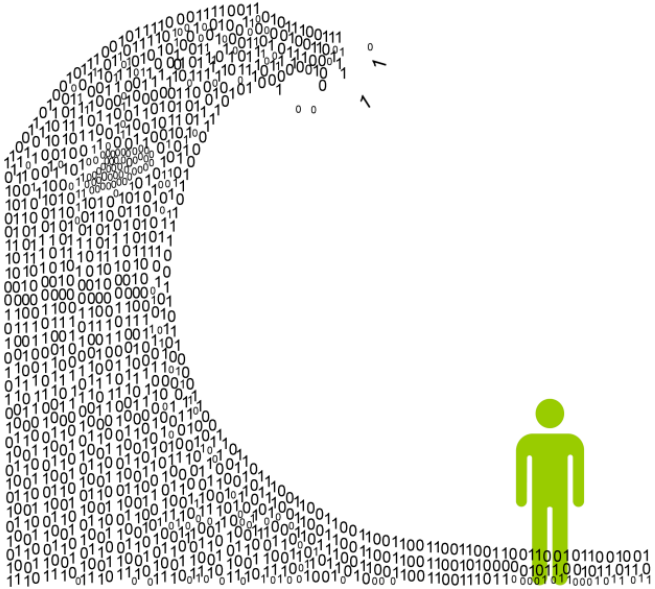
Hadoop

Bioinformatics Big Data

Paolo D'Onorio De Meo p.donoriodemeo@cineca.it
Mattia D'Antonio m.dantonio@cineca.it

Big Data

Too much information!

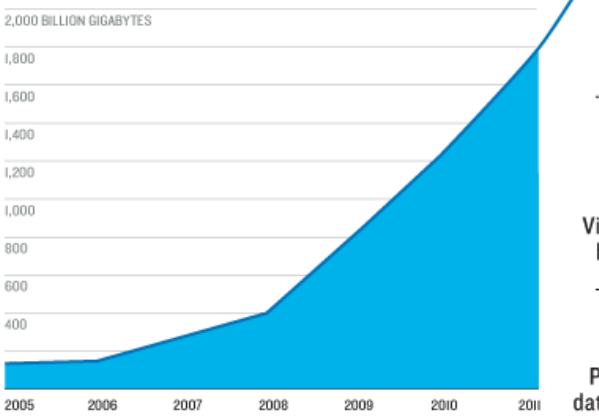


65 billion
Location-tagged payments made in the U.S. annually

154 billion
E-mails sent per day

87%
U.S. adults whose location is known via their mobile phone

Digital Information Created Each Year, Globally



2,000%
Expected increase in global data by 2020

III Megabytes
Video and photos stored by Facebook, per user

75%
Percentage of all digital data created by consumers

Sources: IDC, Radicati Group, Facebook, TR research, Pew Internet

Big Data

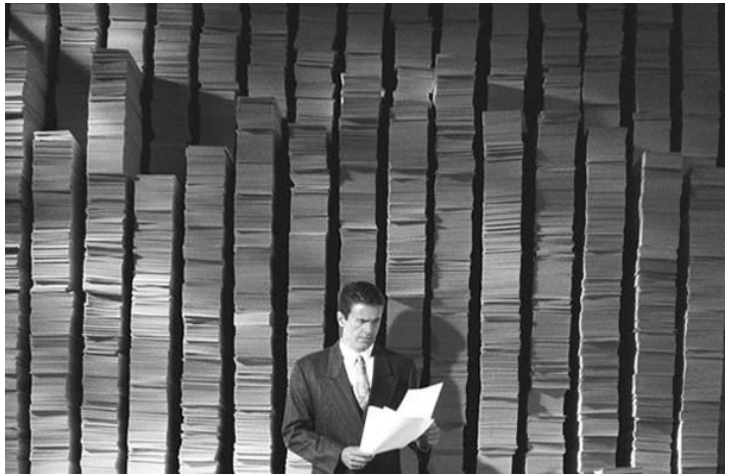
- Explosive data growth
  - proliferation of data capture
  - interconnection = more data
  - inexpensive storage
- Not just the size of data
  - access all the data
  - increase retention
  - machine learning
  - data compression

Big Data

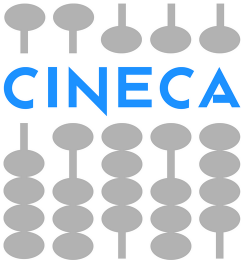
Where are you going?

Big Data main problem

Data analysis is slower than data creation

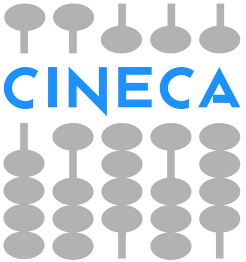


Big Data



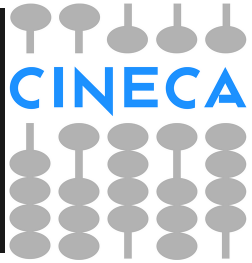
- Semi-structured data
  - looser
  - though there may be a schema, it is often *ignored*
- Why can’t we use **databases** to do large-scale batch analysis?
  - *seek time* is improving slowly than *transfer rate*

Old approach



- HPC and Grid Computing
  - doing large-scale data processing for years:
    - APIs as Message Passing Interface (MPI)
    - distribute the work across a cluster of machines
    - access a shared filesystem (hosted by a SAN)
  - Works well for compute-intensive jobs
    - becomes a problem when nodes need to access larger data volumes
    - the network bandwidth is the bottleneck

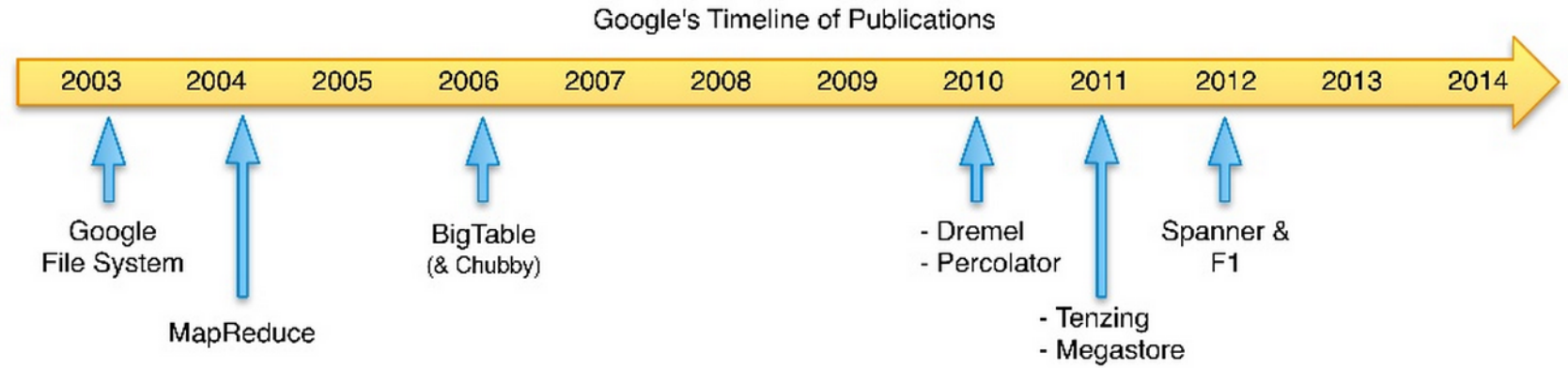
Bandwidth



"bandwidth is the bottleneck and compute nodes become idle"

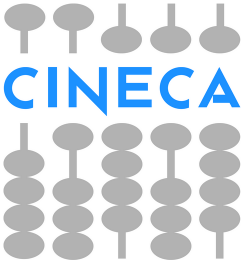
- HPC and Grid can be overloaded
- Bandwidth solutions focus on obtaining better performance for very fast workloads

Google approach



10 years ago: the famous MapReduce paper

MapReduce

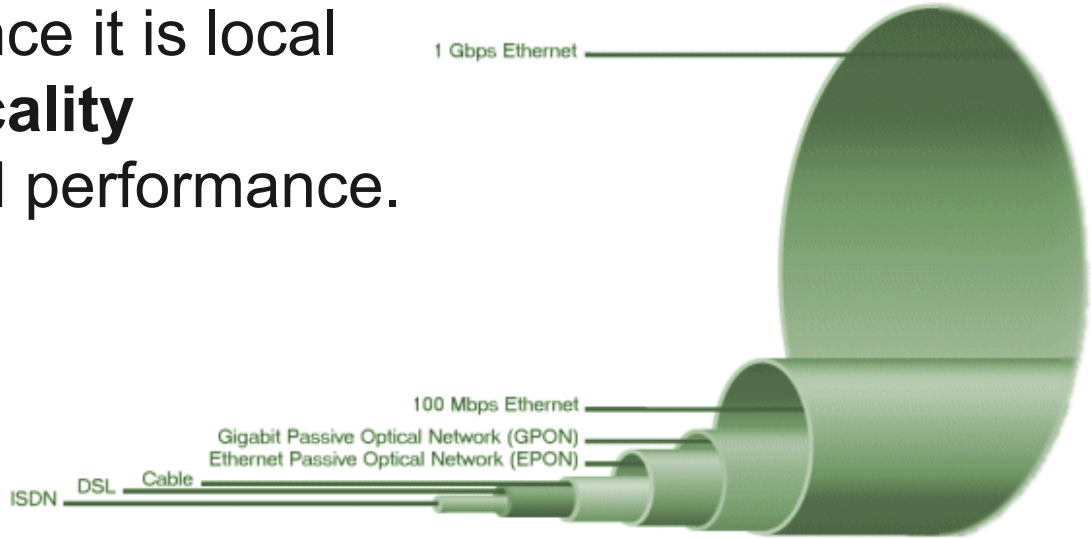


- Batch query processor
  - ability to run an *ad hoc queries* against your whole dataset
  - get the results in a reasonable time
- Unstructured or semi-structured data
  - designed to interpret the data at processing time

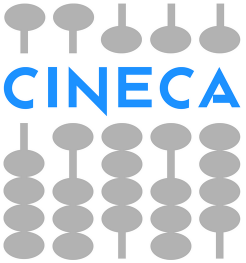
MapReduce

BANDWIDTH

- tries to collocate the data with the compute node
- data access is fast since it is local
  - known as **data locality**
  - reason for its good performance.



Apache Hadoop



- Apache works on Lucene (2005)
  - Full-featured text search engine library
  - Indexing system from scratch
  - Decide to go with MapReduce
  - Splits into a new project Hadoop (2006)
- April 2008
  - Hadoop broke a world record to become the **fastest system** to sort a terabyte of data

Apache Hadoop

CINECA

- Open source platform
  - for data storage and processing
- Scalable
- Fault tolerant
- Distributed

HDFS

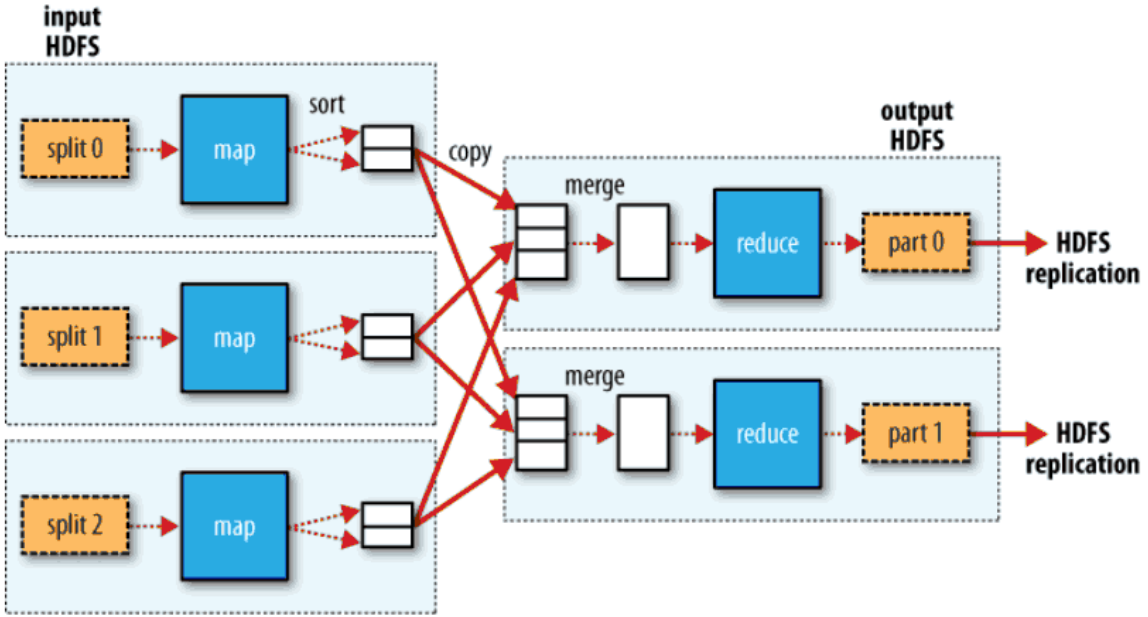
CINECA

Hadoop File System

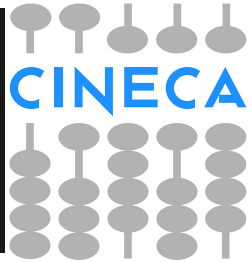
- Build to avoid transferring data over the network
- Hadoop works on input ***splits***
  - Split time << job execution
  - Small splits
    - faster nodes consumes more splits and jobs than slowers ones
  - If too small overhead breaks performance
  - Fine tuning
  - Best split = HDFS size (64MB default)
- Hadoop needs **topography**
  - don’t distribute on different racks if not needed
  - *Data locality optimization*

MapReduce Hadoop jobs

- Single Job
  - Map tasks
    - build splits
    - local outputs
  - Reduce tasks
    - HDFS output redundant



The hadoop framework



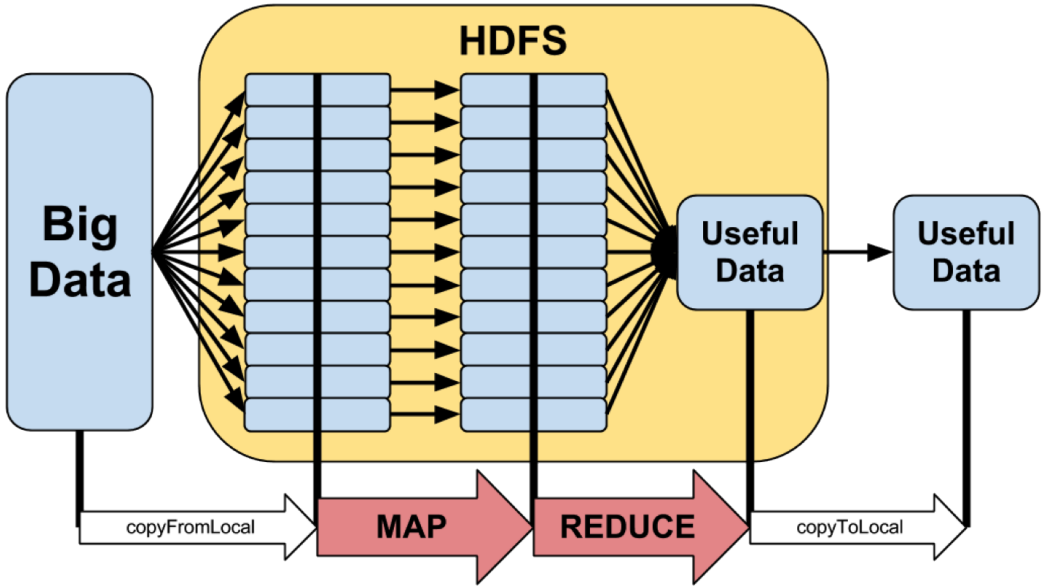
- Hadoop is written in Java
- Its main framework is Java-based
- Write code in many languages (e.g. Python).
- API to check cluster status and configuration

Hadoop: hands on

To work on any example, even the simplest, you clearly need a *Hadoop Cluster*.

Two ways of simulating a Hadoop cluster on your local machine:

1. A pseudo distributed single-node Hadoop cluster on Linux/Ubuntu
2. A pre-configured virtual machine



Hadoop: hands on

CINECA

A python example

Why python?

- Not native
  - Which will help you to better understand the Hadoop system
- Easy to write code for **Mappers** and **Reducers**

Hadoop: hands on

Files example (columns)

Stadium (String) - The name of the stadium
Capacity (Int) - The capacity of the stadium
ExpandedCapacity (Int) - The expanded capacity of the stadium
Location (String) - The location of the stadium
PlayingSurface (String) - The type of grass, etc that the stadium has
IsArtificial (Boolean) - Is the playing surface artificial
Team (String) - The name of the team that plays at the stadium
Opened (Int) - The year the stadium opened
WeatherStation (String) - The name of the weather station closest to the stadium
RoofType (Possible Values:None,Retractable,Dome) - The type of roof in the stadium
Elevation - The elevation of the stadium

Our question:

Find the *number* of stadiums with **artificial** and **natural** playing surfaces

Python Mapper

(mapper.py)

```
for line in sys.stdin:
    line = line.strip()
    stadium, capacity, expanded, location,
surface, turf, team, opened, weather,
roof, elevation = line.split(",")
    results = [turf, "1"]
    print("\t".join(results))
```

In the middle?

```
...
streaming...
TRUE 1
TRUE 1
TRUE 1
TRUE 1
FALSE 1
FALSE 1
FALSE 1
```

The reducer interface for streaming is actually different than in Java. Instead of receiving

`reduce(k, Iterator[V])`

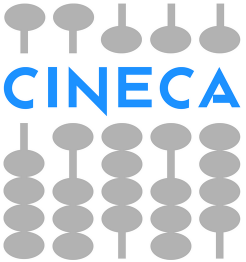
your script is actually sent one line per value, including the key.

Python Reducer

(reducer.py)

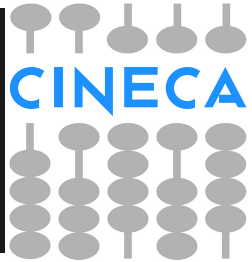
```
last_turf = None
turf_count = 0

for line in sys.stdin:
    line = line.strip()
    turf, count = line.split("\t")
    count = int(count)
    if not last_turf:              # if this is the first iteration
        last_turf = turf
    if turf == last_turf:          # if they're the same, log it
        turf_count += count
    else:                          # state change
        result = [last_turf, turf_count]
        print("\t".join(str(v) for v in result))
        last_turf = turf
        turf_count = 1

#catch the final counts after all records have been received.
print("\t".join(str(v) for v in [last_turf, turf_count]))
```

Testing on Hadoop




```
$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-
streaming-2.0.0-mr1-cdh4.4.0.jar \
    -mapper mapper.py \
    -reducer reducer.py \
    -input nfldata/stadiums \
    -output nfldata/pythonoutput \
    -file simple/mapper.py \
    -file simple/reducer.py


# ...twiddle thumbs for a while

$ hadoop fs -text nfldata/pythonoutput/part-*
FALSE 15
TRUE 17
```

Test… on a laptop

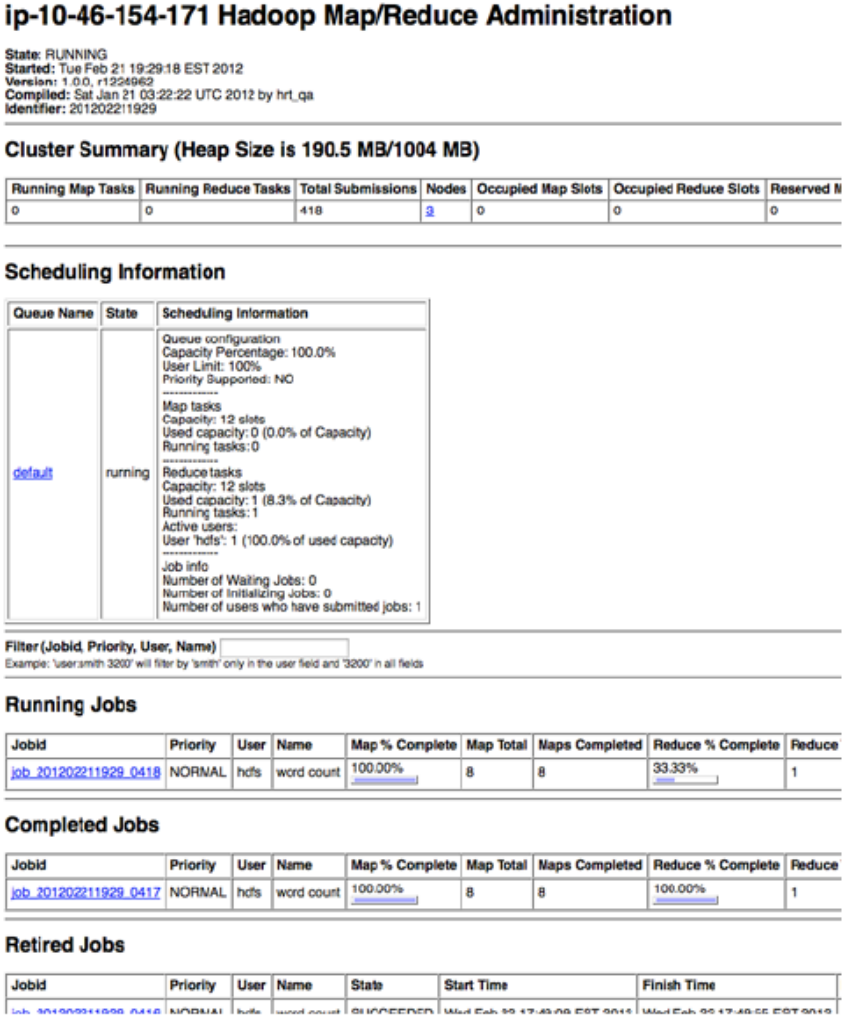



```
# Testing the same code as a bash pipe

$ cat ~/workspace/nfldata/unixstadiums.csv | simple/mapper.py | sort | simple/reducer.
py

# FALSE 15
# TRUE 17
```

Jobtracker

Hadoop web-dashboard:

Status and statistics of job executed on our Hadoop cluster



ip-10-46-154-171 Hadoop Map/Reduce Administration

State: RUNNING
Started: Tue Feb 21 19:29:18 EST 2012
Version: 1.0.0, r1224962
Compiled: Sat Jan 21 03:22:22 UTC 2012 by hrt_qa
Identifier: 201202211929

Cluster Summary (Heap Size is 190.5 MB/1004 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved M
0	0	418	a	0	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	Queue configuration Capacity Percentage: 100.0% User Limit: 100% Priority Supported: NO --- Map tasks Capacity: 12 slots Used capacity: 0 (0.0% of Capacity) Running tasks: 0 --- Reduce tasks Capacity: 12 slots Used capacity: 1 (8.3% of Capacity) Running tasks: 1 Active users: User 'hdfs': 1 (100.0% of used capacity) --- Job info Number of Waiting Jobs: 0 Number of Initializing Jobs: 0 Number of users who have submitted jobs: 1

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

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce
job_201202211929_0418	NORMAL	hdfs	word count	100.00%	8	8	33.33%	1

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce
job_201202211929_0417	NORMAL	hdfs	word count	100.00%	8	8	100.00%	1

Retired Jobs

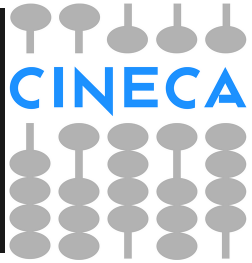
Jobid	Priority	User	Name	State	Start Time	Finish Time

A new cluster prototype

Working @CINECA

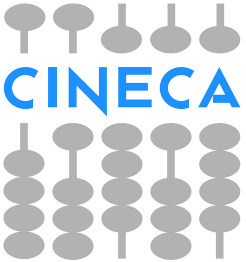
- Cloud
- Virtual nodes
- Virtual networks
- Virtual FS
- OpenStack
- Hadoop

NGS and bioinformatics



- Next Generation Sequencing = **NGS**
  - new platforms
  - high throughput
- Many analysis application
- New algorithms & codes and challenges
- Small costs!
- Producing Big Data

Why NGS fits Hadoop



Embarassingly parallel

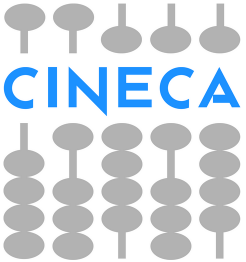
- Little or no effort to separate the problem into a number of parallel tasks
- Often no dependency (or communication) between parallel tasks

VS

Distributed system

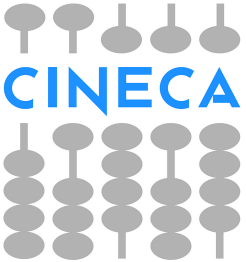
- Components are located on networked computers
- Which communicate and coordinate their actions by passing messages

NGS Hadoop today



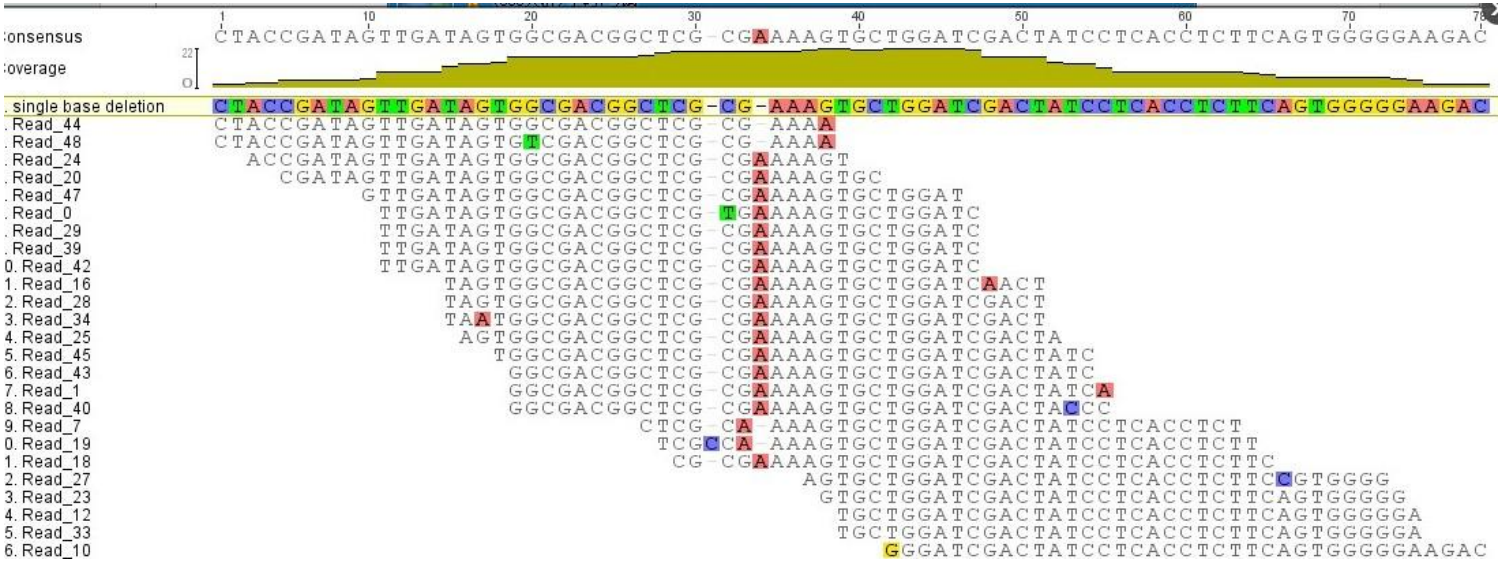
- Hadoop bam (mapping utilities)
  - http://bioinformatics.oxfordjournals.org/content/28/6/876
- Solve bio
  - http://techcrunch.com/2012/03/29/cloud-will-cure-cancer/
- Crossbow (mapping)
  - http://bowtie-bio.sourceforge.net/crossbow/index.shtml
- Cloudburst
  - http://sourceforge.net/apps/mediawiki/cloudburst-bio/index.php?title=CloudBurst
- Eoulsan (RNAseq)
  - http://transcriptome.ens.fr/eoulsan/
- Myrna (RNAseq gene expression)
  - http://bowtie-bio.sourceforge.net/myrna/manual.shtml
- SeqPig (Hadoop Pig for processing sequences)
  - http://sourceforge.net/projects/seqpig/
- Next Bio + Intel!!

Coverage problem

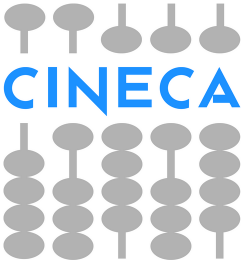


- New sequencing platforms produce big data files with many (short) sequences
- The *targeted sequencing* gives as output many sequences inside the same small genomic regions
- **Alignment**
  - mapping sequences on a reference genome
- **Coverage**
  - *how deep* is covered each genomic position in the experiment
  - base per base (one nucleotide at the time)
  - If coverage is too low (given a threshold) in one region we cannot use that region in our results

Coverage problem



SAM format



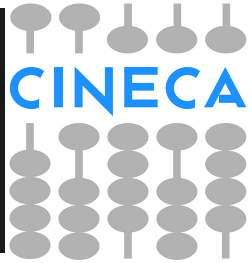
Example Header Lines

```
@HD    VN:1.0 SO:coordinate
@SQ    SN:1   LN:249250621  AS:NCBI37    UR:file:/data/local/ref/GATK/human_g1k_v37.fasta     M5:
1b22b98cdeb4a9304cb5d48026a85128
@SQ    SN:2   LN:243199373  AS:NCBI37    UR:file:/data/local/ref/GATK/human_g1k_v37.fasta     M5:
a0d9851da00400dec1098a9255ac712e
@SQ    SN:3   LN:198022430  AS:NCBI37    UR:file:/data/local/ref/GATK/human_g1k_v37.fasta     M5:
fdfd811849cc2fadebc929bb925902e5
@RG    ID:UM0098:1   PL:ILLUMINA   PU:HWUSI-EAS1707-615LHAAXX-L001   LB:80  DT:2010-05-05T20:00:00-0400       SM:SD37743
@RG    ID:UM0098:2   PL:ILLUMINA   PU:HWUSI-EAS1707-615LHAAXX-L002   LB:80  DT:2010-05-05T20:00:00-0400       SM:SD37743
```

Example Alignments

```
1:497:R:-272+13M17D24M     113    1      497    37     37M    15     100338662     0
CGGGTCTGACCTGAGGAGAACTGTGCTCCGCCTTCAG   0;==-==9;>>>>>=>>>>>>>>>>>=>>>>>>>>>>   XT:A:U NM:i:0 SM:i:37       AM:i:0 X0:i:1
X1:i:0 XM:i:0 XO:i:0 XG:i:0 MD:Z:37
19:20389:F:275+18M2D19M    99     1      17644  0      37M    =      17919  314    TATGACTGCTAATAATACCTACACATGTTAGAACCAT
>>>>>>>>>>>>>>>>>>>>>><<>>><<>>4::>>:<9   RG:Z:UM0098:1 XT:A:R NM:i:0 SM:i:0 AM:i:0 X0:i:4 X1:i:0 XM:i:0 XO:i:0 XG:i:0 MD:Z:37
19:20389:F:275+18M2D19M    147    1      17919  0      18M2D19M       =      17644  -314
GTAGTACCAACTGTAAGTCCTTATCTTCATACTTTGT   ;44999;499<8<8<<<8<<><<<<><7<;<<<>>><<   XT:A:R NM:i:2 SM:i:0 AM:i:0 X0:i:4 X1:i:0
XM:i:0 XO:i:1 XG:i:2 MD:Z:18^CA19
9:21597+10M2I25M:R:-209    83     1      21678  0      8M2I27M        =      21469  -244
CACCACATCACATATACCAAGCCTGGCTGTGTCTTCT   <;9<<5><<<<><<<><<><><><9>><<>>>9>>><>   XT:A:R NM:i:2 SM:i:0 AM:i:0 X0:i:5 X1:i:0
XM:i:0 XO:i:1 XG:i:2 MD:Z:35
```

NGS project



Write a **python Hadoop job**
which calculates coverage from a *SAM* file
for each available genomic position

Extra: count the single bases (A,C,T,G,N)

NGS project: skills

- **What you need**

a little python knowledge, linux experience, curiosity

- **What you learn**

python, Hadoop installation and comprehension, how to work on a real case scenario, bioinformatics problems

Thesis: working with us

Write a **python Hadoop daemon** to:

- distribute steps of bioinformatics pipeline
  - of a real bioinformatic service
- while tuning available cloud resources
  - based on OpenStack and Hadoop API

This is not the end...

...but the beginning!