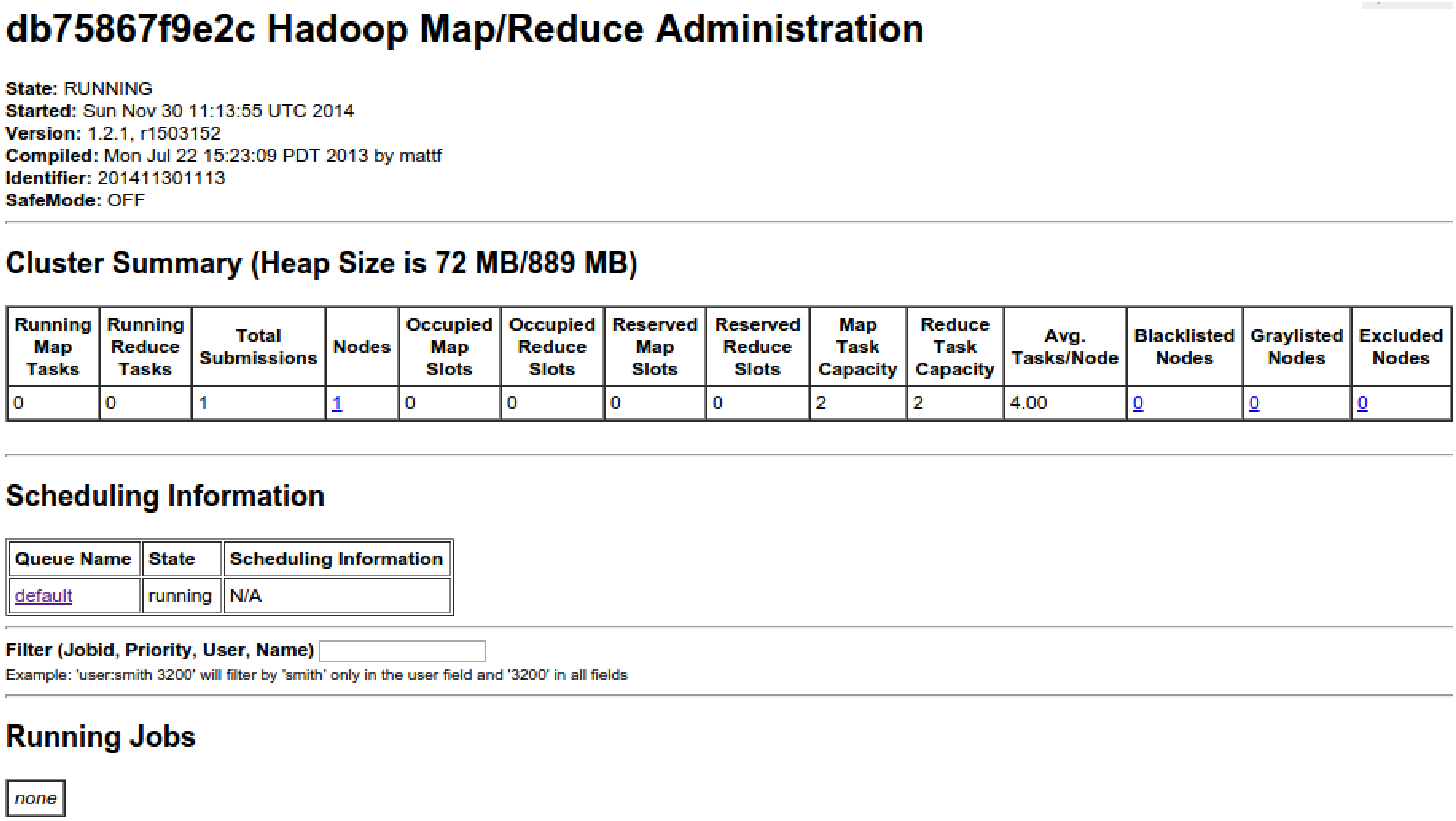


- **Hadoop** comes with a web UI for viewing information about your jobs. It is useful for following a job’s progress while it is running, as well as finding job statistics and logs after the job has completed.

- 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

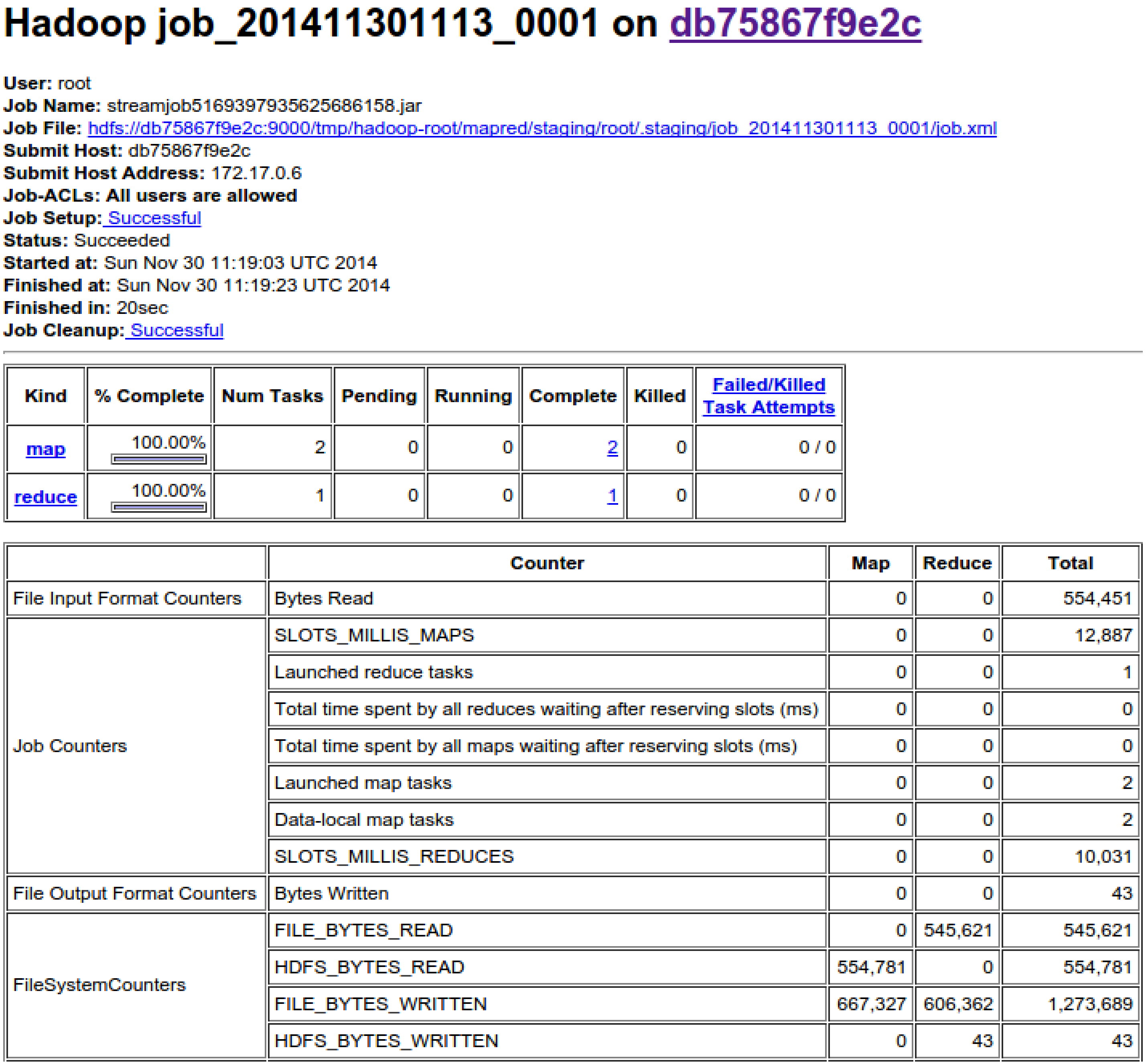
Queue Name	State	Scheduling Information
default	running	N/A

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

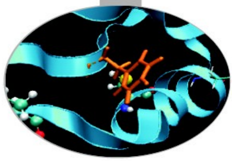


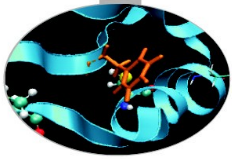
Hadoop job_201411301113_0001 on db75867f9e2c

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



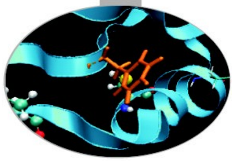


Hadoop Reporter

- 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.

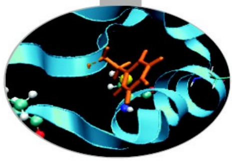


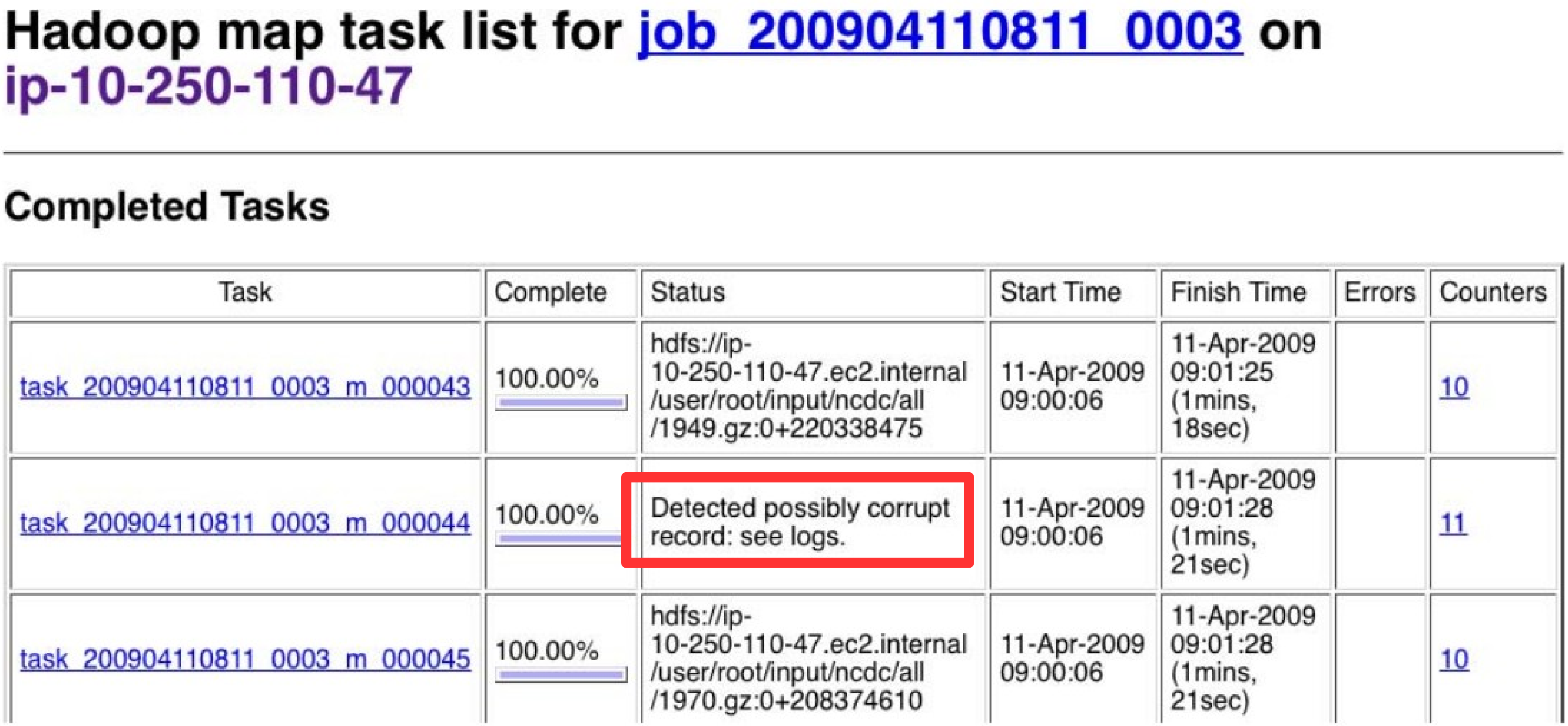
Hadoop Reporter

“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



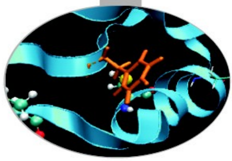


Hadoop map task list for job_200904110811_0003 on ip-10-250-110-47

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

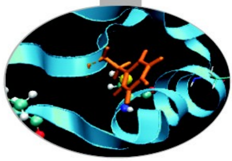
Runtime monitor



- 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

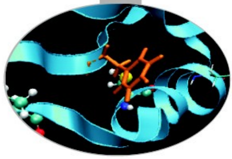
- **Python**

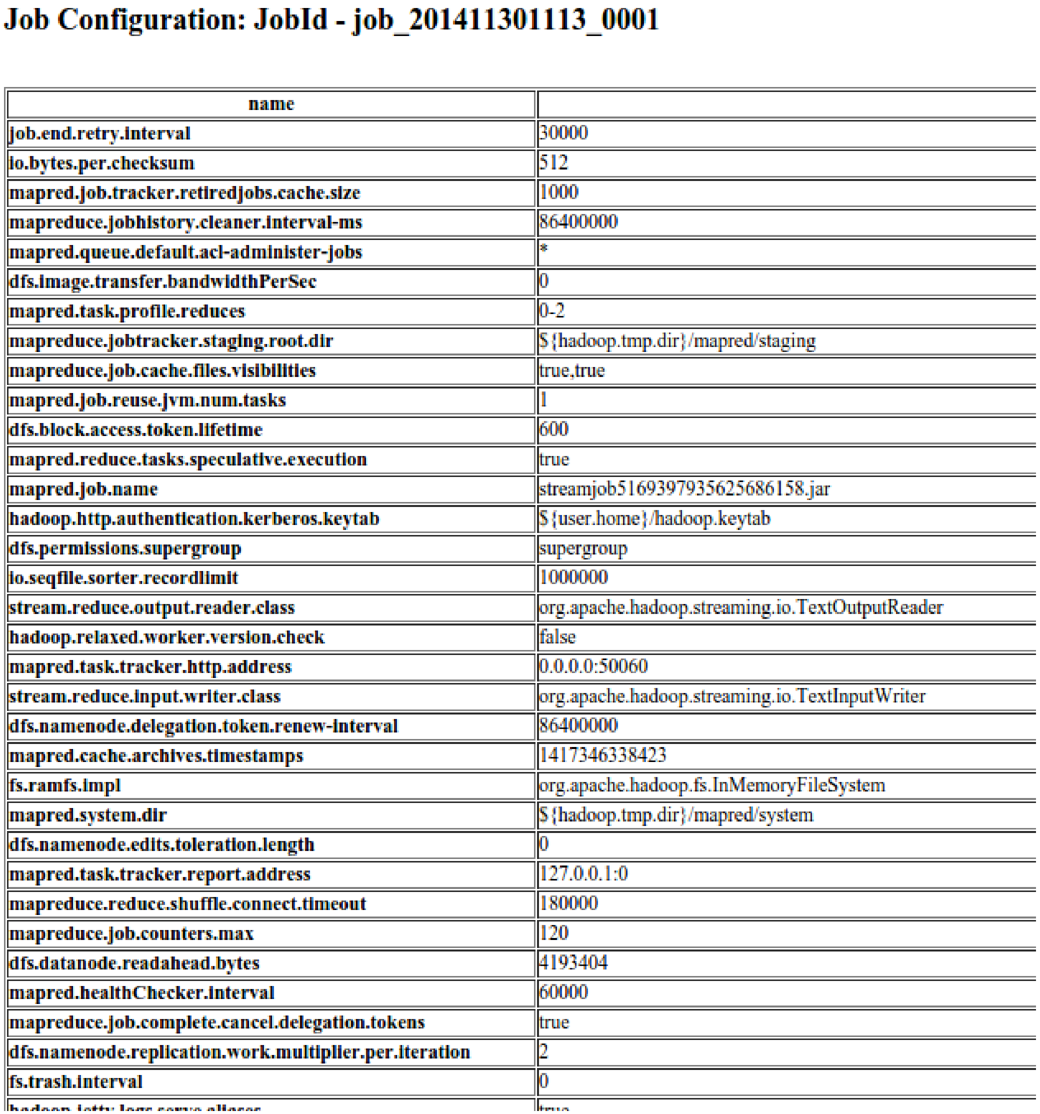
```
sys.stderr(out).write("REDUCER INPUT: ({0},{1})\n".format(j,
values))
```

- **Java**

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

Debugging/profiling

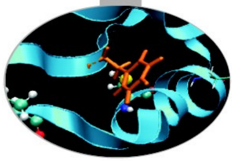




Job Configuration: JobId - job_201411301113_0001

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

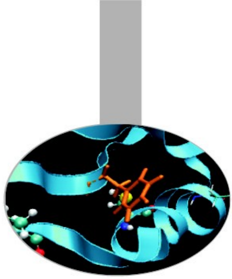
Profiling



- 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.

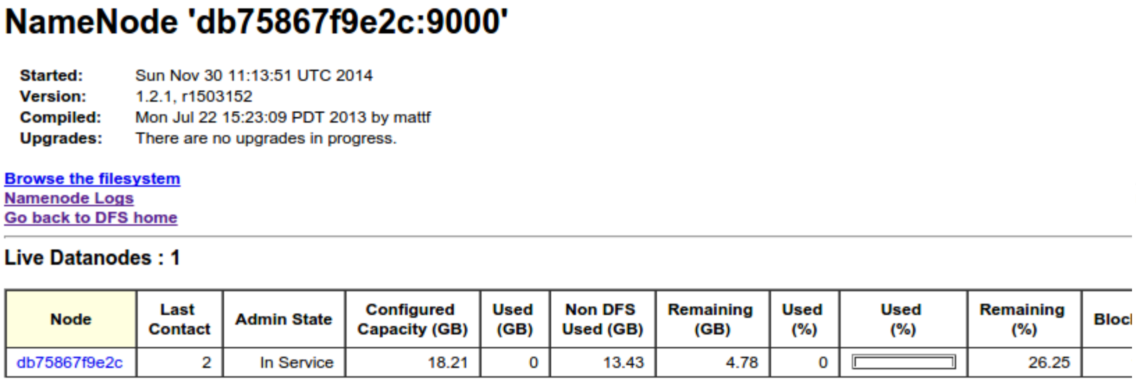
```
conf.setProfileEnabled(true);
conf.setProfileParams("-
agentlib:hprof=cpu=samples,heap=sites,depth=6," +
"force=n,thread=y,verbose=n,file=%s");
conf.setProfileTaskRange(true, "0-2");
```

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



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

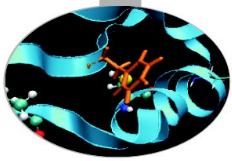
Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Bloc
db75867f9e2c	2	In Service	18.21	0	13.43	4.78	0		26.25	

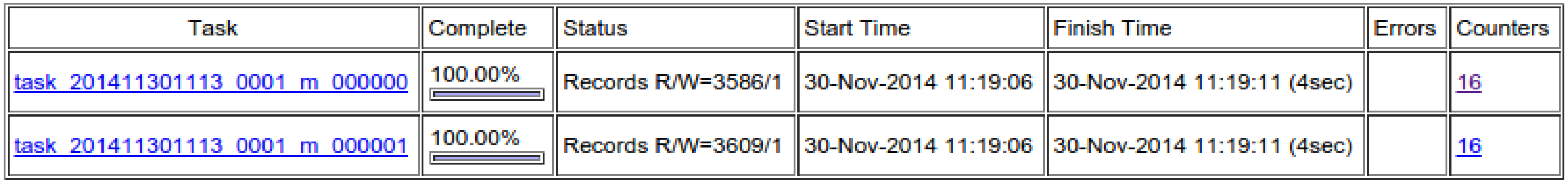
Map Completion Graph - close

100
90
80
70
60
50
40
30
20
10
0
0 1 2

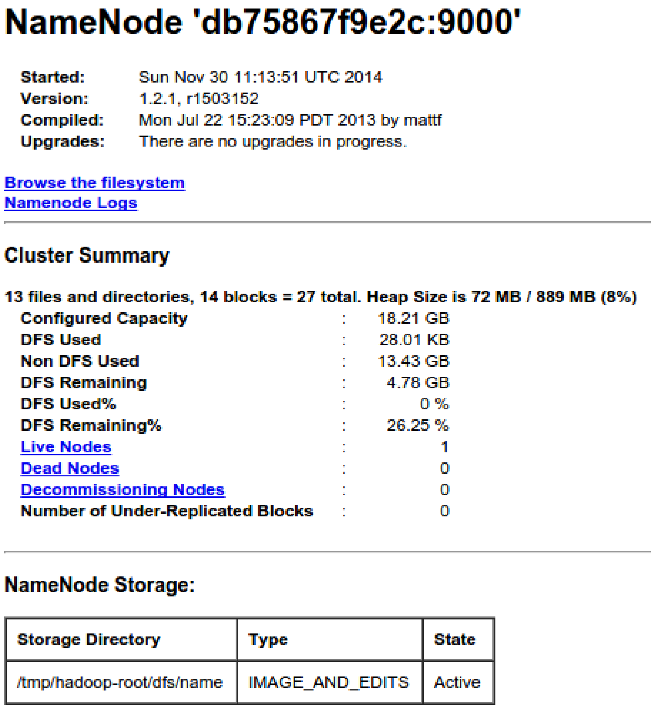
Reduce Completion Graph - close

100
90
80
70
60
50
40
30
20
10
0
0 1
copy
sort
reduce





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

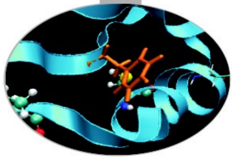
13 files and directories, 14 blocks = 27 total. Heap Size is 72 MB / 889 MB (8%)

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

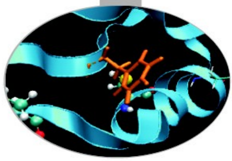
Cluster optimizations



The problem:

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

Hadoop basic options

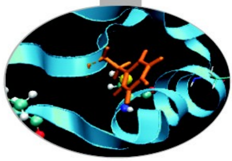


All hadoop commands are invoked by the bin/hadoop script. Running the hadoop script without any arguments prints the description for all commands.

```
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.

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.

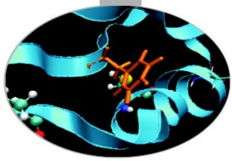
-jt <local|jobtracker:port>   Specify a job tracker. Applies only to job.

-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

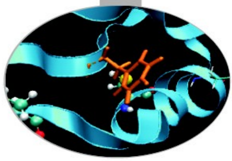


Compression *mapred.compress.map.output* → Map Output Compression

- **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

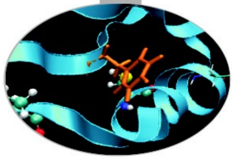


Speculative Execution *mapred.map/reduce.speculative.execution*
→ Enable/Disable task (map/reduce) speculative Execution

- **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.

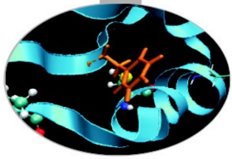
```
$ bin/hadoop jar -Dmapred.map.tasks.speculative.execution=false \
                 -Dmapred.reduce.tasks.speculative.execution=false
```

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

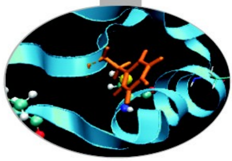


Number of Maps/Reducers

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).

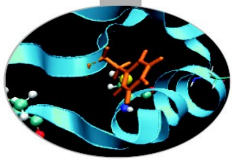
File block size



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.

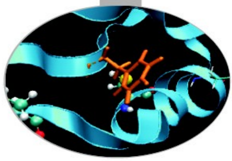
Sort size



Sort size *io.sort.mb →* Buffer size (MBs) for sorting

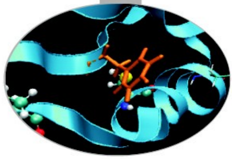
- **Default**: 100
- **Suggestions**: For Large jobs (the jobs in which map output is very large), this value should be increased keeping in mind that it will increase the memory required by each map task. So the increment in this value should be according to the available memory at the node. Greater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

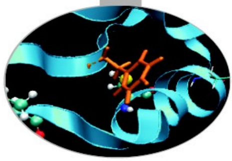


JVM Reuse

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.

CINECA SCAI

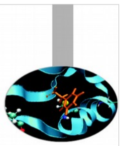
SuperComputing Applications and Innovation

Welcome!

Tool and techniques for massive data processing

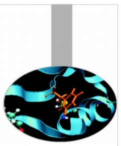
Giuseppe Fiameni
April 12th 2016

CINECA



Why this course?

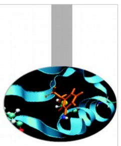
- Data are becoming more and more important
- Processing large data sets has become an issue for young researchers
- Many interesting technologies are emerging and entering the HPC domain
- HPC common technologies, although are the only viable solutions in many cases, have a steep learning curve which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



Goals

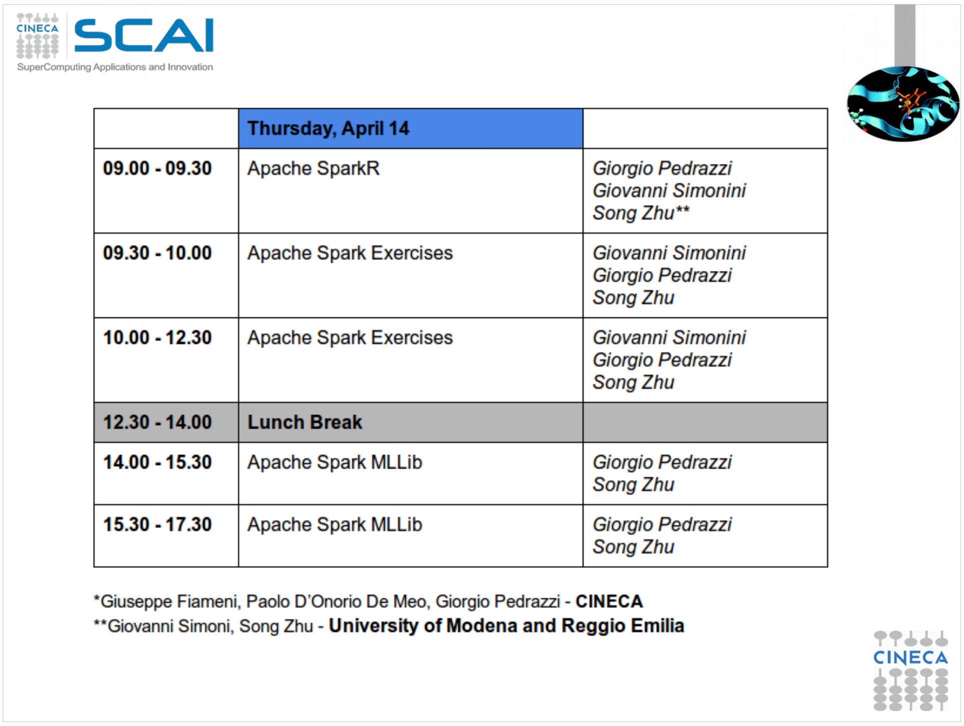
During this course, you will learn:

- the trends and challenges surrounding the BigData definition
- how the most relevant technologies and methods in this domain work
  - *Map-Reduce*
  - *Apache Hadoop*
  - *Apache Spark, Apache SparkSQL*
  - *Apache Spark MLLIB (Machine Learning Library)*
- how to structure and program your code using Python/Scala



General notes

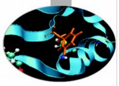
- Each participant will receive personal credentials to access Cineca resources for a limited amount of time
  - requests for extensions are possible
- **There is one workstation every two participants**:
  - **U**: corsi **P**: corsi_2013!
- **Course web site**:
  - https://goo.gl/Pq03qz
- Feel free to ask questions at any time → do not be shy!

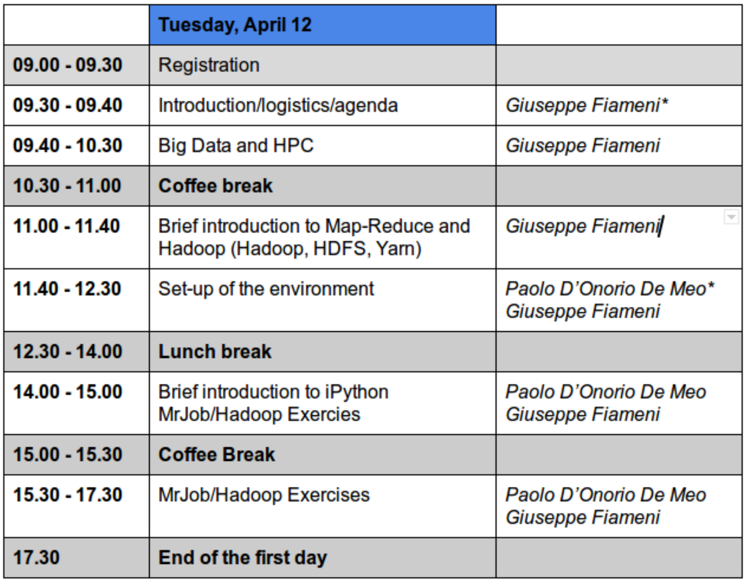


	Thursday, April 14	
09.00 - 09.30	Apache SparkR	*Giorgio Pedrazzi* *Giovanni Simonini* *Song Zhu***
09.30 - 10.00	Apache Spark Exercises	*Giovanni Simonini* *Giorgio Pedrazzi* *Song Zhu*
10.00 - 12.30	Apache Spark Exercises	*Giovanni Simonini* *Giorgio Pedrazzi* *Song Zhu*
12.30 - 14.00	**Lunch Break**	
14.00 - 15.30	Apache Spark MLLib	*Giorgio Pedrazzi* *Song Zhu*
15.30 - 17.30	Apache Spark MLLib	*Giorgio Pedrazzi* *Song Zhu*

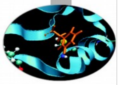
*Giuseppe Fiameni, Paolo D'Onorio De Meo, Giorgio Pedrazzi - **CINECA**

Giovanni Simoni, Song Zhu - **University of Modena and Reggio Emilia



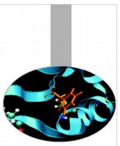


	Tuesday, April 12	
09.00 - 09.30	Registration	
09.30 - 09.40	Introduction/logistics/agenda	*Giuseppe Fiameni**
09.40 - 10.30	Big Data and HPC	*Giuseppe Fiameni*
10.30 - 11.00	**Coffee break**	
11.00 - 11.40	Brief introduction to Map-Reduce and Hadoop (Hadoop, HDFS, Yarn)	*Giuseppe Fiameni*
11.40 - 12.30	Set-up of the environment	*Paolo D'Onorio De Meo** *Giuseppe Fiameni*
12.30 - 14.00	**Lunch break**	
14.00 - 15.00	Brief introduction to iPython MrJob/Hadoop Exercies	*Paolo D'Onorio De Meo* *Giuseppe Fiameni*
15.00 - 15.30	**Coffee Break**	
15.30 - 17.30	MrJob/Hadoop Exercises	*Paolo D'Onorio De Meo* *Giuseppe Fiameni*
17.30	**End of the first day**	



	Wednesday, April 13	
09.30 - 11.00	MrJob/Hadoop Exercises	*Paolo D'Onorio De Meo* *Giuseppe Fiameni*
11.00 - 11.15	**Coffee break**	
11.15 - 12.30	MrJob/Hadoop Exercises (NGS Use Case)	*Paolo D'Onorio De Meo* *Giuseppe Fiameni*
12.30 - 14.00	**Lunch break**	
14.00 - 15.00	Apache Spark	*Giovanni Simonini*** *Giorgio Pedrazzi**
15.00 - 16.00	Apache Spark Examples	*Giovanni Simonini* *Giorgio Pedrazzi*
16.00 - 16.30	**Coffee break**	
16.30 - 17.30	Apache SparkSQL	*Giovanni Simonini* *Giorgio Pedrazzi*
17.30	**End of the second day**	

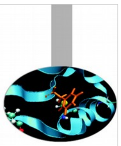
SuperComputing Applications and Innovation



	Thursday, April 14	
09.00 - 09.30	Apache SparkR	*Giorgio Pedrazzi* *Giovanni Simonini* *Song Zhu***
09.30 - 10.00	Apache Spark Exercises	*Giovanni Simonini* *Giorgio Pedrazzi* *Song Zhu*
10.00 - 12.30	Apache Spark Exercises	*Giovanni Simonini* *Giorgio Pedrazzi* *Song Zhu*
12.30 - 14.00	**Lunch Break**	
14.00 - 15.30	Apache Spark MLLib	*Giorgio Pedrazzi* *Song Zhu*
15.30 - 17.30	Apache Spark MLLib	*Giorgio Pedrazzi* *Song Zhu*

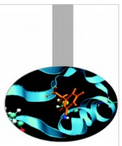
*Giuseppe Fiameni, Paolo D'Onorio De Meo, Giorgio Pedrazzi - **CINECA**

Giovanni Simoni, Song Zhu - **University of Modena and Reggio Emilia



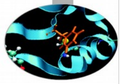
Quick poll

- How many of you already know part of the mentioned Big Data technologies?
- How many of you know any HPC methods (MPI, OpenMP, etc.)?
- How many of you know Python?
- Scala?



Before starting

Size of computational applications



Computational Dimension:

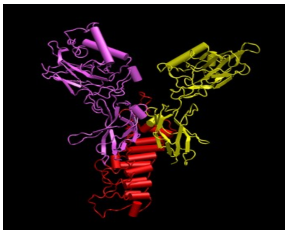
number of operations needed to solve the problem, in general is a function of the size of the involved data structures (n, n^2 , n^3 , n log n, etc.)

flop - Floating point operations

indicates an arithmetic floating point operation.

flop/s - Floating points operations per second

is a unit to measure the speed of a computer.

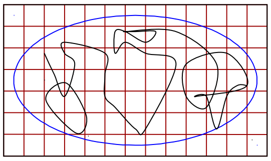


Computational problems today: 10^{15} – 10^{22} flop

One year has about 3 x 10^7 seconds!

Most powerful computers today have reach a sustained performance is of the order of Tflop/s - Pflop/s (10^{12} - 10^{15} flop/s).

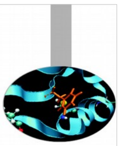
Example: Weather Prediction



Forecasts on a global scale:

- **3D Grid to represent the Earth**
  - Earth's circumference: ≅ 40000 km
  - radius: ≅ 6370 km
  - Earth's surface: ≅ $4\pi r^2$ ≅ $5 \cdot 10^8$ km^2
- **6 variables:**
  - temperature
  - pressure
  - humidity
  - wind speed in the 3 Cartesian directions
- cells of 1 km on each side
- 100 slices to see how the variables evolve on the different levels of the atmosphere
- a 30 seconds time step is required for the simulation with such resolution
- Each cell requires about 1000 operations per time step (Navier-Stokes turbulence and various phenomena)

Example: Weather Prediction / 1



Grid: $5 \bullet 10^{8} \bullet 100 = 5 \bullet 10^{10}$ cells

- each cell is represented with 8 Byte
- Memory space:
  - → (6 var)•(8 Byte)•($5 \bullet 10^{10}$ cells) $\cong 2 \bullet 10^{12}$ Byte = 2TB

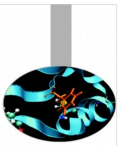
A 24 hours forecast needs:

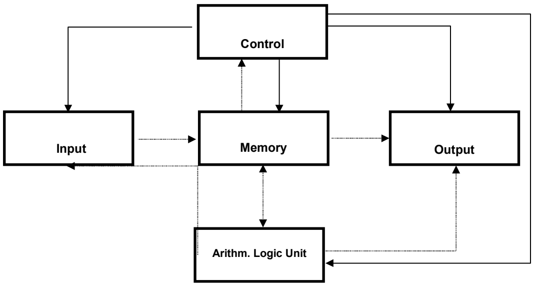
- → $24 \bullet 60 \bullet 2 \cong 3 \bullet 10^{3}$ time-step
- → ($5 \bullet 10^{10}$ cells) • (10^{3} oper.) • ($3 \bullet 10^{3}$ time-steps) = $1.5 \bullet 10^{17}$ operations !

A computer with a power of 1Tflop/s will take $1.5 \bullet 10^{5}$ sec.

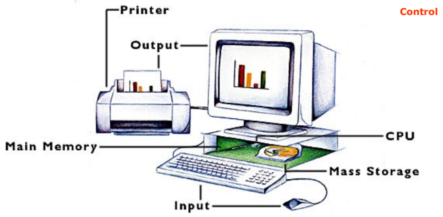
→ **24 hours forecast will need 2days to run ... but we shall obtain a very accurate forecast**

Von Neumann Model





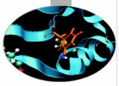
Von Neumann Model of Computer Architecture



Instructions are processed sequentially

- A single instruction is loaded from memory (fetch) and decoded
- Compute the addresses of operands
- Fetch the operands from memory;
- Execute the instruction ;
- Write the result in memory (store).

Speed of Processors: Clock Cycle and Frequency



The *clock cycle* τ is defined as the time between two adjacent pulses of oscillator that sets the time of the processor.

The number of these pulses per second is known as clock speed or clock frequency, generally measured in GHz (gigahertz, or billions of pulses per second).

The clock cycle controls the synchronization of operations in a computer: All the operations inside the processor last a multiple of τ.

Processor	τ (ns)	freq (MHz)
CDC 6600	100	10
Cyber 76	27.5	36,3
IBM ES 9000	9	111
Cray Y-MP C90	4.1	244
Intel i860	20	50
PC Pentium	< 0.5	> 2 GHz
Power PC	1.17	850
IBM Power 5	0.52	1.9 GHz
IBM Power 6	0.21	4.7 GHz

Increasing the clock frequency:

The **speed of light** sets an upper limit to the speed with which electronic components can operate .

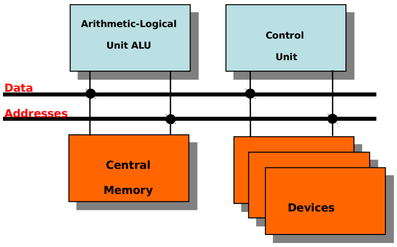
Propagation velocity of a signal in a vacuum: **300.000 Km/s = 30 cm/ns**

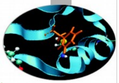
Heat dissipation problems inside the processor. Also Quantum tunelling expected to become important.

Other factors that affect Performance

In addition to processor power, other factors affect the performance of computers:

- Size of memory
- Bandwidth between processor and memory
- Bandwidth toward the I/O system
- Size and bandwidth of the cache
- Latency between processor, memory, and I/O system

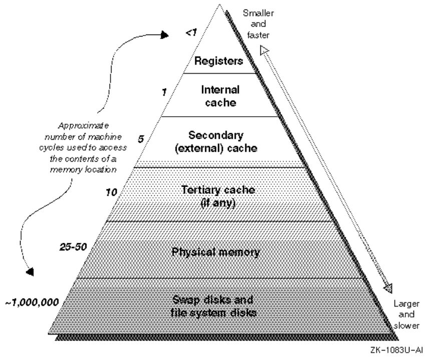




Memory hierarchies

Time to run code = clock cycles running code + clock cycles waiting for memory

Memory access time: the *time* required by the processor to *access* data or to write data from / to *memory*



Smaller and faster
<1
Registers
1
Internal cache
5
Secondary (external) cache
10
Tertiary cache (if any)
25-50
Physical memory
~1,000,000
Swap disks and file system disks
Larger and slower
Approximate number of machine cycles used to access the contents of a memory location
ZK-1083U-AI

The hierarchy exists because :

- fast memory is expensive and small
- slow memory is cheap and big

Latency

- – how long do I have to wait for the data?
- – (cannot do anything while waiting)

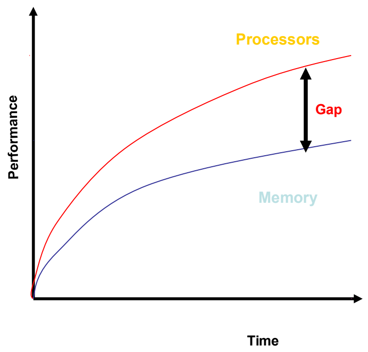
Throughput

- – how many bytes/second. but not important if waiting.

Total time = latency + (amount of data / throughput)

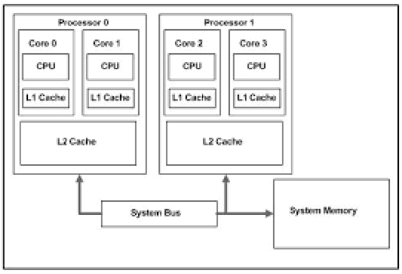
Memory access

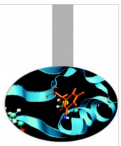
- Important problem for the performance of any computer is access to main memory. Fast processors are useless if memory access is slow!
- Over the years the difference in speed between processors and main memory has been growing.



Cache Memory

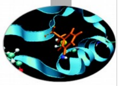
- High speed, small size memory used as a buffer between the main memory and the processor. When used correctly, reduces the time spent waiting for data from main memory.
- Present as various “levels” (e.g. L1, L2, L3, etc) according to proximity to the functional units of the processor.
- Cache efficiency depends on the locality of the data references:
  - *Temporal locality* refers to the re-use of data within relatively small time frame.
  - *Spatial locality refers to the use of data within close storage locations (e.g. one dimensional array).*
- *Cache can contain Data, Instructions or both.*



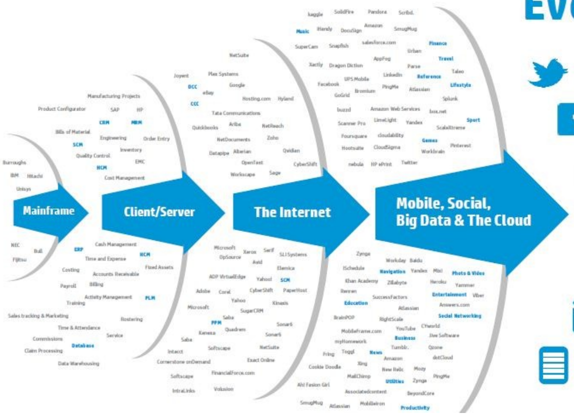


Aspects of parallelism

- It has been recognized for a long time that constant performance improvements cannot be obtained just by increasing factors such as processor clock speed – parallelism is needed.
- Parallelism can be present at many levels:
  - Functional parallelism within the CPU
  - Pipelining and vectorization
  - Multi-processor and multi-core
  - Accelerators
  - Parallel I/O



A new style of IT emerging



Every 60 seconds

98,000+ tweets

695,000 status updates

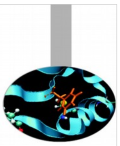
11million instant messages

698,445 Google searches

168 million+ emails sent

1,820TB of data created

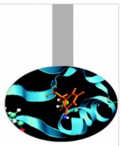
217 new mobile web users



Why?

- **Overwhelming amounts of data generated** by all kinds of devices, networks and programs, *e.g. sensors, mobile devices, internet, social networks, computer simulations, satellites, radiotelescopes, LHC, etc.*
- **Increasing storage capacity**
  - ✓ *Storage capacity has doubled every 3 years since 1980 with prices steadily going down*
  - ✓ *1,8 zetabytes: an estimation for the data stored by humankind in 2011 (Digital Universe study of International Data Corporation)*
- **Massive data can produce high-value information and knowledge**
- **Critical for data analysis, decision support, forecasting, business intelligence, research, (data-intensive) science, etc.**

Big Data



A buzz word!

- With different meanings depending on your perspective - e.g. 100 TBs is big for a transaction processing system, but small for a world-wide search engine

A simple "definition" (Wikipedia)

- ***Consists of data sets that grow so large that they become awkward to work with using on-hand database management tools***
  - Difficulties: capture, storage, search, sharing, analytics, visualizing

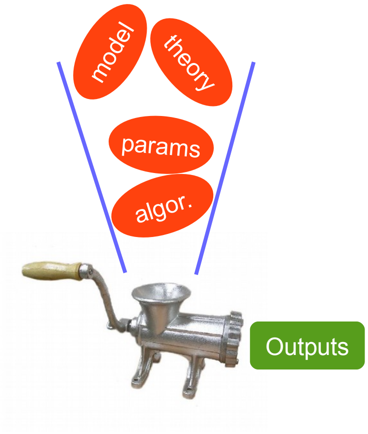
How big is big?

- Moving target: terabyte (1012 bytes), petabyte (1015 bytes), exabyte (1018), zetabyte (1021)

Scale is only one dimension of the problem!

Operate without models

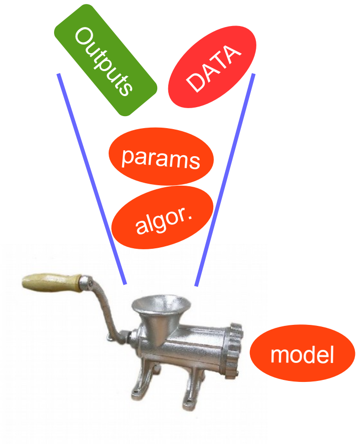
forward problem



Before...

Operate without models

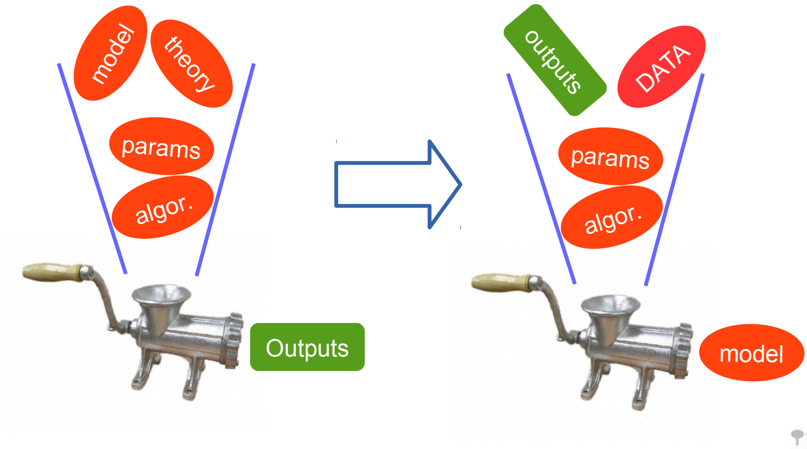
inverse problem



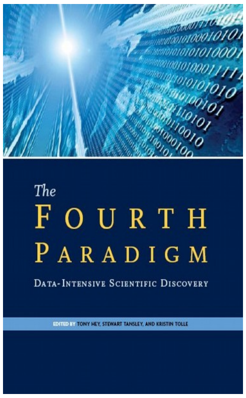
Now, future...

SCAI
SuperComputing Applications and Innovation

Operate without models (Big Data)



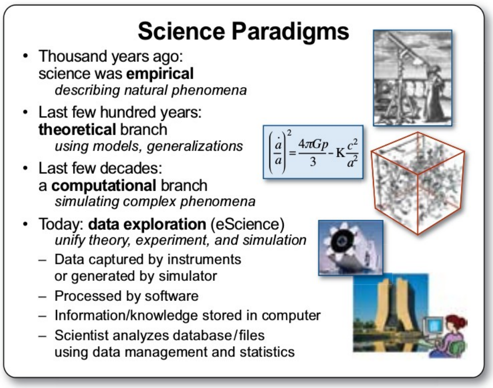
Where all began...a decade ago!



The FOURTH PARADIGM

DATA-INTENSIVE SCIENTIFIC DISCOVERY

EDITED BY TONY HEY, STEWART TANSLEY, AND KRISTIN TOLLE



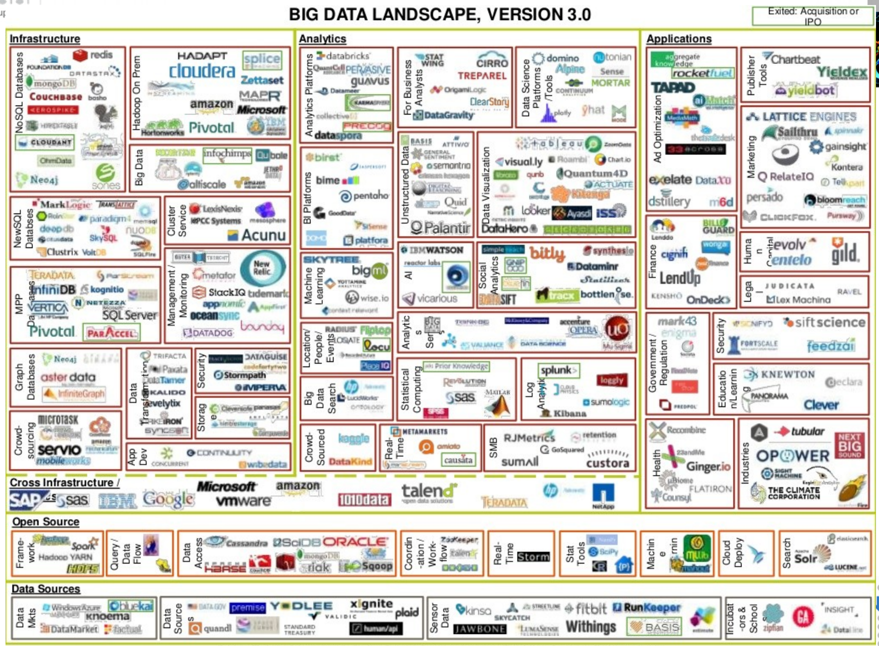
Science Paradigms

- Thousand years ago:
  science was **empirical**
  describing natural phenomena
- Last few hundred years:
  theoretical branch
  using models, generalizations
- Last few decades:
  a **computational** branch
  simulating complex phenomena
- Today: **data exploration** (eScience)
  unify theory, experiment, and simulation
  – Data captured by instruments
  or generated by simulator
  – Processed by software
  – Information/knowledge stored in computer
  – Scientist analyzes database/files
  using data management and statistics

$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{4\pi G\rho}{3} - K\frac{c^2}{a^2}$$

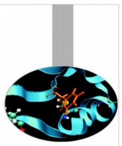
http://research.microsoft.com/en-us/collaboration/fourthparadigm/4th_paradigm_book_complete_lr.pdf

BIG DATA LANDSCAPE, VERSION 3.0

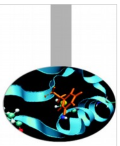


© Matt Turck (@mattturck), Sutian Dong (@sutiandong) & FirstMark Capital (@firstmarkcap)

Dimensions of the problem

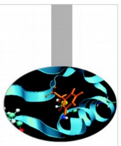


- **Volume**
  - Refers to massive amounts of data
  - Makes it hard to store and manage, but also to analyze (big analytics)
- **Velocity**
  - Continuous data streams are being captured (e.g. from sensors or mobile devices) and produced
  - Makes it hard to perform online processing
- **Variety**
  - Different data formats (sequences, graphs, arrays, …), different semantics, uncertain data (because of data capture), multiscale data (with lots of dimensions)
  - Makes it hard to integrate and analyze



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

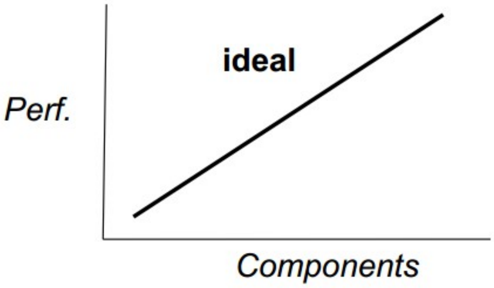
Parallel data processing!

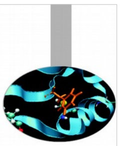


- **Exploit a massively parallel computer**
  - A computer that interconnects lots of CPUs, RAM and disk units
- **To obtain**
  - High performance through data-based parallelism
    - *High throughput for transaction-oriented (OLTP) loads*
    - *Low response time for decision-support (OLAP) queries*
  - *High availability and reliability through data replication*
  - Extensibility with the ideal goals
    - *Linear speed-up*
    - *Linear scale-up*

Speed-up

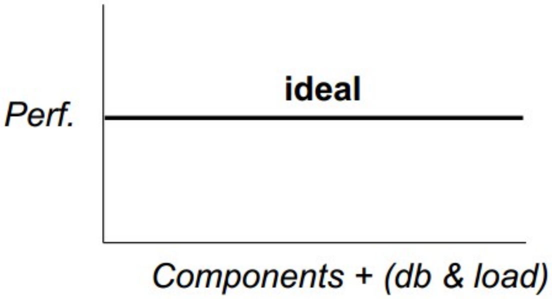
- Linear increase in performance for a constant database size and load, and proportional increase of the system components (CPU, memory, disk)





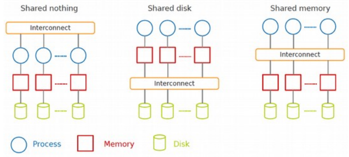
Scale-up

- Sustained performance for a linear increase of database size and load, and proportional increase of components

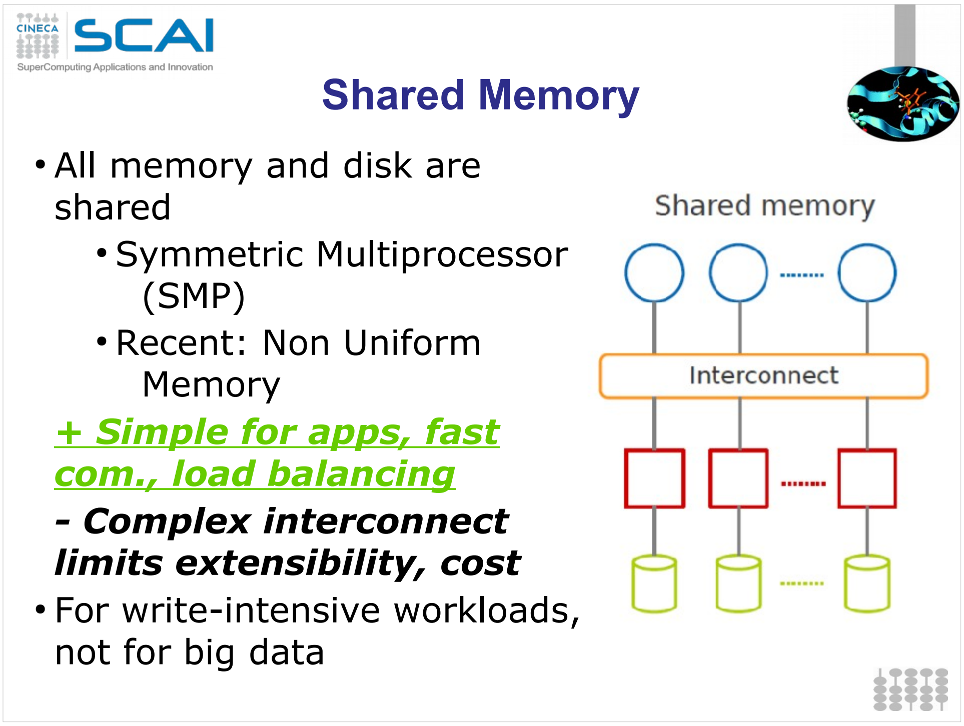


Parallel Architectures for Data Processing

- Three main alternatives, depending on how processors,memory and disk are interconnected
  - ***Shared-memory computer***
  - ***Shared-disk cluster***
  - ***Shared-nothing cluster***



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



Shared Memory

- All memory and disk are shared
  - Symmetric Multiprocessor (SMP)
  - Recent: Non Uniform Memory

+ Simple for apps, fast com., load balancing

- Complex interconnect limits extensibility, cost

- For write-intensive workloads, not for big data

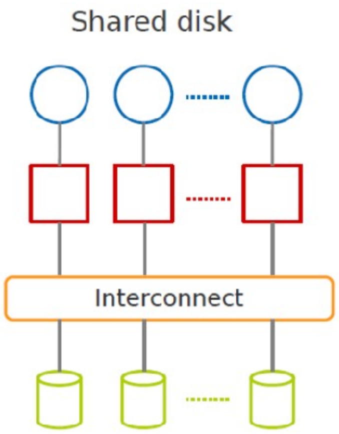
Shared Disk

- Disk is shared, memory is private
  - Storage Area Network (SAN) to interconnect memory and disk (block level)
  - Needs distributed lock manager (DLM) for cache coherence

+ Simple for apps, extensibility

- Complex DLM, cost

- For write-intensive workloads or big data



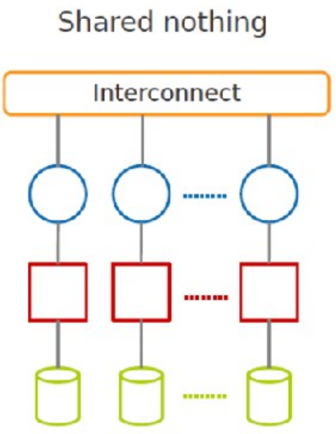
Shared Nothing

- No sharing of memory or disk across nodes
  - No need for DLM
  - But needs data partitioning

+ highest extensibility, cost

- updates, distributed trans

- For **big data** (read intensive)

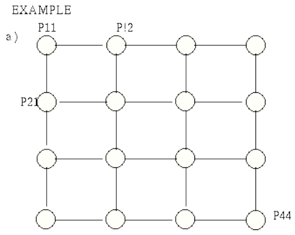


Example networks

Some variations of the mesh model have wrap-around type connections between the nodes to the edges of the mesh (torus topology).

The Cray T3E adopts a 3D torus topology

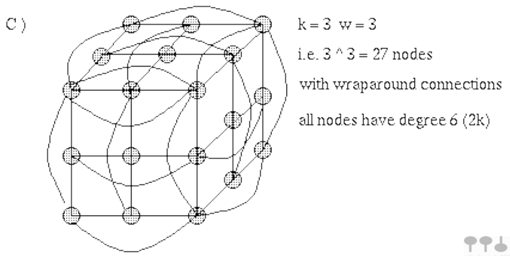
IBM BG/Q adopts a 5D torus topology



2D mesh of width 4 with
no wraparound connections
on edge or corner nodes

corner nodes have degree 2
edge nodes have degree 3

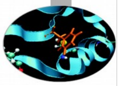
MESH Topology



k = 3 w = 3
i.e. 3 ^ 3 = 27 nodes
with wraparound connections
all nodes have degree 6 (2k)

Toroidal Topology

Commodity Interconnects



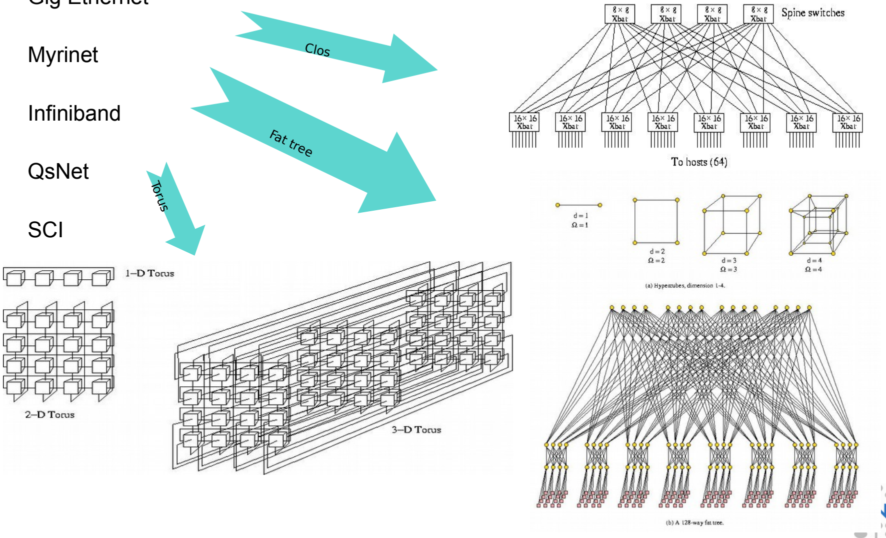
Gig Ethernet

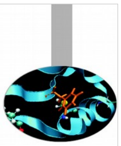
Myrinet

Infiniband

QsNet

SCI





Simple Model for Parallel Data

Shared-nothing architecture

- The most general and scalable

Set-oriented

- Each dataset **D** is represented by a table of rows

Key-value

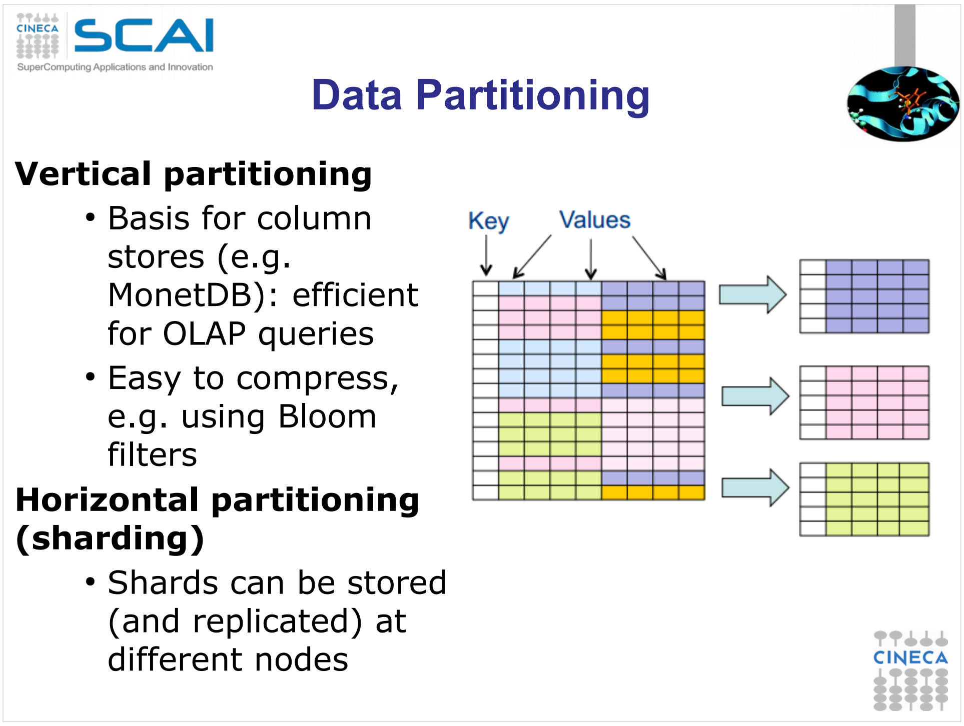
- Each row is represented by a <key, value> pair where
  - *Key uniquely identifies the value in D*
  - *Value is a list of (attribute name : attribute value)*

Can represent structured (relational) data or NoSQL data

- But graph is another story (see Pregel, DEX or Spark)

Examples

- *<row-id#5, (part-id:5, part-name:iphone5, supplier:Apple)>*
- *<doc-id#10, (content:<html> html text … </html>)>*
- *<akeyword, (doc-id:id1, doc-id:id2, doc-id:id10)>*



Data Partitioning

Vertical partitioning

- Basis for column stores (e.g. MonetDB): efficient for OLAP queries
- Easy to compress, e.g. using Bloom filters

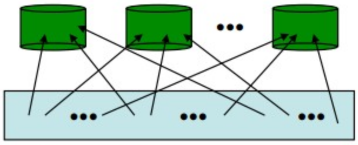
Horizontal partitioning (sharding)

- Shards can be stored (and replicated) at different nodes

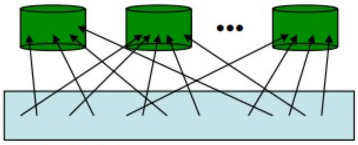
a query returns either "possibly in set" or "definitely not in set"

On-Line Analytical Processing

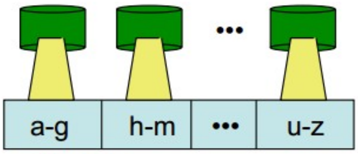
Sharding Schemes



Round-Robin
- *i*th row to node (*i mod n*)
- perfect balancing
- but full scan only

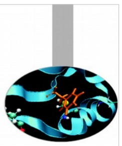


Hashing
- (k,v) to node h(k)
- exact-match queries
- but problem with skew



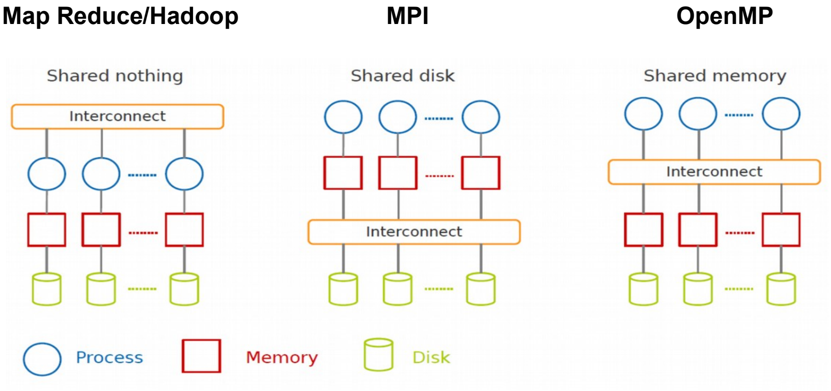
Range
- (k,v) to node that holds k's interval
- exact-match and range queries
- deals with skew

Different classes of applications

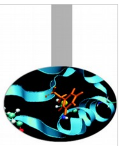


- **MPI (Message Passing Interface)**
  - A shared disk infrastructure for processing large data sets with a parallel algorithm on clusters
- **OpenMP (Open MultiProcessing)**
  - A shared memory infrastructure for processing large data sets with a parallel algorithm on a node
- **Map Reduce (Hadoop)**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

Parallel Architectures



Programming Models: What is MPI?



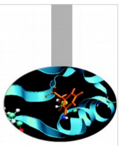
- **Message Passing Interface (MPI)**
  - World’s most popular distributed API
  - MPI is “de facto standard” in scientific computing
  - C and FORTRAN, ver. 2 in 1997

- **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

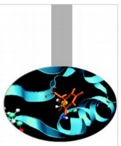
- **Basics:**
  - One executable run on every node
  - Each node process has a rank ID number assigned
  - Call API functions to send messages

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

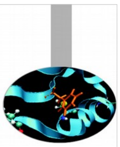
Challenges with MPI



- **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!**



Brief recap

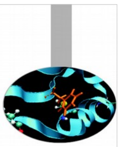


Map Reduce flow

- Programmers must specify:
  map (k, v) → list(<k’, v’>)
  reduce (k’, list(v’)) → <k’’, v’’>
  – All values with the same key are reduced together

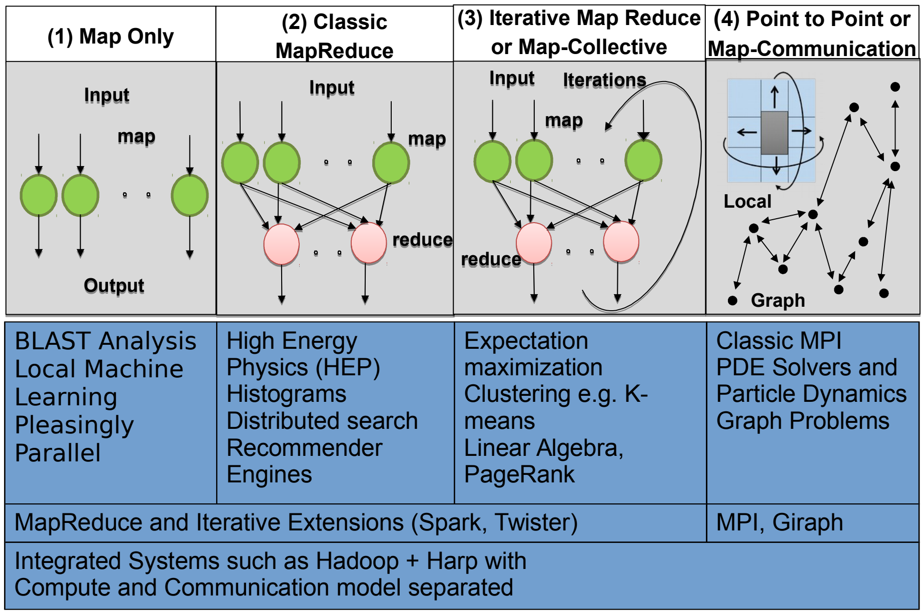
- Optionally, also:
  partition (k’, number of partitions) → partition for k’
  – Often a simple hash of the key, e.g., hash(k’) mod n
  – Divides up key space for parallel reduce operations
  combine (k’, v’) → <k’, v’>*
  – Mini-reducers that run in memory after the map phase
  – Used as an optimization to reduce network traffic

- The execution framework handles everything else…

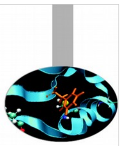


“Everything Else”

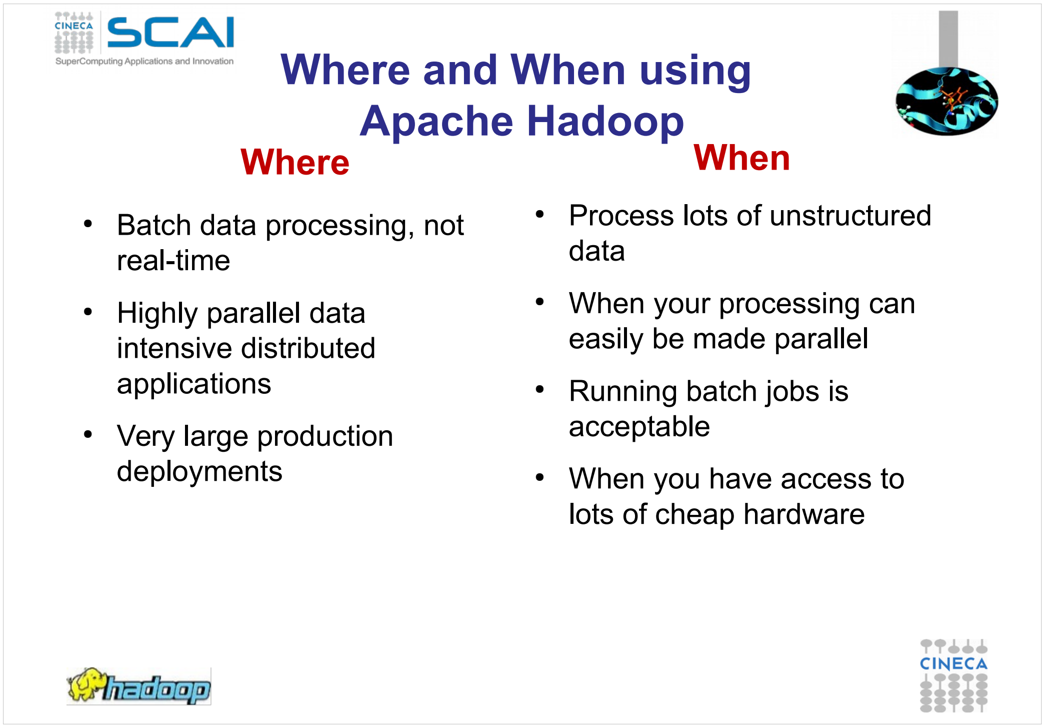
- The execution framework handles everything else…
  - **Scheduling**: assigns workers to map and reduce tasks
  - **“Data distribution”**: moves processes to data
  - **Synchronization**: gathers, sorts, and shuffles intermediate data
  - **Errors and faults**: detects worker failures and restarts
- Limited control over data and execution flow
  - All algorithms must expressed in m, r, c, p
- You don’t know:
  - Where mappers and reducers run
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which intermediate key a particular reducer is processing



(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			



Are emerging data analytics techniques the new El Dorado?



Where and When using Apache Hadoop

Where

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

When

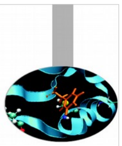
- Process lots of unstructured data
- When your processing can easily be made parallel
- Running batch jobs is acceptable
- When you have access to lots of cheap hardware

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

Parallel Computing Model



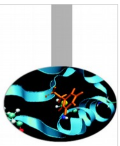
MapReduce can be classified as a SIMD (single-instruction, multiple-data) problem.

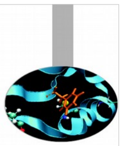
- ✔ Indeed, the map step is highly scalable because the same instructions are carried out over all data. Parallelism arises by breaking the data into independent parts with no forward or backward dependencies (side effects) within a Map step; that is, the Map step may not change any data (even its own).
- ✔ 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.

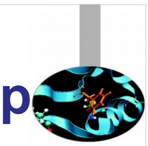
A Tale of Two Data-Intensive Paradigs: Applications, Abstractions, and Architectures
Shantenu Jha , Judy Qiu, Andre Luckow , Pradeep Mantha , Geoffrey C.Fox





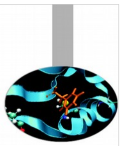
When to use Apache Hadoop

When to use Apache Hadoop

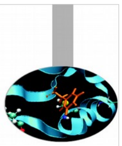


- **Your Data Sets Are Really Big**
  - *Don’t even think about Hadoop if the data you want to process is measured in MBs or GBs.* If the data driving the main problem you are hoping to use Hadoop to solve is measured in GBs, save yourself the hassle and use Excel, a SQL BI tool on Postgres, or some 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 MapReduce



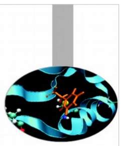
- **You Find Yourself Throwing Away Perfectly Good Data**
  - One of the great things about Hadoop is its capability to store petabytes of data. If you find that you are throwing away potentially valuable data because its costs too much to archive, you may find that setting up a Hadoop cluster allows 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 MapReduce

- **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.

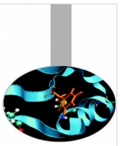
When to NOT use MapReduce



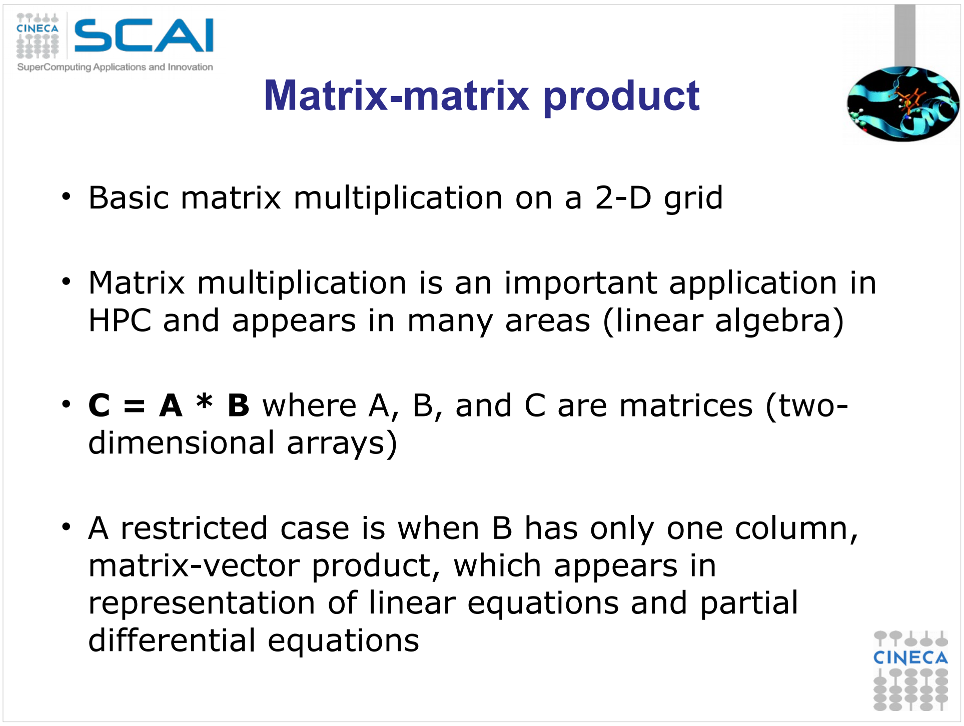
- **You Require Random, Interactive Access to Data**
  - The pushback from the limitations of the batch-oriented MapReduce paradigm in early Hadoop led the community to improve SQL performance and boost its capability to serve interactive queries against random data. While SQL on Hadoop is getting better, 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



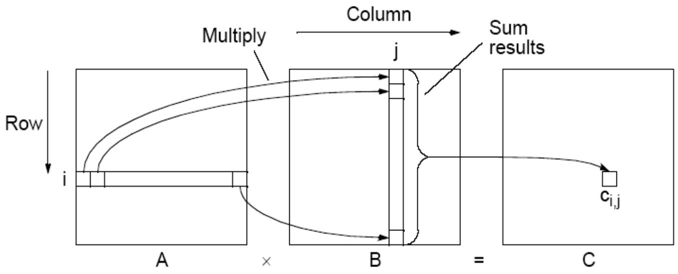
Matrix-matrix product v1



Matrix-matrix product

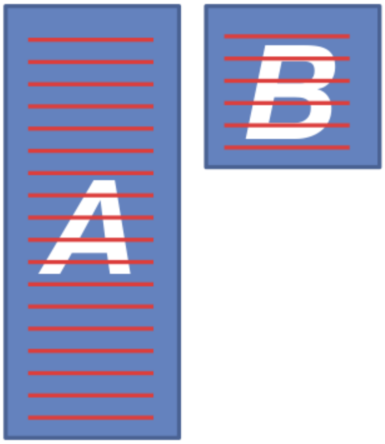
- Basic matrix multiplication on a 2-D grid
- Matrix multiplication is an important application in HPC and appears in many areas (linear algebra)
- **C = A * B** where A, B, and C are matrices (two-dimensional arrays)
- A restricted case is when B has only one column, matrix-vector product, which appears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

C = A x B



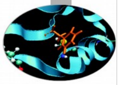
$$c_{i,\,j} = \sum_{k\,=\,0}^{l\,-\,1} a_{i,k}b_{k,j}$$

Matrix-matrix product



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

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



Matrix-matrix product

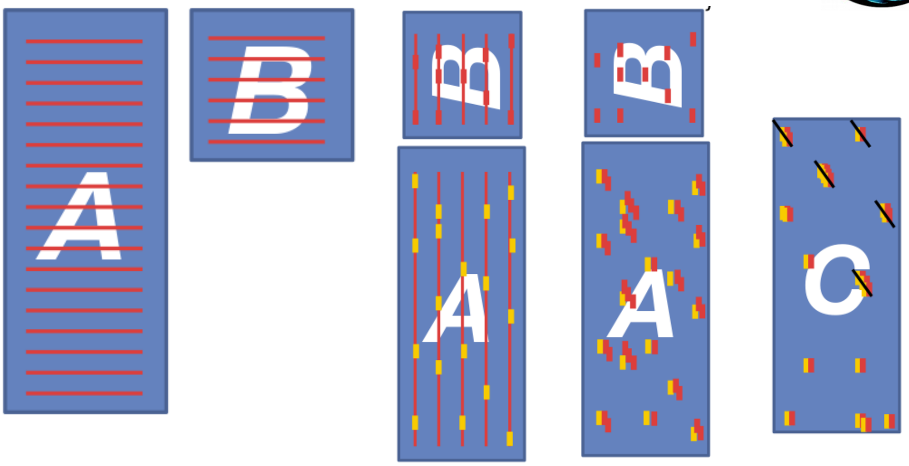
A is stored by row (`$ head data/mat/smat_10x5_A`)

```
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



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

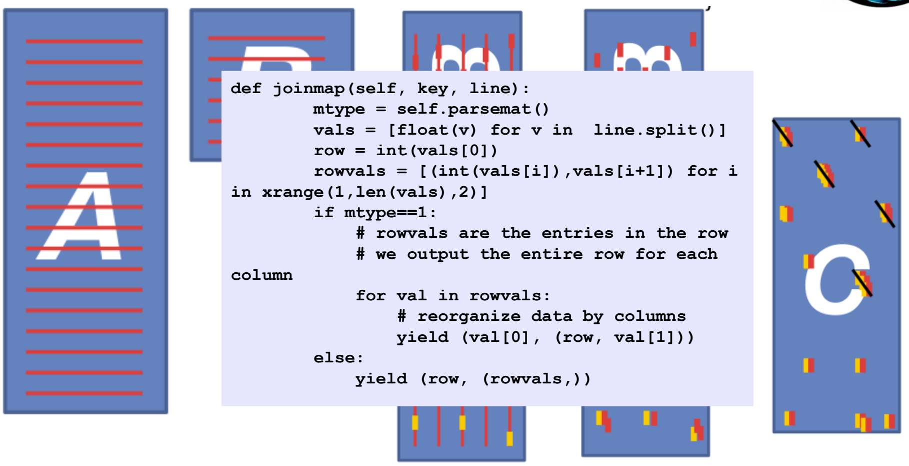
$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Map 1
Align on columns

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

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

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,))
```

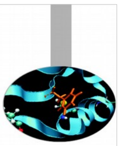
$$AB = C$$

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

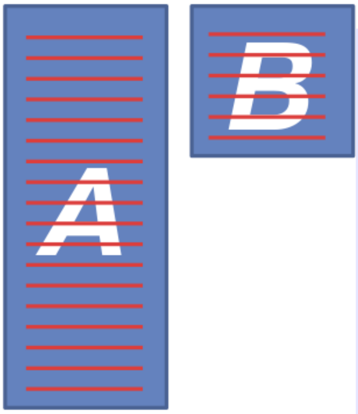
Map 1
Align on columns

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)
```

$$AB = C$$

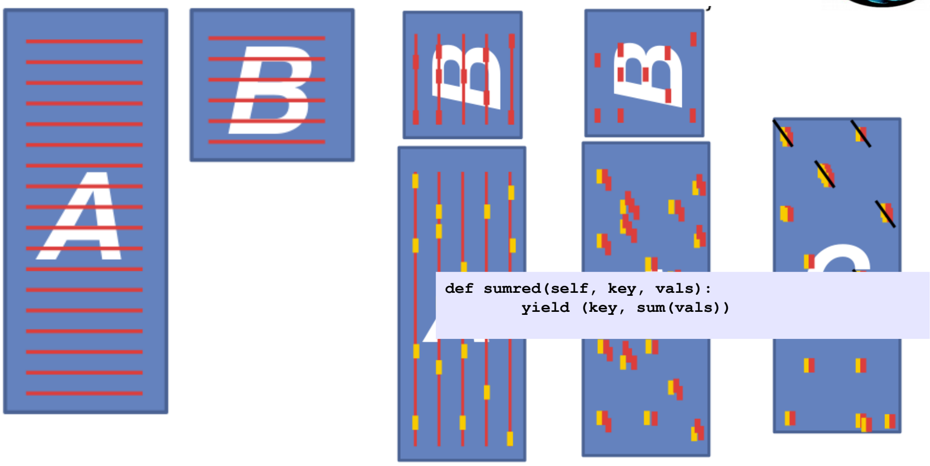
$$C_{ij} = \sum_k A_{ik} B_{kj}$$

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)

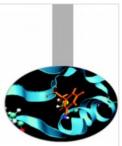
Reduce 2
Output
sum(A_{ik}, B_{kj})

SCAI SuperComputing Applications and Innovation

sparse_matmat.py

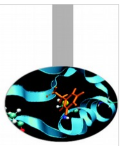
```
from mrjob.job import MRJob
from mrjob.compat import get_jobconf_value
import itertools
import sys

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()
```



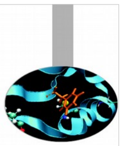
Matrix-matrix product v2

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

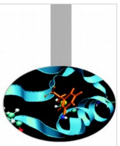
The Reduce Function:

- 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

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)
```

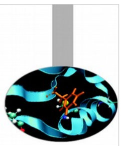
```
  def multiply_values(self, j, values):

       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
```



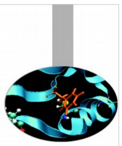
Log based debug

- Python

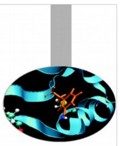
```
sys.stderr(out).write("REDUCER INPUT: ({0},{1})\n".format(j,
values))
```

- Java

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

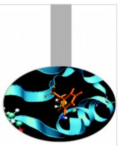


Debugging



Debug mechanisms

- The Web Interface
- Runtime monitor
- Log based debug

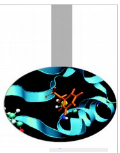


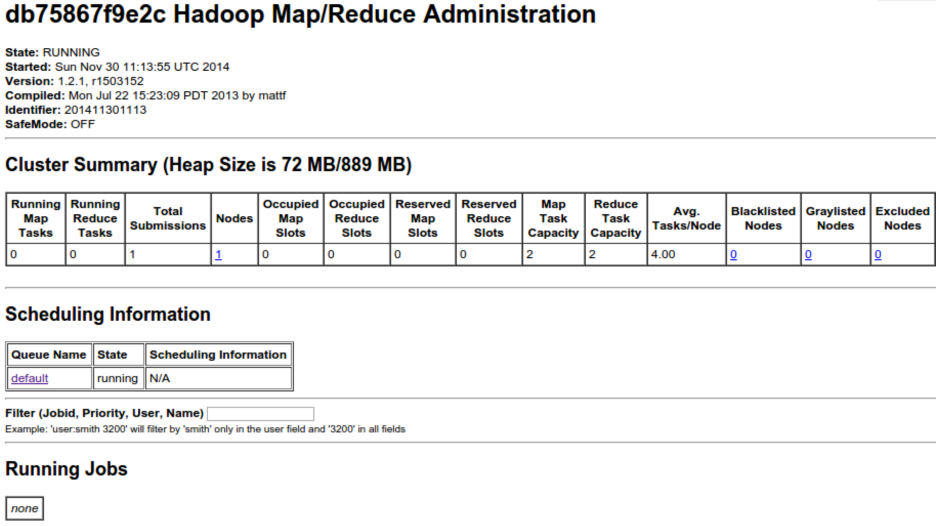
The Web User Interface

- **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

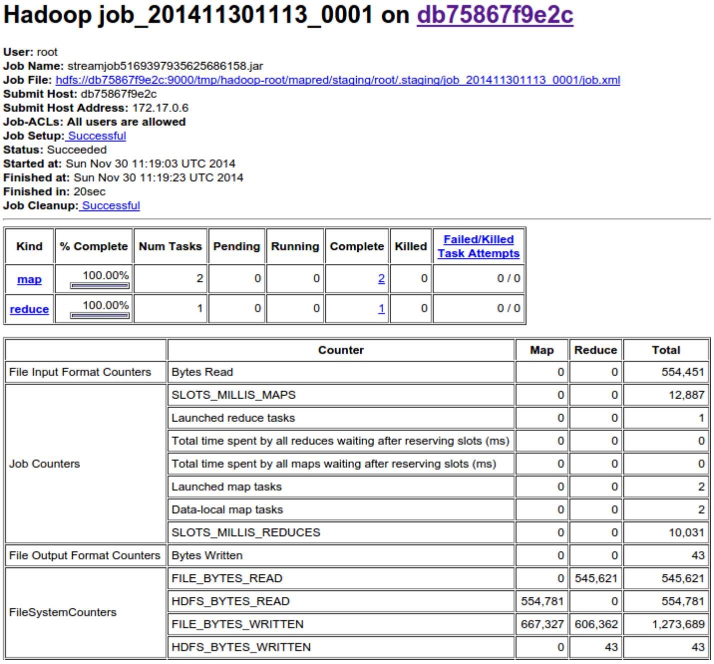
Scheduling Information

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

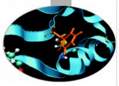


Hadoop job_201411301113_0001 on db75867f9e2c

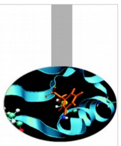
User: root
Job Name: streamjob5169397935625686158.jar
Job File: hdfs://db75867f9e2c:9000/tmp/hadoop-root/mapred/staging/root/.staging/job_201411301113_0001/job.xml
Submit Host: db75867f9e2c
Submit Host Address: 172.17.0.6
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started 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



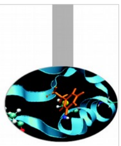
Hadoop Reporter



- 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.

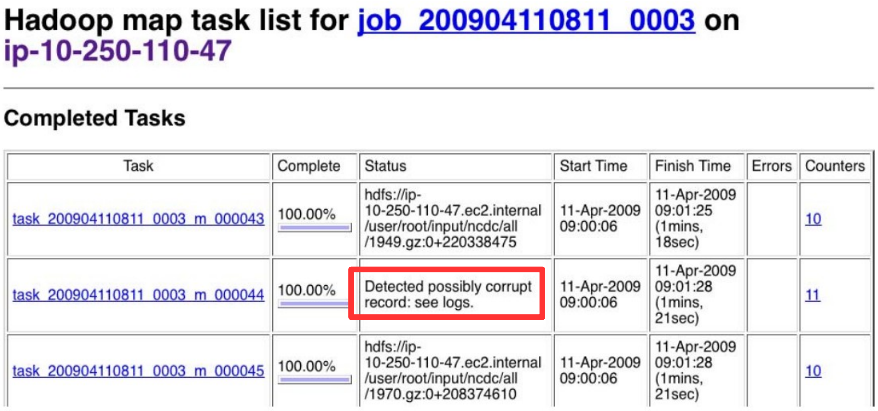


Hadoop Reporter

“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

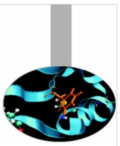


Hadoop map task list for job_200904110811_0003 on ip-10-250-110-47

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

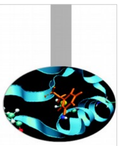


- The Java Platform Debugger Architecture is a collection of APIs to debug Java code.

- 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

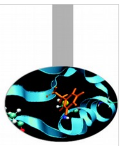
- **Python**

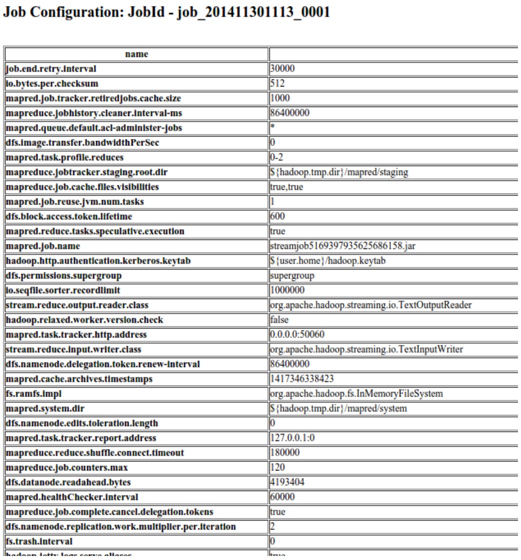
```
sys.stderr(out).write("REDUCER INPUT: ({0},{1})\n".format(j,
values))
```

- **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

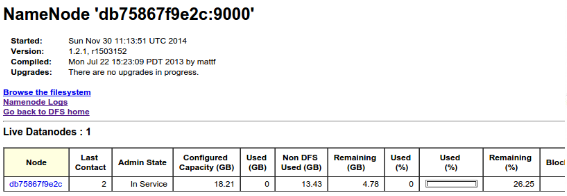
- Like debugging, profiling a job running on a distributed system like MapReduce presents some challenges. Hadoop allows you to profile a fraction of the tasks in a job, and, as each task completes, pulls down the profile information to your machine for later analysis with standard profiling tools.
- **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

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



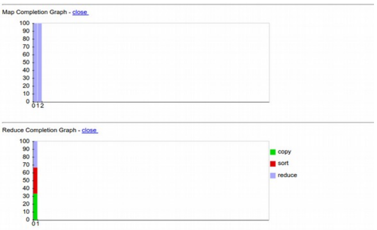
NameNode 'db75867f9e2c:9000'

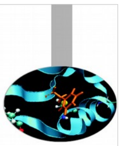
Started: Sun Nov 30 11:13:51 UTC 2014
Version: 1.2.1, r1503152
Compiled: Mon Jul 22 15:23:09 PDT 2013 by mattf
Upgrades: There are no upgrades in progress.

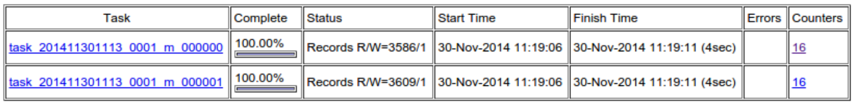
Browse the filesystem
Namenode Logs
Go back to DFS home

Live Datanodes : 1

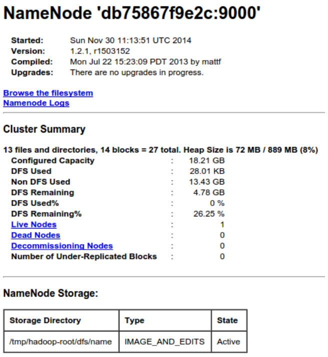
Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Bloc
db75867f9e2c	2	In Service	18.21	0	13.43	4.78	0	▭	26.25	







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

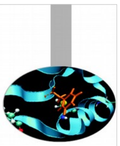
13 files and directories, 14 blocks = 27 total. Heap Size is 72 MB / 889 MB (8%)

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

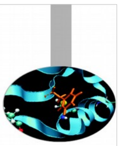
Cluster optimizations



The problem:

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

Hadoop basic options

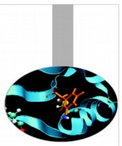


All hadoop commands are invoked by the bin/hadoop script. Running the hadoop script without any arguments prints the description for all commands.

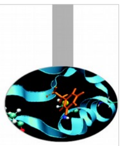
```
Usage: hadoop [--config confdir] [COMMAND]
   [GENERIC_OPTIONS] [COMMAND_OPTIONS]
```

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

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.
```

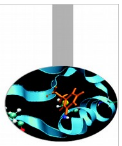


Configuration parameters

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>
```



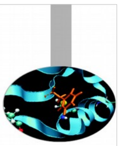
Speculative Execution

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
```

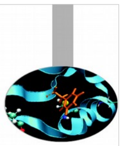
Speculative execution



- 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

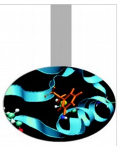


Number of Maps/Reducers

Number of Maps/Reducers

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

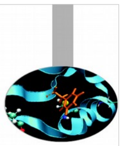
- **Default**: 2
- **Suggestions**: Recommended range - (cores_per_node)/2 to 2x(cores_per_node), especially for large clusters. This value should be set according to the hardware specification of cluster nodes and resource requirements of tasks (map/reduce).



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.

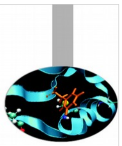


Sort size

Sort size *io.sort.mb →* Buffer size (MBs) for sorting

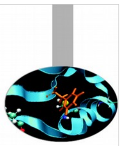
- **Default**: 100
- **Suggestions**: For Large jobs (the jobs in which map output is very large), this value should be increased keeping in mind that it will increase the memory required by each map task. So the increment in this value should be according to the available memory at the node. Greater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

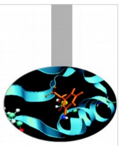


JVM Reuse

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.