Big Data: technologies, problems and solutions

An approach from High Performance Computing

Juan Carlos Pichel

Centro Singular de Investigación en Tecnoloxías da Información

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

citius.usc.es

Outline

- High Performance Computing (HPC)
  - Towards exascale computing: a brief history
  - Challenges in the exascale era
- Big Data meets HPC
  - Some facts about Big Data
  - Technologies
  - HPC and Big Data converging
- Case Studies:
  - Bioinformatics
  - Natural Language Processing (NLP)

CiTIUS

Outline

- **High Performance Computing (HPC)**
  - **Towards exascale computing: a brief history**
  - **Challenges in the exascale era**
- Big Data meets HPC
  - Some facts about Big Data
  - Technologies
  - HPC and Big Data converging
- Case Studies:
  - Bioinformatics
  - Natural Language Processing (NLP)

High Performance Computing (HPC)

Here we are

- Tianhe-2 (China)
  - ▷ 160,000 hybrid nodes
  - ▷ 3,120,000 cores
  - ▷ **55 PFLOPS** ($x10^{15}$ FLOPS)

  55.000.000.000.000.000 FLOPS
  - ▷ **17.8 MW**

- PlayStation 4 (PS4)
  - ▷ 1.8 TFLOPS

- Tianhe-2 = 30,400 PS4
- 180,000 light bulbs (100 W) or hydro-electric plant (Viana do Bolo, Ourense)

CiTIUS

High Performance Computing (HPC)

What types of big problem might require a “Big Computer”?

- **Compute Intensive**: A single problem requiring a large amount of computation
- **Memory Intensive**: A single problem requiring a large amount of memory
- **High Throughput**: Many unrelated problems to be executed over a long period
- **Data Intensive**: Operation on a large amount of data

CiTIUS

High Performance Computing (HPC)

What types of big problem might require a “Big Computer”?

- **Compute Intensive**:
  - Distribute the work across multiple CPUs to reduce the execution time as far as possible:
    - Each thread performs a part of the work on its own CPU, concurrently with the others
  - CPUs may need to exchange data rapidly, using specialized hardware
  - Large systems running multiple parallel jobs also need fast access to storage
  - Many use cases from Physics, Chemistry, Energy, Engineering, Astronomy, Biology...
  - The traditional domain of HPC and supercomputers

High Performance Computing (HPC)

What types of big problem might require a “Big Computer”?

- **Memory Intensive**:
  - Aggregate sufficient memory to enable solution at all
  - Technically more challenging if the program cannot be parallelized
- **High Throughput:**
  - Distribute work across multiple CPUs to reduce the overall execution time as far as possible
  - Workload is trivially (or embarrassingly) parallel
    - Workload breaks up naturally into independent pieces
    - Each piece is performed by a separate process on a separate CPU (concurrently)
  - Emphasis is on throughput over a period, rather than on performance on a single problem
  - Obviously a supercomputer can do this too

High Performance Computing (HPC)

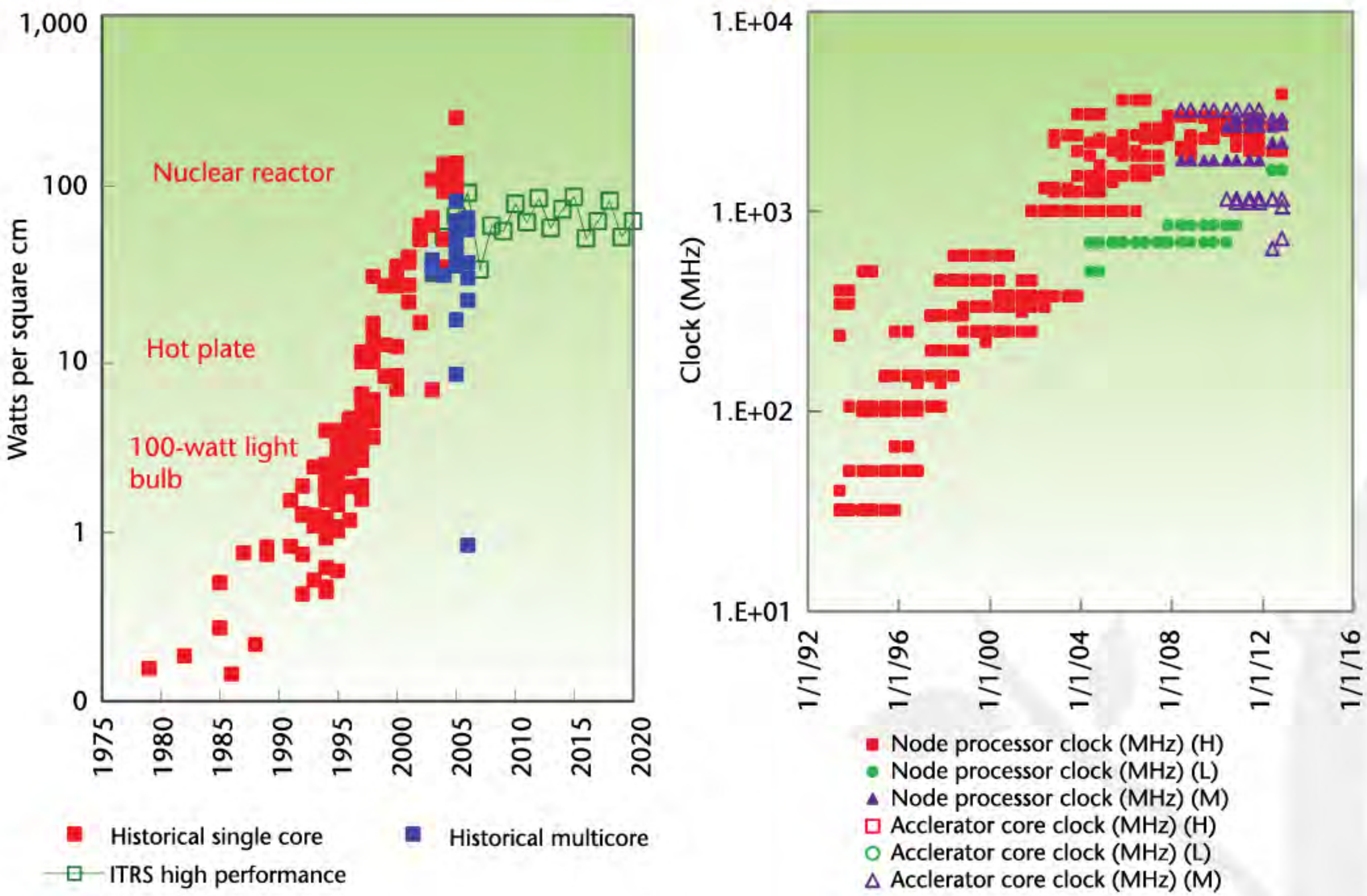
What types of big problem might require a “Big Computer”?

- **Data Intensive**:
  - Distribute the data across multiple CPUs to process in a reasonable time
  - Note that the same work may be done on each data segment
  - Rapid movement of data in and out of (disk) storage becomes important
  - **Big Data** and how to efficiently process it currently occupies much thought

CiTIUS

High Performance Computing (HPC)

A brief history: a technological storm

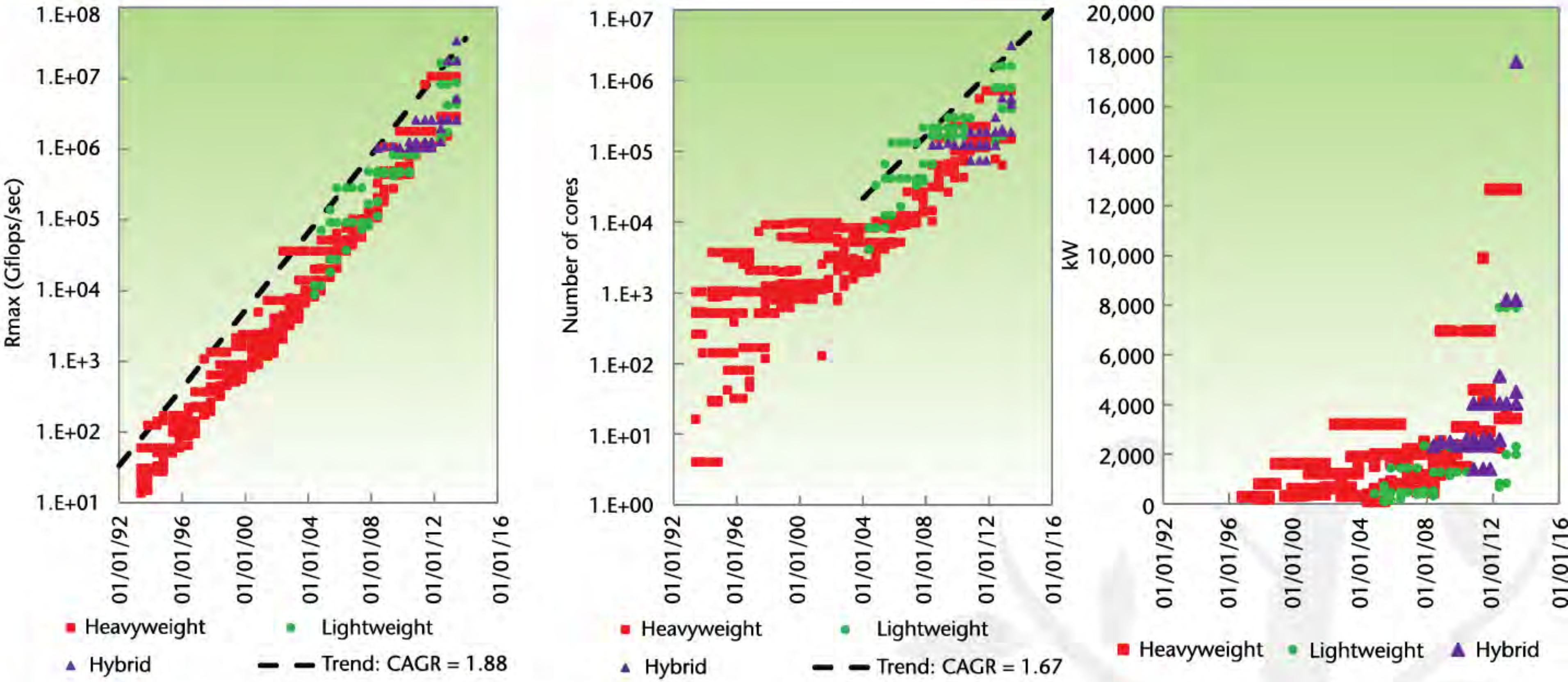


CiTIUS

P. Kogge and J. Shalf. "Exascale Computing Trends: Adjusting to the "new normal" for Computer Architecture", Computing in Science & Engineering 2013

High Performance Computing (HPC)

A brief history: today's architectures

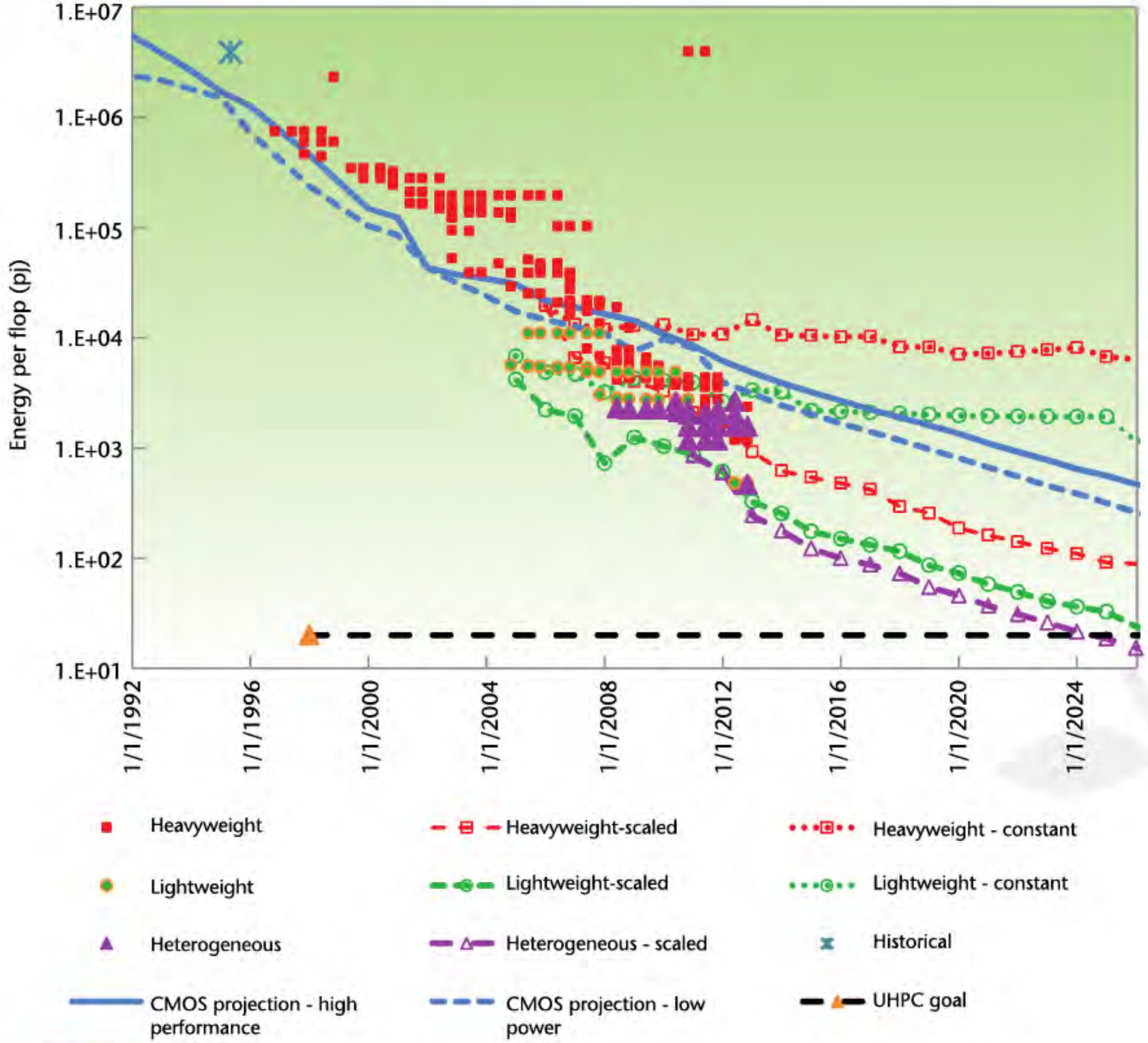


- Heavyweight: High-end multicore chips
- Lightweight: Much simpler cores, much lower clock rate
- Hybrid: Heterogeneous Architectures

P. Kogge and J. Shalf. "Exascale Computing Trends: Adjusting to the "new normal" for Computer Architecture", Computing in Science & Engineering 2013

High Performance Computing (HPC)

A brief history: it's the end of the world as we know it



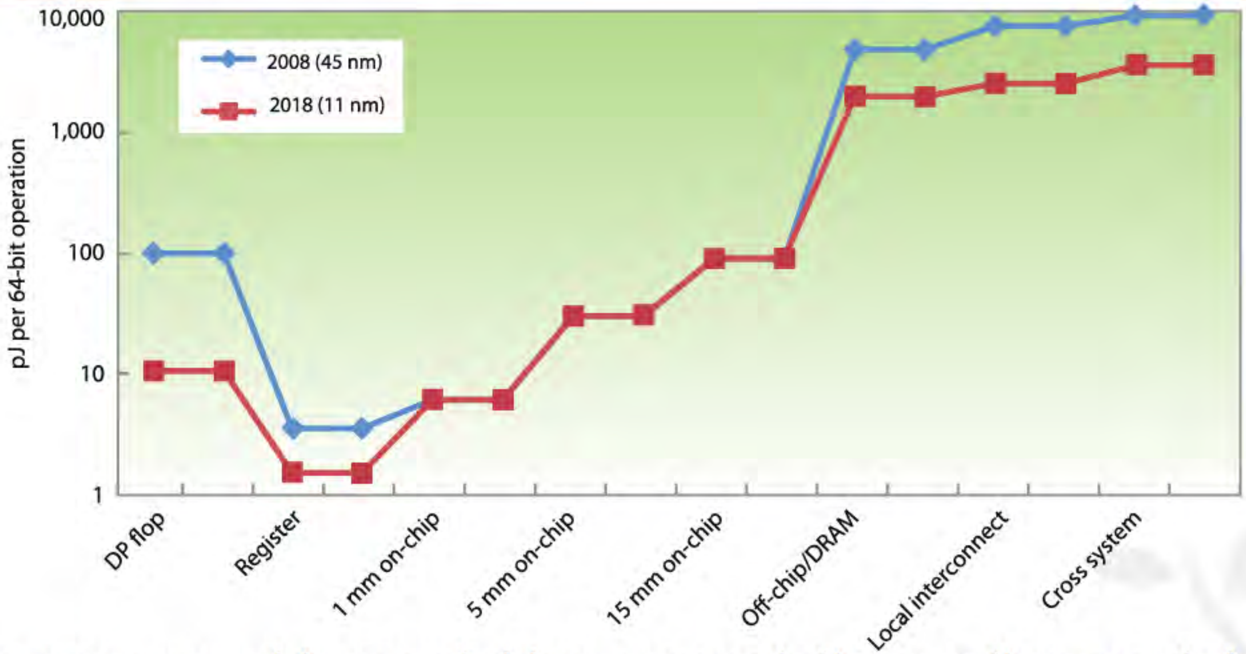
- 20 pJ/Flop to have a system power less than 20 MW
- Only *hybrids* have a chance by 2024 (Inefficient!!!)

CiTIUS

P. Kogge and J. Shalf. "Exascale Computing Trends: Adjusting to the "new normal" for Computer Architecture", Computing in Science & Engineering 2013

High Performance Computing (HPC)

Challenges in the **exascale** era



- Data movement is overtaking computation as the most dominant cost (money and energy consumption)
- Current parallel programming models (MPI, OpenMP, etc…): **computer-centric**
- Focus on equal partitioning computation, implicitly moving data to PEs
- **Big Data era: we have to move to data-centric models!!!**

CiTIUS

P. Kogge and J. Shalf. "Exascale Computing Trends: Adjusting to the "new normal" for Computer Architecture", Computing in Science & Engineering 2013

High Performance Computing (HPC)

Challenges in the **exascale** era

- **FLOPS** is not *on command*
- Once upon a time… when FPU was the most expensive and precious resource in a supercomputer
- Metrics: FLOPS, FLOPS and FLOPS
- But Data movement's energy efficiency isn't imporving as fast as Flop's energy efficiency
- **Algorithm designer should be thinking in terms of wasting the inexpensive resource (flops) to reduce data movement**
- Communication-avoiding algorithms

High Performance Computing (HPC)

Challenges in the **exascale** era

Old constraints	New constraints
Clock frequency: provide performance scaling for each generation	**Clock frequency**: not increasing; all performance scaling from parallelism
Power: not a significant concern	**Power**: primary design constraint for future high-performance computing (HPC) systems
Cost: flops are biggest cost for system, so optimize for compute	**Cost**: data movement dominates, so optimize to minimize data movement
Concurrency: modest growth of parallelism by adding nodes	**Concurrency**: exponential growth of parallelism within chips
Memory scaling: maintain byte per flop capacity and bandwidth	**Memory scaling**: compute growing 2× faster than capacity or bandwidth
Locality: Message Passing Interface at exascale (MPI+X) model (uniform costs within node and between nodes)	**Locality**: must reason about data locality and possibly topology
Uniformity: assume uniform system performance	**Heterogeneity**: architectural and performance nonuniformity increase
Reliability: it's the hardware's problem	**Reliability**: can't count on hardware protection alone

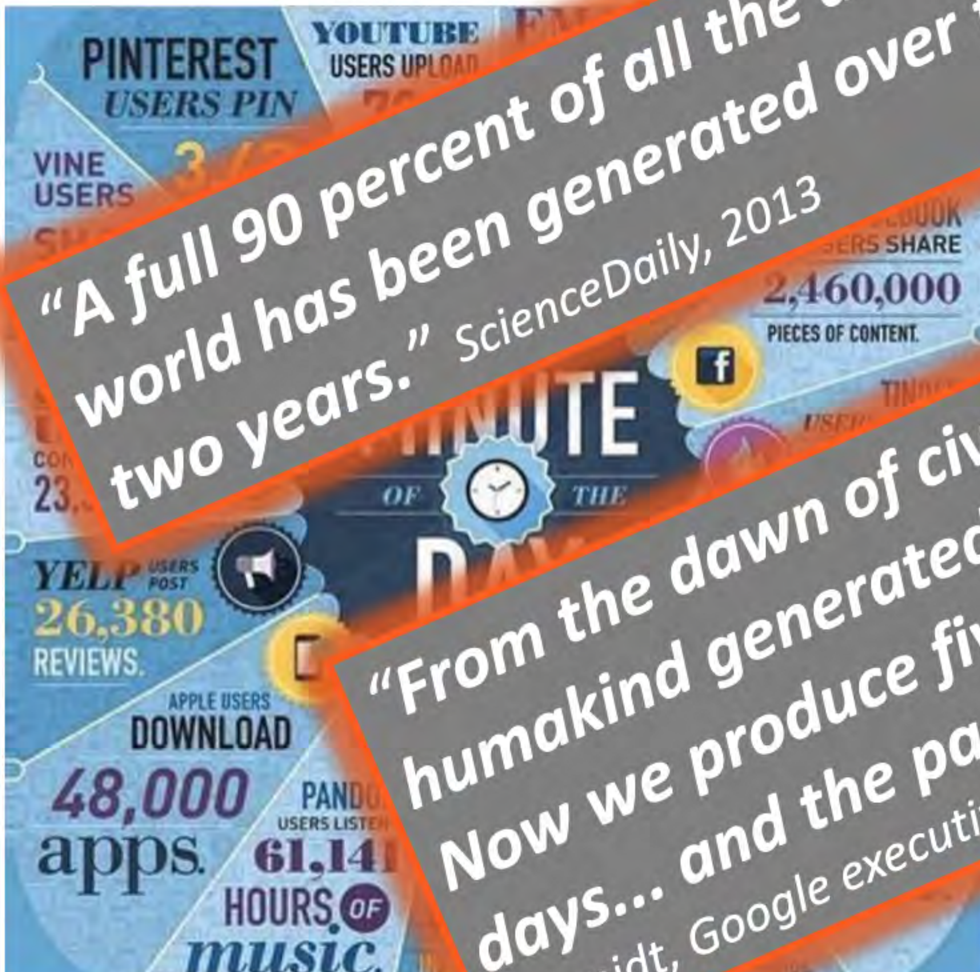
P. Kogge and J. Shalf. "Exascale Computing Trends: Adjusting to the "new normal" for Computer Architecture", Computing in Science & Engineering 2013

Outline

- High Performance Computing (HPC)
  - Towards exascale computing: a brief history
  - Challenges in the exascale era
- **Big Data meets HPC**
  - **Some facts about Big Data**
  - **Technologies**
  - **HPC and Big Data converging**
- Case Studies:
  - Bioinformatics
  - Natural Language Processing (NLP)

Big Data meets HPC

Some facts about Big Data



"A full 90 percent of all the data in the world has been generated over the last two years." ScienceDaily, 2013

- Sensor netw... (smart cities)

"From the dawn of civilization until 2003, humakind generated five exabytes of data. Now we produce five exabytes every two days... and the pace is accelerating." Eric Schmidt, Google executive.

Big Data meets HPC

Some facts about Big Data

- Data in 2013: **4.4 Zettabytes** (4.4 x 10^{21} bytes)
- Estimation in 2020: **44 Zettabytes**

Searching for one element in a 1 MB file: < 0.1 seconds

Searching for one element in a 1 GB file: a few minutes

Searching for one element in a 1 TB file: about 30 hours

Searching for one element in a 1 PB file: > 3 years

Searching for one element in a 1 EB file: 30 centuries

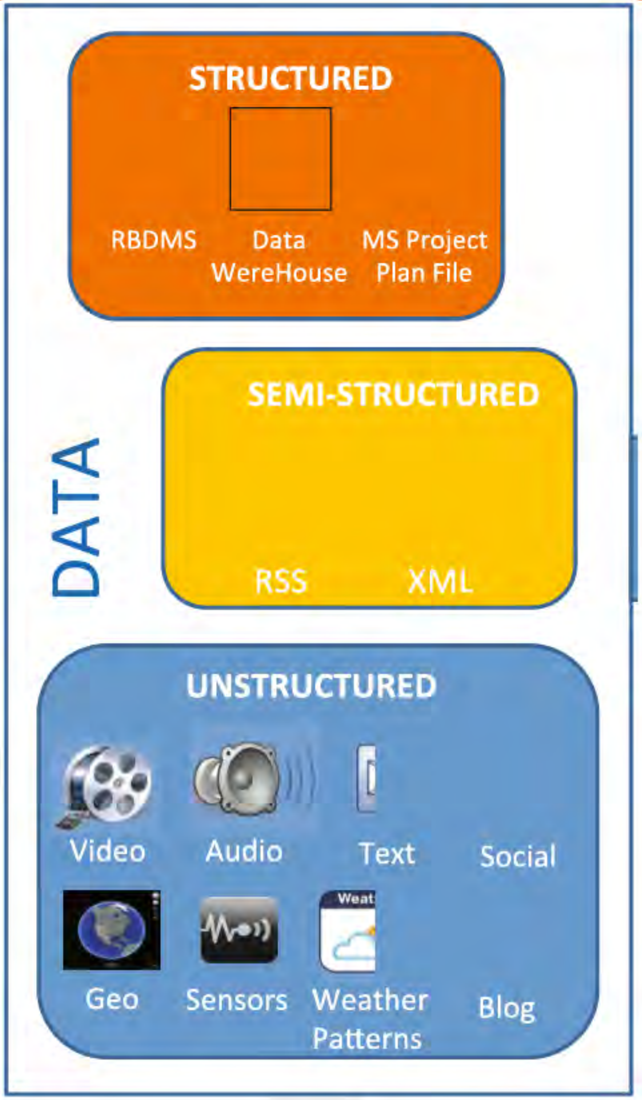
Searching for one element in a 1 ZB file: 3,000 millennium

Estimation using a PC

Big Data meets HPC

Some facts about Big Data

- **V** is the letter…
  - **Volume**: Big Data are huge in quantity, but how much?
  - **Velocity**: more data implies increased speed in accessing, transmitting and processing (new technological and architectural solutions)
  - **Variety**: large number of sources mean wide variety of formats (structured, unstructured and semi-structured)
  - **Verification**: of the quality and compliance with rules
  - **Value**: main goal, generate value from the data
  - **Maybe you can find your own V. Please fill here:**



Big Data meets HPC

Technologies: how to process all these data

- Illumina HiSeqX™ Ten:
  - Human genome sequencing
  - Up to 6 billion reads (6×10^9 reads)
  - Thousands of GB of data
- Burrows-Wheeler Aligner (BWA)
  - Very popular software for mapping sequence reads
  - Sequence alignment is a very time consuming process

Sequential processing: > 40 days!!!!!

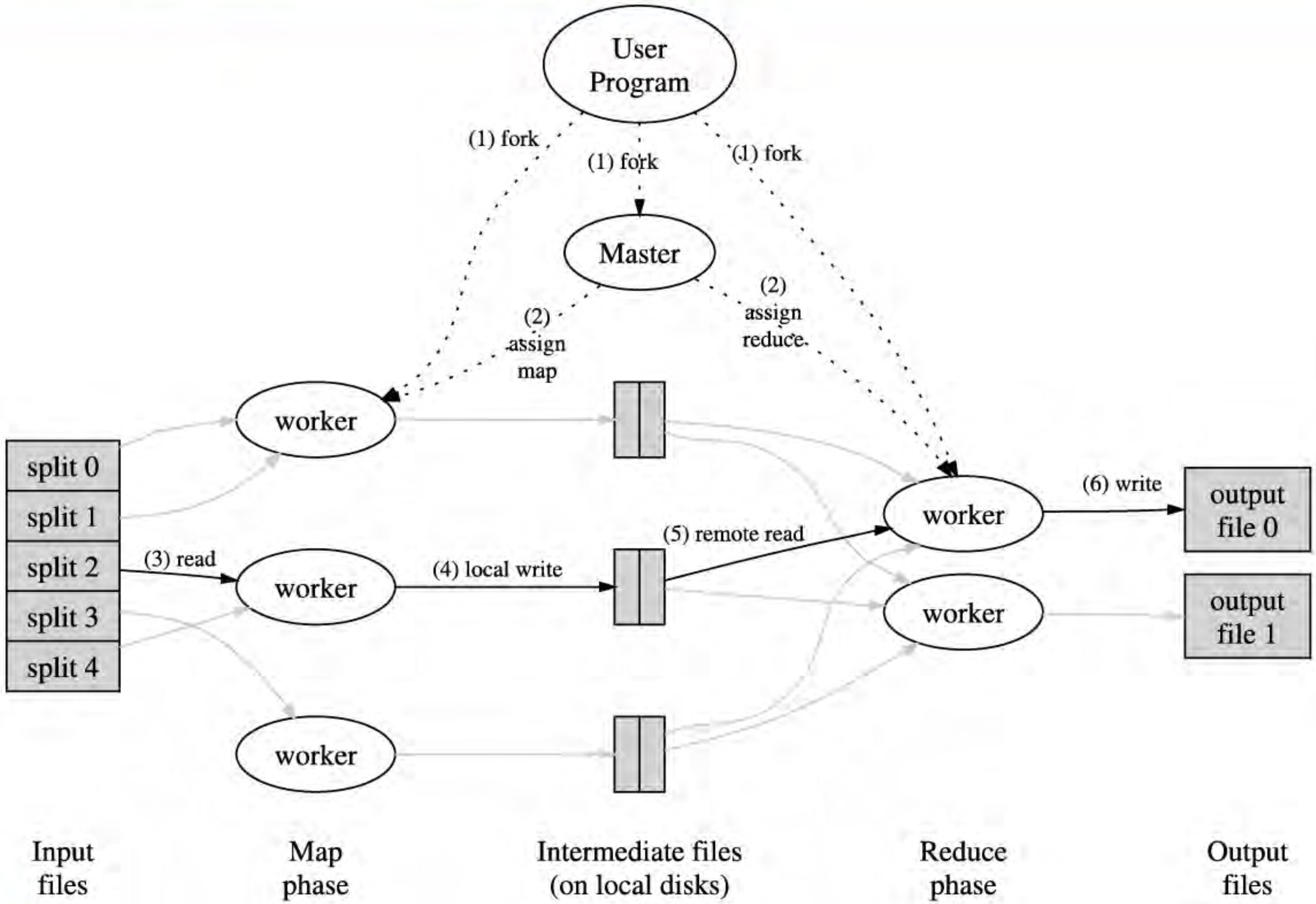
CiTIUS

Big Data meets HPC

Technologies: how to process all these data

- **Map/Reduce paradigm**
- Introduced by Dean and Ghemawat (Google, 2004)
- As simple as providing:
  - MAP function that processes a key/value pair to generate a set of intermediate key/value pairs
  - REDUCE function that merges all intermediate values associated with the same intermediate key
- Runtime takes care of:
  - Partitioning the input data (**Parallelism**)
  - Scheduling the program's execution across a set of machines (**Parallelism**)
  - Handling machine failures
  - Managing inter-machines communication (**Parallelism**)

CiTIUS



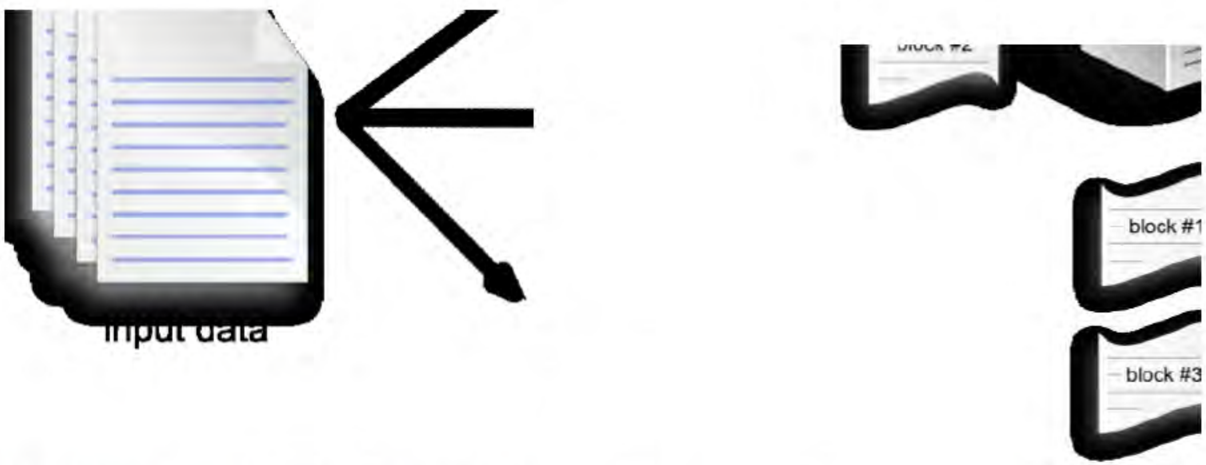
J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters" , OSDI 2004

Big Data meets HPC

Technologies: Apache Hadoop

- **Apache Hadoop** is an open-source implementation of the Map/Reduce paradigm
- It's a framework for large-scale data processing
- It is designed to run on cheap commodity hardware
- It automatically handles data replication and node failure
- Hadoop provides:
  - API+implementation for working with Map/Reduce
  - Job configuration and efficient scheduling
  - Browser-based monitoring of important cluster stats
  - A distributed filesystem optimized for HUGE amounts of data (HDFS)

Big Data meets



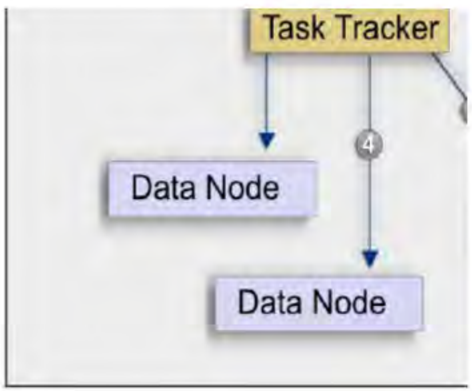
ata copied into HDFS is split i
ypical block size: UNIX = 4KB

- Data copied into HDFS is split into blocks
- Each data block is replicated to multiple machines
- Allows for node failure without data loss

- Each data blocks is replicated to multiple machines
- Allows for node failure without data loss

CiTIUS

Big Data meets HPC



- ıe **Node**: it manages the ystem (only one active)
- of **Data Nodes**: ıage data blocks and d them to clients
- **Tracker**: receives job ıests, schedules and ıitors MR jobs on Task ckers
- k **Tracker**: execute MR ration, read blocks from a Nodes

CiTIUS

- **Pros**:
  - Write here all the advantages commented previously
  - Hadoop it's written in Java but allows to execute codes from different programming languages (*Hadoop Streaming*)
  - Hadoop Ecosystem (Pig, Hive, HBase, etc…)

- **Cons:**
  - The problem must fit the Map/Reduce paradigm (embarrassingly parallel problems)
  - Bad for iterative applications
  - Important degradations in performance when using *Hadoop Streaming* (i.e. when codes are written in languages as Fortran, C, Python, Perl, etc.)
  - Intermediate results output is always stored on disks (In-Memory MapReduce – IMMR)
  - No reuse computation for jobs with similar input data:
    - For example, job runs everyday to find the most frequently read news over the past week
  - *Hadoop was not-designed by HPC people* (joke!!!)

Big Data meets HPC

Technologies: Apache Spark

- **Apache Spark** is an open source project
- It starts as a research project in Berkeley
- Cluster computing framework designed to be *fast* and *general-purpose*

- **Pros (I):**
  - Extends the Map/Reduce paradigm to support more types of computations (interactive queries and stream processing)
  - APIs in Python, Java and Scala
  - Spark has the ability to run computations in memory (Resilient Distributed Datasets – RDDs)

Big Data meets HPC

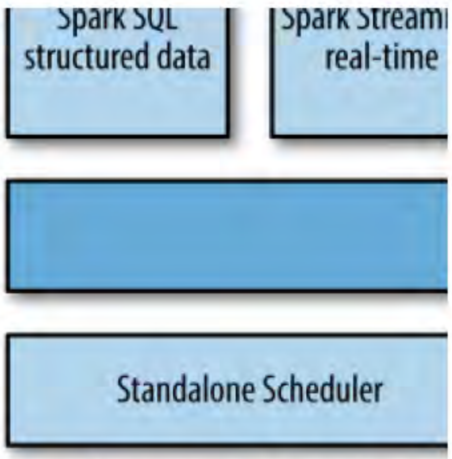
Technologies: Apache Spark

- **Pros (II):**
  - It supports different workloads in the same engine:
    - Batch applications
    - Iterative algorithms
    - Streaming and iterative queries
  - Good integration with Hadoop

- **Cons:**
  - Memory requirements

Big Data meets HPC

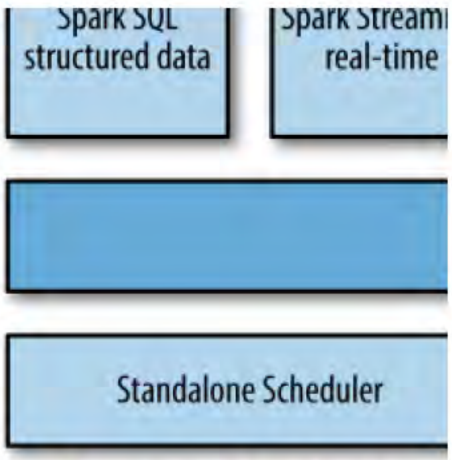
Technologies: Apache Spark



- **Spark Core**: similar architecture than Hadoop. Components for task scheduling, memory management, fault recovery, interacting with storage systems
- **SparkSQL**: package for working with structured data (querying via SQL or Apache Hive). Supports many sources of data.
- **SparkStreaming**: enables processing of live streams of data
- **MLib**: library containing Machine Learning algorithms for classification, regression, clustering, etc.

Big Data meets HPC

Technologies: Apache Spark



- **GraphX:** library for manipulating graphs (e.g. a social network's friends graph) and performing graph-parallel computations
- **Cluster Managers**: Spark is designed to scale up from one to thousands of compute nodes. It can run on the top of several cluster managers: Hadoop YARN, Apache Mesos and a simple cluster manager.

CiTIUS

- **Apache Flink**
  - It's an European project
  - Functionalities very similar to those explained for Spark

- If you like **R**, try this:
  - **RHIPE**
    - Released as R package
    - Map and Reduce functions as R code
  - **Big R** (O. D. Lara *et al.* "Big R_ Large-scale Analytics on Hadoop Using R", IEEE Int. Congress on Big Data, 2014)
    - It hides the Map/Reduce details to the programmer

Big Data meets HPC

Other Technologies

RHIPE

```
rhinit(TRUE, TRUE)
# Output from map is:
# "CARRIER|YEAR|MONTH \t DEPARTURE_DELAY"
map <- expression({
    # For each input record, parse out required fields and output new record:
    extractDeptDelays = function(line) {
        fields <- unlist(strsplit(line, "\\,"))
        # Skip header lines and bad records:
        if (!(identical(fields[[1]], "Year")) & length(fields) == 29) {
            deptDelay <- fields[[16]]
            # Skip records where departure dalay is "NA":
            if (!(identical(deptDelay, "NA"))) {
                # field[9] is carrier, field[1] is year, field[2] is month:
                rhcollect(paste(fields[[9]], "|", fields[[1]], "|", fields[[2]],
                    sep=""),
                deptDelay)
            }
        }
    }
    # Process each record in map input:
    lapply(map.values, extractDeptDelays)
})
# Output from reduce is:
# YEAR \t MONTH \t RECORD_COUNT \t AIRLINE \t AVG_DEPT_DELAY
reduce <- expression(
    pre = {
        delays <- numeric(0)
    },
    reduce = {
        # Depending on size of input, reduce will get called multiple times
        # for each key, so accumulate intermediate values in delays vector:
        delays <- c(delays, as.numeric(reduce.values))
    },
    post = {
        # Process all the intermediate values for key:
        keySplit <- unlist(strsplit(reduce.key, "\\|"))
        count <- length(delays)
        avg <- mean(delays)
        rhcollect(keySplit[[2]],
        paste(keySplit[[3]], count, keySplit[[1]], avg, sep="\t"))
    }
)
inputPath <- "/data/airline/"
outputPath <- "/dept-delay-month"
# Create job object:
z <- rhmr(map=map, reduce=reduce,
    ifolder=inputPath, ofolder=outputPath,
    inout=c('text', 'text'), jobname='Avg Departure Delay By Month',
    mapred=list(mapred.reduce.tasks=2))
# Run it:
rhex(z)
```

Big R

```
> air <- bigr.frame(dataPath = "airline.csv",
                    dataSource = "DEL", na.string="NA")
> summary(mean(DepDelay) ~ UniqueCarrier + Year + Month,
          dataset = air)
```

-- Dataset contains more than 20 years of flight/arrival information (USA)

-- We are interested in computing the average mean departure delay for each airline on a monthly basis

Big Data meets HPC

HPC and Big Data converging

- Those technologies were designed to run on "cheap" commodity clusters, but…
- … there is more to Big Data than large amounts of information
- It also related to **massive distributed activities such as complex queries and computation** (analytics or data-intensive scientific computing)
- **High Performance Data Analytics (HPDA)**

CiTIUS

Big Data meets HPC

HPC and Big Data converging

- **Infiniband**:
  - It's the standard interconnect technology used in HPC supercomputers
  - Commodity clusters use 1Gbps or 10Gbps ethernet
  - Hadoop is very network-intensive (e.g. Data Nodes and Task Trackers exchange a lot of information)
  - 56Gbps FDR can be 100x faster than 10 Gbps ethernet due to its superior **bandwidth and latency**
  - It allows to scale the big data platform to the desired size, without worrying about bottlenecks

Big Data meets HPC

HPC and Big Data converging

- **Accelerators**:
  - Hadoop and Spark only work on homogeneous clusters (CPUs)
  - HPC is moving to heterogeneous platforms consisting of nodes which include GPUs, co-processors (Intel Xeon Phi) and/or FPGAs.
  - Most promising systems to reach the exaflop
  - Accelerator can boost the performance (**not all applications fit well)**
  - For complex analytics or data-intensive scientific computing
  - A lot of interest: *HadoopCL, Glasswing, GPMR, MapCG, MrPhi, etc…*

Outline

- High Performance Computing (HPC)
  - Towards exascale computing: a brief history
  - Challenges in the exascale era
- Big Data meets HPC
  - Some facts about Big Data
  - Technologies
  - HPC and Big Data converging
- **Case Studies:**
  - **Bioinformatics**
  - **Natural Language Processing (NLP)**

CiTIUS

- **Burrows-Wheeler aligner (BWA)**
- Mapping sequence reads to a large reference genome
- Dataset size: about 500 GB

- **BigBWA** (https://github.com/citiususc/BigBWA)**:**
  - Alignment process is performed in parallel
  - It's fault tolerant
  - **No modifications to BWA source code are required**

J. M. Abuín, Juan C. Pichel, Tomás F. Pena and Jorge Amigo. "BigBWA: Approaching the Burrows-Wheeler Aligner to Big Data Technologies", Bioinformatics, 2015

Sequential processing: > 40 days

Server (BWA): 5 days

Small Cluster (BigBWA): > 1 day

Small HPC cluster (BigBWA): 5 hours

J. M. Abuín, Juan C. Pichel, Tomás F. Pena and Jorge Amigo. “BigBWA: Approaching the Burrows-Wheeler Aligner to Big Data Technologies”, Bioinformatics, 2015

Outline

Natural Language Processing

- NLP suited to structure and organize the textual information accessible through Internet
- Linguistic processing is a complex task that requires the use of several subtasks organized in interconnected modules
- Most of the existent NLP **modules are programmed using Perl** (regular expressions)
- We have integrated into Hadoop three NLP modules (*Hadoop Streaming*):
  - *Named Entity Recognition (NER)*: It consists of identifying as a single unit (or token) those words or chains of words denoting an entity, e.g. *New York*, *University of San Diego*, *Herbert von Karajan*, etc.
  - *PoS-Tagging*: It assigns each token of the input text a single PoS tag provided with morphological information e.g. *singular and masculine adjective*, *past participle verb*, etc.
  - *Named Entity Classification (NEC)*: It is the process of classifying entities by means of classes such as "People", "Organizations", "Locations", or "Miscellaneous".

J. M. Abuín, Juan C. Pichel, Tomás F. Pena, P. Gamallo and M. García. "Perldoop: Efficient Execution of Perl Scripts on Hadoop Clusters", Big Data Conference, 2014

- HPC people (including myself) are “obsessed” with performance… Flops, Flops and Flops
- Hadoop Streaming is really nice, but it shows an important degradation in performance (w.r.t. Java codes)

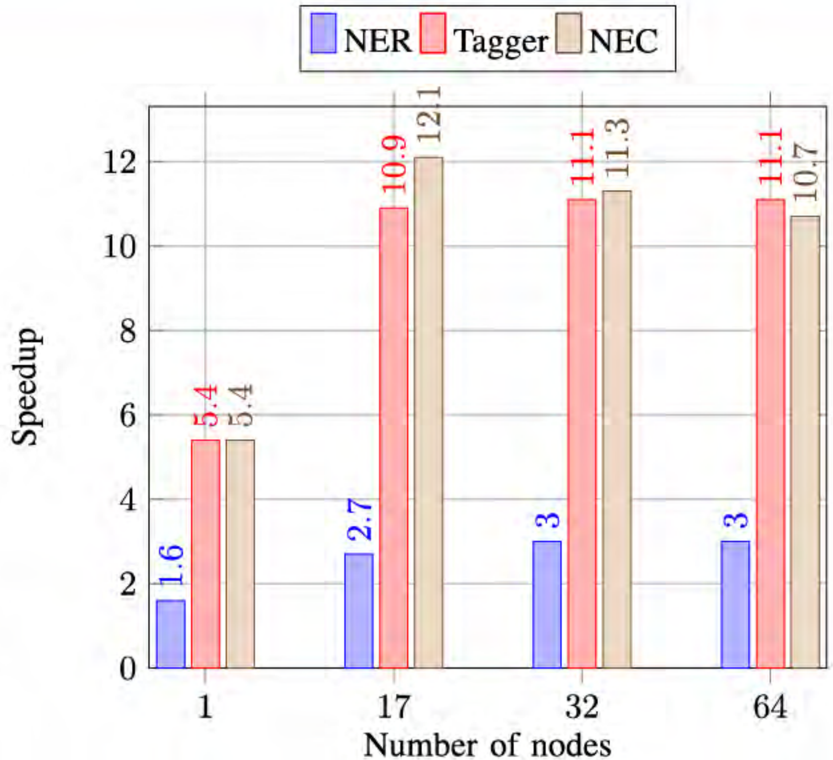
- **Perldoop:** we designed an automatic source-to-source translator Perl to Java:
  - It’s not general-purpose
  - Perl scripts should be in Map/Reduce format
  - Perl codes should follow some simple programming rules

J. M. Abuín, Juan C. Pichel, Tomás F. Pena, P. Gamallo and M. García. “Perldoop: Efficient Execution of Perl Scripts on Hadoop Clusters”, Big Data Conference, 2014

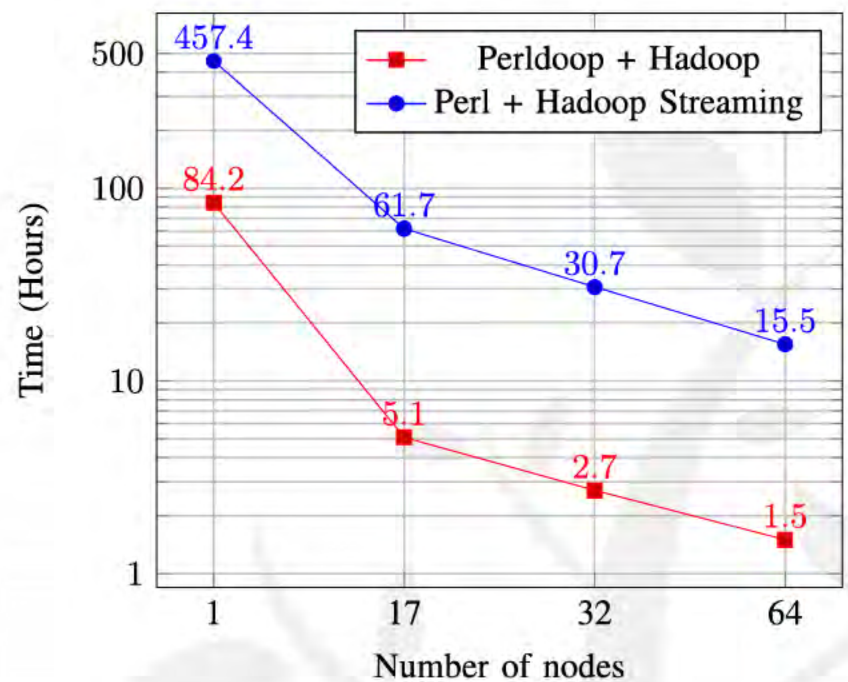
```perl
#!/usr/bin/perl -w

#<perl><start>

my $line;                  #<var><string>
my @words;                 #<array><string>
my $key;                   #<var><string>
my $valueNum = "1";        #<var><string>
my $val;                   #<var><string>

while ($line = <STDIN>) {        #<map>
  chomp ($line);
  @words = split ("␣",$line);
  foreach my $w (@words) {       #<var><string>
    $key = $w."\t";
    $val = $valueNum."\n";
    print $key.$val;
  }
}

#<perl><end>
```

```java
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public static class WordCountMap extends Mapper<Object, Text, Text, Text>{
  @Override
  public void map(Object incomingKey, Text value,
  Context context) throws IOException, InterruptedException {
    try{

      //<java><start>
      String line;
      String[] words;
      String key;
      String valueNum  = "1";
      String val;
      line  = value.toString();
      line = line.trim();
      words  = line.split("␣");
      for (String w : words) {
        key  = w+;
        val  = valueNum;
        context.write(new Text(key),new Text(val));
      }
      //<java><end>

    }
    catch(Exception e){
      System.out.println(e.toString());
    }
  }
}
```

Word Count (Mapper)



Hadoop vs. Hadoop Streaming



NEC

Dataset: Wikipedia (Spanish)

1 node = 1 core

J. M. Abuín, Juan C. Pichel, Tomás F. Pena, P. Gamallo and M. García. "Perldoop: Efficient Execution of Perl Scripts on Hadoop Clusters", Big Data Conference, 2014

Thank you!

juancarlos.pichel@usc.es

citius.usc.es

CiTIUS