Approaching Big Data technologies to scientific applications

Doctoral Meeting

José Manuel Abuín Mosquera

Centro Singular de Investigación en Tecnoloxías da Información

Universidade de Santiago de Compostela

BigData4Science

citius.usc.es

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

CAMPUS VIDA CAMPUS DE EXCELENCIA INTERNACIONAL

CiTiUS Centro Singular de Investigación en Tecnoloxías da Información

Contents

Contents

What is Big Data?

- It is estimated that each day are created around **2.5 quintillion** bytes of data (2.5 Exabytes).
- Companies want to **process,understand and obtain conclusions** from all these data.
- A few years ago, the **MapReduce programming model** was introduced.
- It is a **parallel** model which uses a **distributed file system.**

The HDFS

- **HDFS** stands for Hadoop Distributed File System.
- It is a distributed system that stores a **block** in one or several nodes with a given **replication** factor.

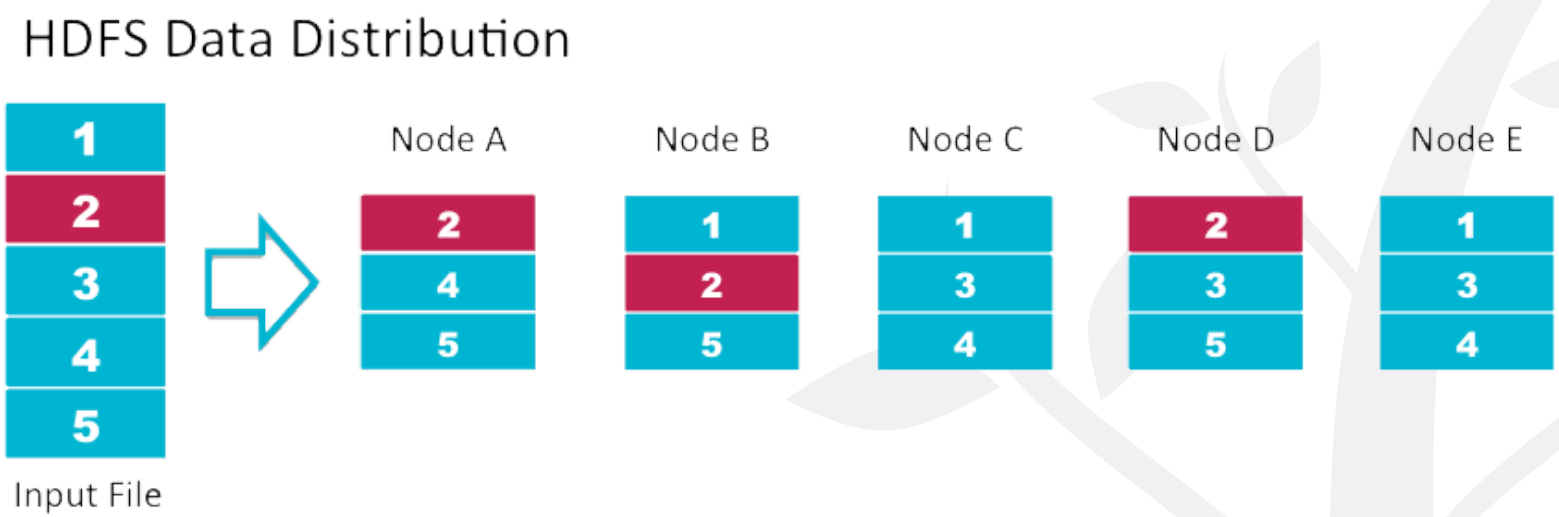


Figure: HDFS Distribution. *(http://www.cloudera.com)*

The MapReduce programming model

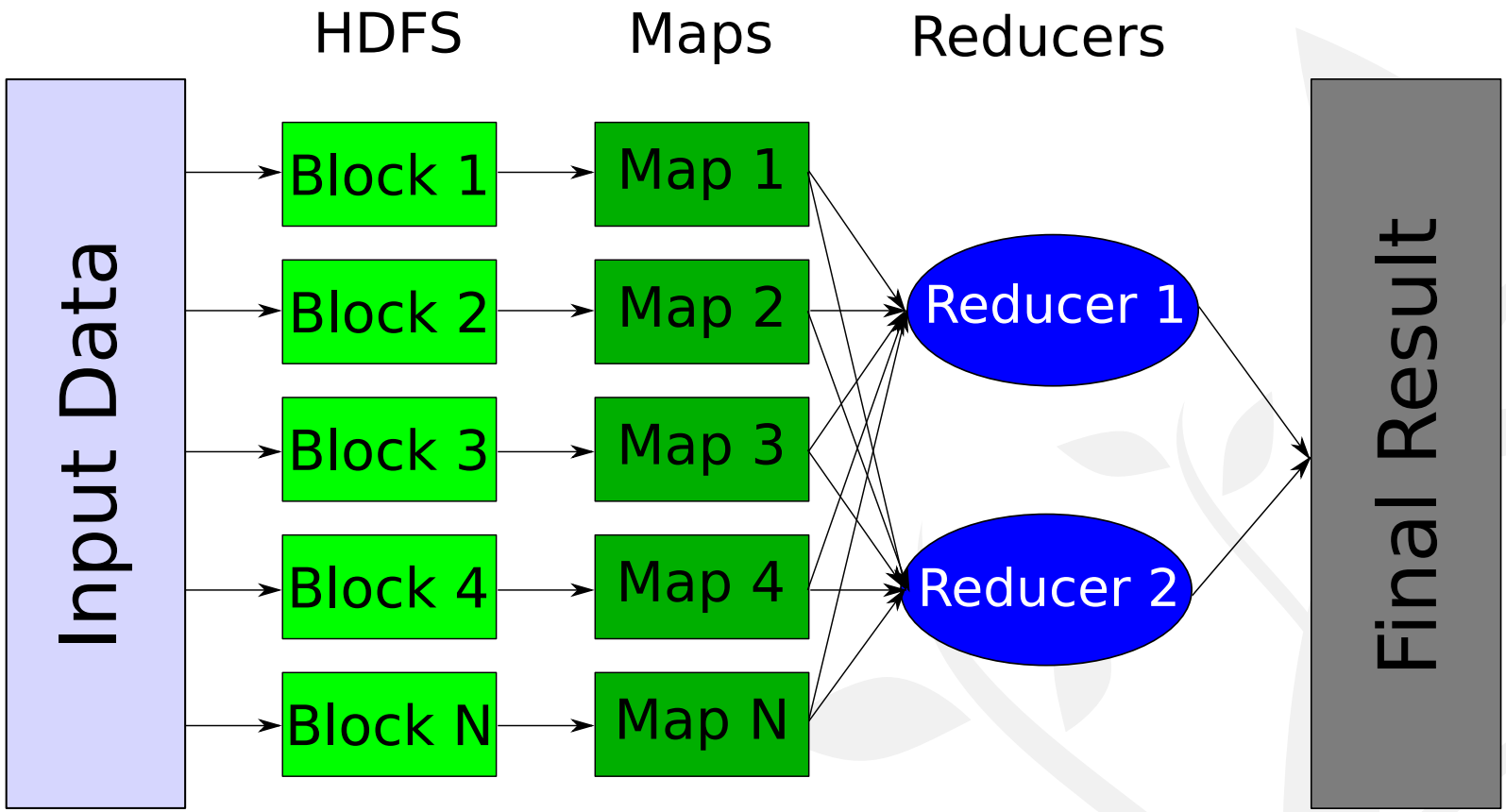


Figure: MapReduce model.

Apache Hadoop

- One of the most famous **Big Data frameworks.**
- Based on **Google's** MapReduce programming model.
- Implemented in **Java**, but its core is in **C** language.
- **Problem:** It support **iterative processes,** but the user has to write **intermediary results to disk**, with the consequent lack of performance.

Apache Spark

- Open-source cluster computing framework.
- It provides **in-memory primitives**, by loading data into a cluster's memory and query it repeatedly,
- Allow us to **chain several maps**, reducers and more operations.
- It can works **together with Hadoop** in the same cluster using **YARN**.
- **100**× **faster** than Hadoop on memory and **10**× **faster** on disk.
- Resilient Distributed Dataset **(RDD)**. Fault-tolerant collection of elements that can be operated on in parallel. **In memory**, in disk or both.

Motivation

- Is it possible to use this paradigm for **classic HPC scientific computations?.**
- Which are the **scientific fields** more suitable to this paradigm?.
- Is this paradigm better than the **classic HPC** approach?.

Contents

Sequence alignment in Genomics

Main research efforts in bioinformatics

- **Sequence alignment**.
- Gene prediction.
- Structural alignment of proteins.

Alignment

Comparison of two or more nucleotide or protein sequences to determine the **degree of similarity**. Commonly used to deduct **functional or evolutionary** relationships between genes and proteins.

DNA Index

Human DNA Index

- We need and **index** to align against our reads.
- There are several formats, but the most common is **FASTA.**
- The ***human_g1k_v37*** index has 3 GB of size in this format (2009).

```
ACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTA
ACCCTAACCCTAACCCTAACCCTAACCCAACCCTAACCCTAACCCTAACCCTAACCCTAA
CCCTAACCCCTAACCCTAACCCTAACCCTAACCCTAACCTAACCCTAACCCTAACCCTAA
CCCTAACCCTAACCCTAACCCTAACCCTAACCCCTAACCCTAACCCTAAACCCTAAACCC
```

Figure: Human DNA index example.

Ultra Sequentiation

Formal definition

Process of determining the precise order of **nucleotides** within a DNA molecule.

. . . and this means?

1. Tissue **sample**.
2. Put the sample into an ultra sequentiation **machine**.
3. You get **reads** from this sample and store them into your **computer**.

How a sample read looks like?

Human DNA sample

- One of the most common formats is the **FASTQ.**
- In these case we can reach **TeraBytes** of data.
- Each **four lines** is a record, and the **second line** in a record is the read.

```
@SRR062634.1 HWI-EAS110_103327062:6:1:1092:8469/1
GGGTTTTCCTGAAAAAGGGATTCAAGAAAGAAAACTTACATGAGGTGATTGTTTAATGTTGCTACCAAAGAAGAGAGAGTTACCTGCCCATTCACTCAGG
+
@C'@9:BB:?DCCB5CC?5C=?5@CADC?BDB)B@?-A@=:=:@CC'C>5AA+*+2@@'-?>5-?C=@-??)'>>B?D@?*?A##################
@SRR062634.2 HWI-EAS110_103327062:6:1:1107:21105/1
ACCGTGAGCAATCAGCTGCCATCAACGTGGAGGTAAGACTCTCCACCTGCAAAAACATTACAACTTGCTGAAGGCTGAGATACTTGTTCGCACATTTTTA
+
FDEFF?DFEFE?BEEEEED=DB:DCEAEEB,CC=@B=5?B?CC5C?B+A??=>:CC<9-B2=@>-?:-<A@@A?9>*0<:'0%6,>:9&-:>?:>==B??
```

Figure: Human DNA sample read.

What that letters means?

Alphabets

- $\sum_{DNA} = \{A, C, G, T\}$
- $\sum_{RNA} = \{A, C, G, U\}$
- $\sum_{PRO} = \{A, C, D, E, F, G, H, I, L, K, M, N, P, Q, R, S, T, V, W, Y\}$

DNA case, nitrogen bases

- **A** - Adenine
- **C** - Cytosine
- **G** - Guanine
- **T** - Thymine

How we perform the alignment

- **Burrows - Wheeler Aligner (BWA).**
- Open Source. Available on **GitHub.**
- Implemented in **C** language.
- One of the **widely accepted** software to perform the alignment in the computational genomics **community**.
- It implements three algorithms:
  - **BWA-backtrack**
  - BWA-SW
  - **BWA-MEM**

How we perform the alignment

We use BWA to perform the alignment. But we have to integrate it into a Big Data environment. For this reason we developed the **BigBWA** tool *(https://github.com/citiususc/BigBWA).*

BigBWA - I

1. We **don't need to modify** the BWA code, we create a **wrapper** which calls BWA functions from Java by using **JNI**.
2. We need to make the **index available to all the computing nodes.**
3. We distribute the **input data** (FASTQ reads) across our **maps** using HDFS.

How we perform the alignment

BigBWA - II

4. Each **map** is executed in **one** core from our cluster.
5. In this way, we **parallelize** the execution of **BWA** across the number of cores/maps.
6. In the **reduce** phase we put all the **outputs together** in one file (this is **optional**) .

How we perform the alignment

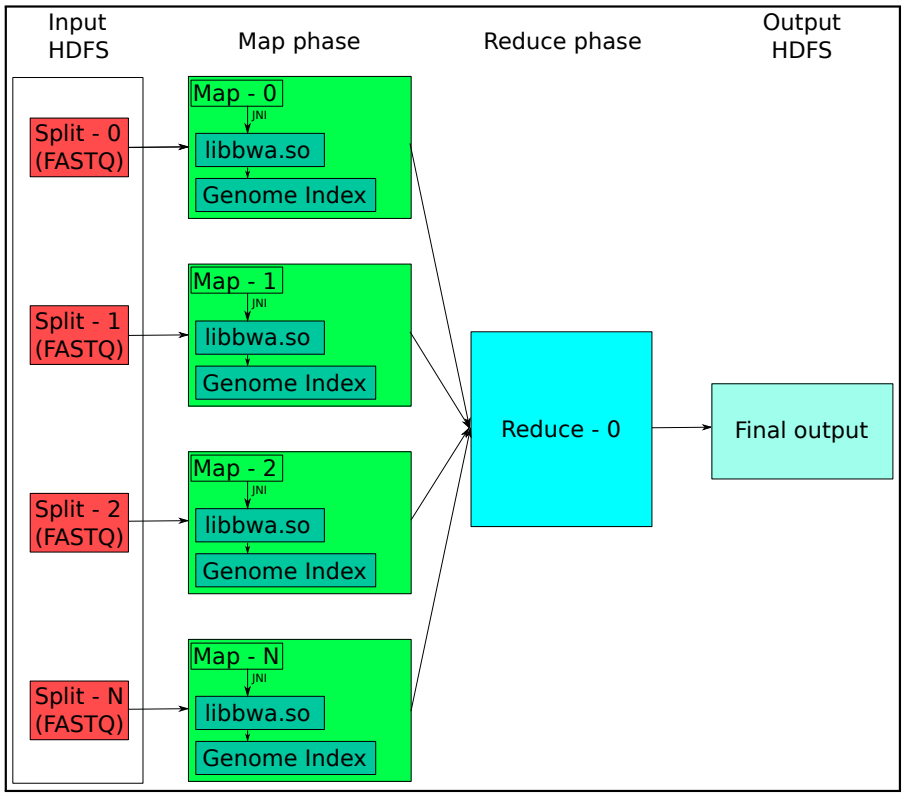


Figure: Model with reducer

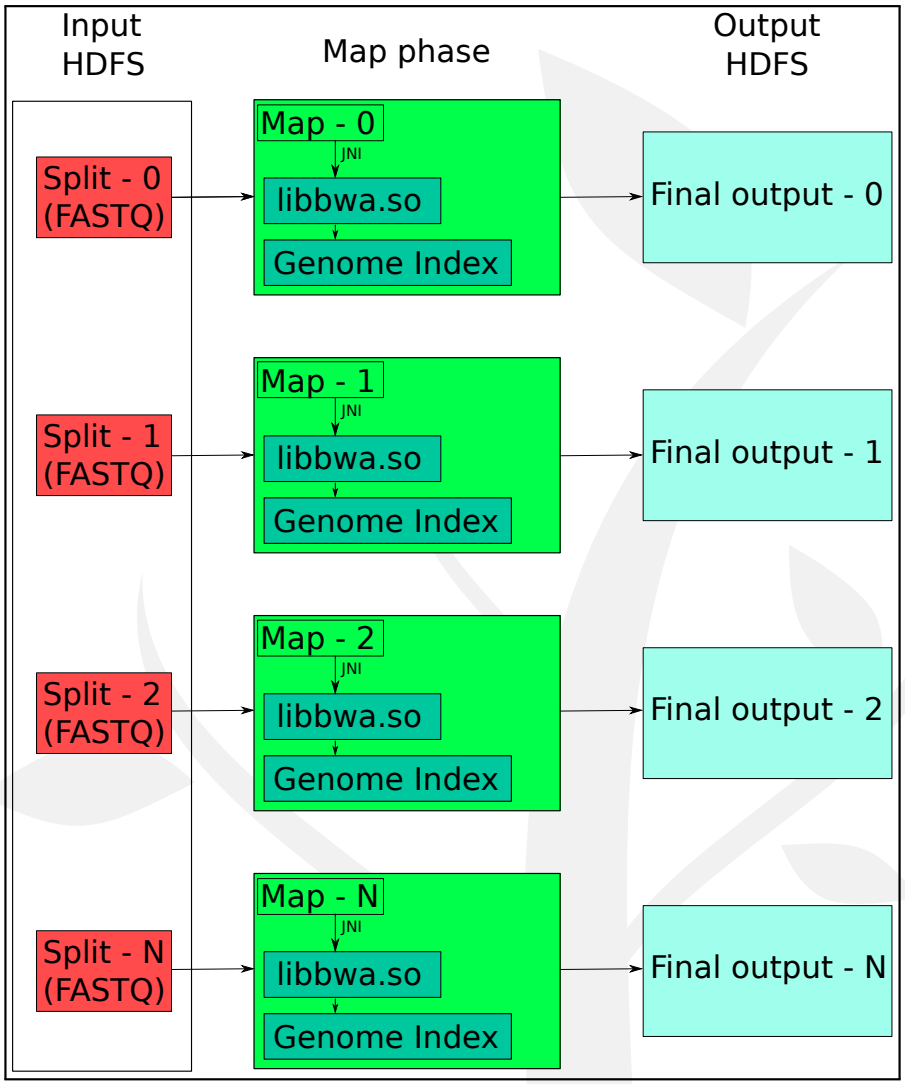


Figure: Model without reducer

Test cases

Comparison against:

- SEAL (BWA-backtrack)[1].
- pBWA (BWA-backtrack)[2].
- BWA version with threads (BWA-MEM).
- Execution time and speed up.

Tag	Name	Number of pairs	bp	Size
D1	NA12750/ERR000589	12×10^6	51	3.9 GB
D2	HG00096/SRR062634	24.1×10^6	100	13.4 GB
D3	150140/SRR642648	98.8×10^6	100	54.7 GB

Table: Main characteristics of the input datasets from 1000 Genomes.

[1] L. Pireddu, S. Leo, and G. Zanetti, "SEAL: a distributed short read mapping and duplicate removal tool," **Bioinformatics**, vol. 27, no. 15, pp. 2159–2160, 2011

[2] D. Peters, X. Luo, K. Qiu, and P. Liang, "Speeding up large-scale next generation sequencing data analysis with pBWA," **Journal of Applied Bioinformatics & Computational Biology**, vol. 1, no. 1, pp. 1–6, 2012

Test environment

Hardware platform characteristics

- Hadoop cluster in Amazon EC2.
- 4 computing nodes.
- 16 cores per node (Intel Xeon E5-2670 at 2.5GHz CPUs).
- Hadoop version is 2.6.0.

Results - Execution time

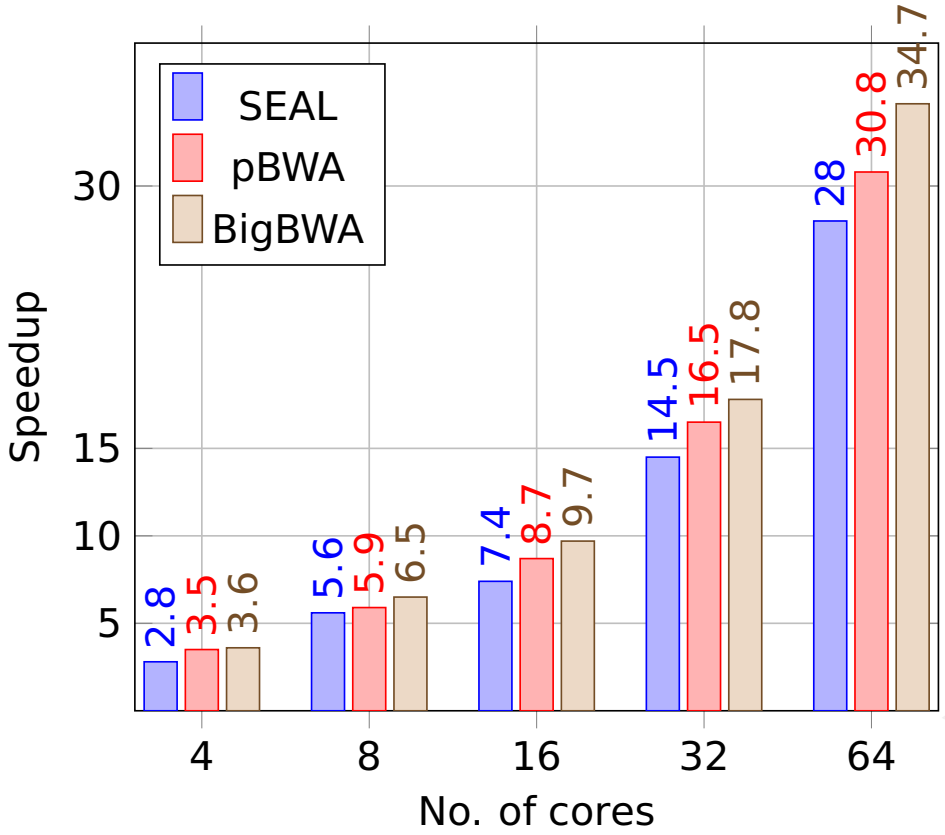
BWA-backtrack

- Sequential time for D1 is **148.5** minutes and **556.9** for D2.
- With 64 cores, for D1 **4.5** minutes, and for D2, **15.3**.
- The speed-up against SEAL has an improvement of **1.27**× and against pBWA of **1.13**×.

BWA-MEM

- Sequential times for D1, D2 and D3 are respectively **106.6**, **258.0** and **3208.6** minutes.
- With BigBWA and 64 cores **3.0**, **7.2** and **87.2** minutes.
- BWA version with threads is a shared memory approach, we can only measure up until **16 cores**.

Results - Speed up



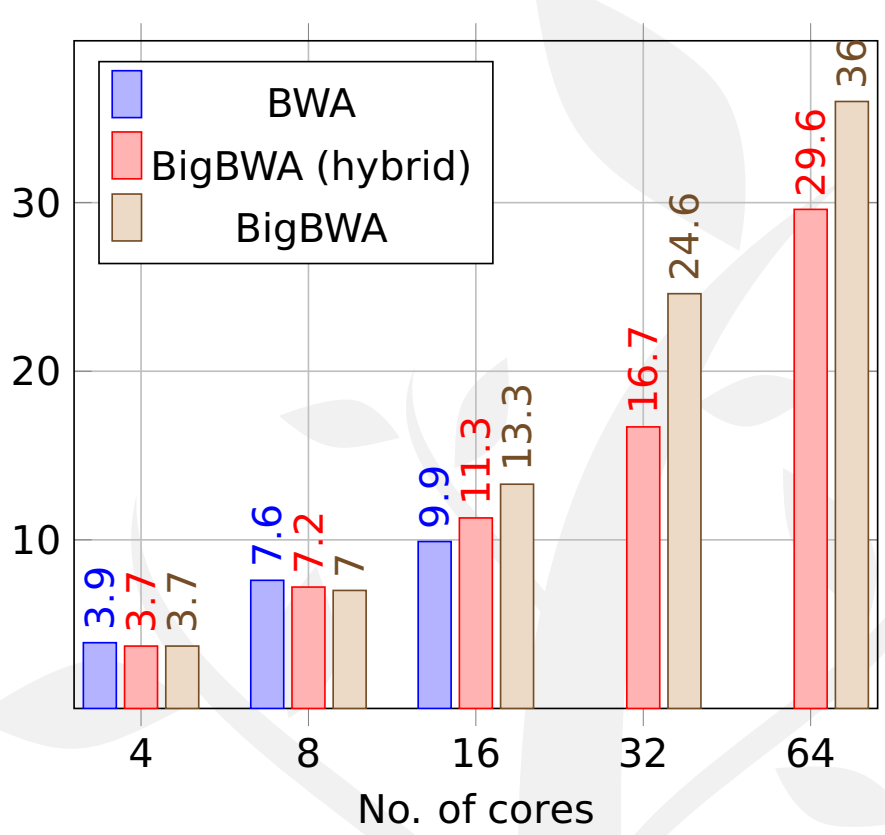


Figure: Average speedups for BWA-backtrack (left) and BWA-MEM (right) algorithms.

Computational Genomics conclusions

Conclusions

- Use of **BWA** to perform the alignment.
- We create an publish **BigBWA**[3]
- **Improvement** respect state of the art solutions **(BWA-backtrack)**.
- The only tool that offers a **distributed memory** approach for **BWA-MEM**.

Future work

- Develop a version to run in **Spark**.
- Create a **data flow** that complete the process by comparing **different alignments**.

[3] J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, "BigBWA: Approaching the Burrows–Wheeler aligner to Big Data technologies," **Bioinformatics**, 2015

Natural Language Processing (NLP)

Motivation

- Each day are created around **2.5 quintillion** bytes of data (2.5 Exabytes).
- Weather sensors, social networks, digital images, etc . . .
- In many cases, this information is **not structured**.

Natural Language Processing (NLP)

Definition

Field of computer science, artificial intelligence, and computational linguistics **concerned** with the **interactions between computers and human** (natural) languages.

Why?

NLP is considered as one of the **most appropriate technologies to structure and organize** textual information on the Internet, in constant and exponential growth.

A Big Data approach for NLP

Problems

- Most of NLP programs are written in **Perl**.
- This is due to its unique ability to process text using **regular expressions**.
- Researchers from this area have found in **Hadoop Streaming** a way to easily analyze big volumes (Gigabytes or even Terabytes) of textual information.

Hadoop Streaming

- Utility included within **Hadoop**.
- It allows to run codes written in any language with Hadoop.
- Restriction: codes must read from “stdin” and write to “stdout”.
- **Important degradations** in the performance have been observed by using this tool[4].
- Anyway, it is a **useful tool** if you do not want to **rewrite** your programs and still run them in a Big Data environment.

[4] M. Ding, L. Zheng, Y. Lu, L. Li, S. Guo, and M. Guo, “More convenient more overhead: the performance evaluation of Hadoop streaming,” in **ACM Symp. on Research in Applied Computation**, 2011, pp. 307–313

How Hadoop Streaming works?

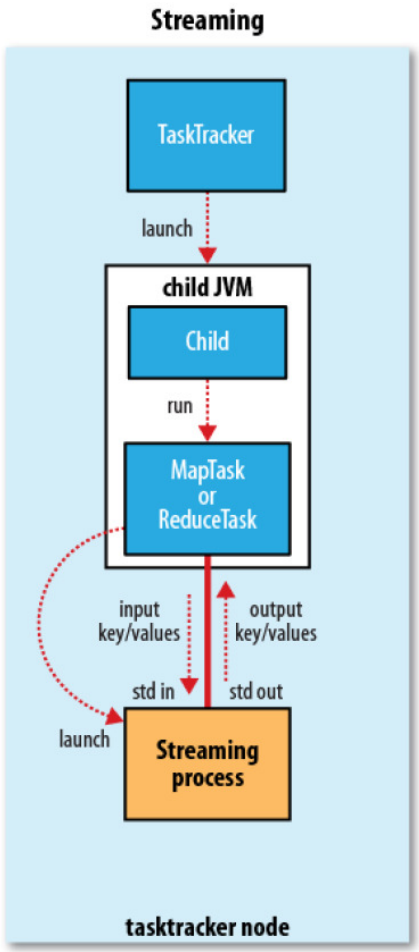


Figure: Relationship of the Streaming executable and its child.

- The **child JVM** creates an external process to launch the user code.
- This external process receives as input the **key - value** pairs from the input.
- As a result, the child JVM receives **key - value** pairs from the external process.
- All these communications between external process and JVM, are made through **pipes**.
- This works for maps and reduces.

T. White, **Hadoop: The Definitive Guide**, 3rd ed. O'Reilly Media, Inc., 2012

Our approach

- Perl is widely used for its unrivaled ability to use **regular expressions.**
- Execute **Perl codes** in Hadoop...
  and we want to do it with a really **good performance and scalability**.
- In consequence, we try to avoid the use of Hadoop Streaming.

What we intend to do (II)

- Translate **MapReduce Perl programs into MapReduce Java programs.**
- This is a hard and tedious job for big applications.
- **Solution**: build a tool that automatically translates MapReduce Perl codes into Java MapReduce programs.
- **Warning**: it will not translate any existing Perl program into Java!!

The Perldoop Tool

Characteristics

- Perldoop *(https://github.com/citiususc/perldoop)* is a system based on **Tags** and Java **Templates**.
- **Tags** in the Perl code are used to identify the piece of code to translate, the type of the variables and other useful parameters in the translation process.
- **Templates** are small pieces of Java code with the “imports” required to run the program, the class name and builder, map and reducer definitions, etc . . .

Templates and tags system

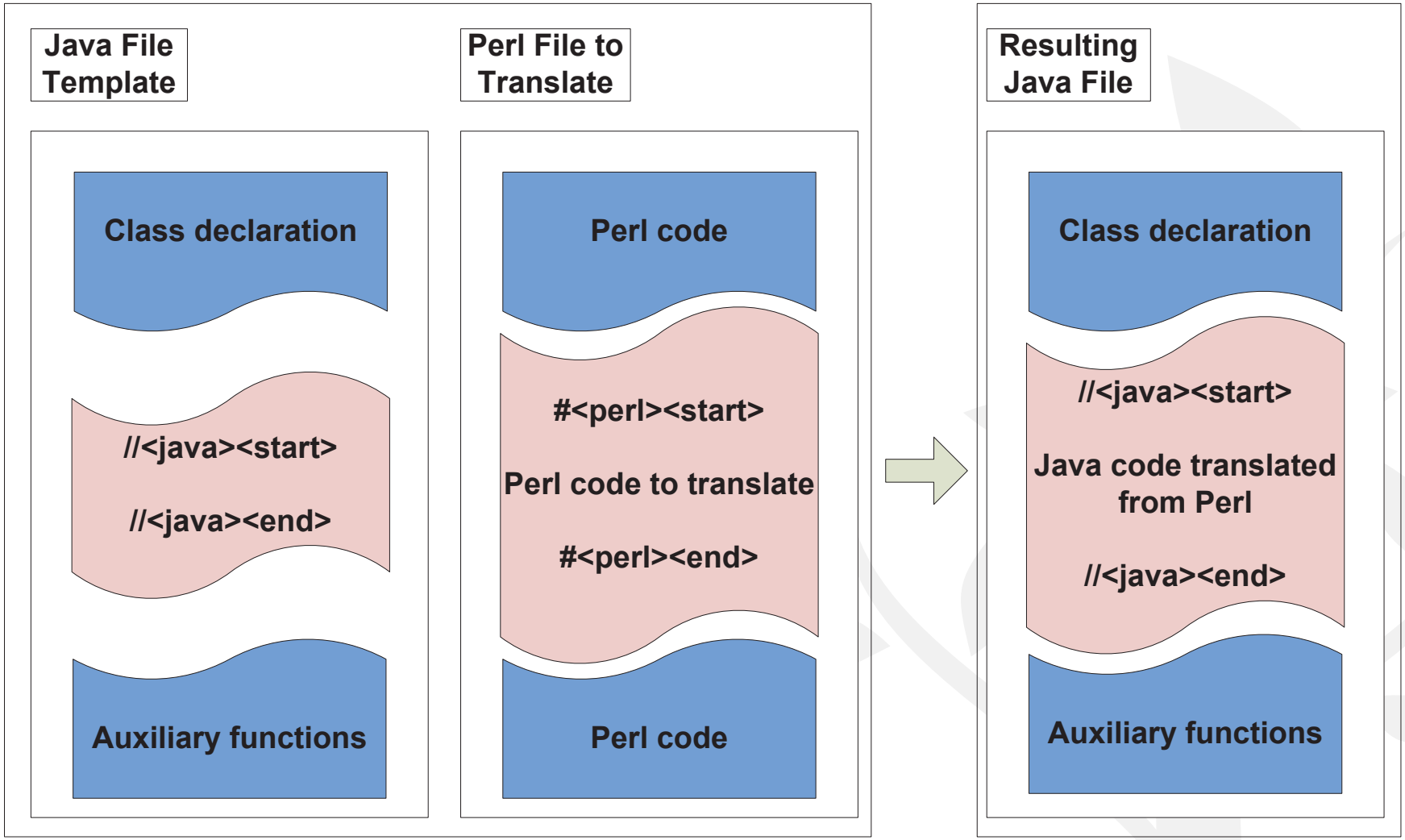


Figure: Templates and tags system.

The tool

- Perldoop was developed using **Python**.
- It takes as input MapReduce Perl codes and produces the equivalent Java codes.
- The user needs few knowledge about Java language.
- It is important to put the same name to variables in Java templates and in the Perl code.
- The user has to follow 6 **implementation rules** to code in Perl:
  1. *Restrict the access to array positions not previously allocated.*
  2. Ordered conditional blocks.
  3. Perform string concatenations with the "." operator.
  4. Declare and initialize the variables with the corresponding label.
  5. *Use a different name for each variable.*
  6. Not to use integer variables as boolean in Perl.

Code example. WordCount Map

```
#!/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>
```

```
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());
    }
  }
}
```

Code example. WordCount Reducer (I)

```
#!/usr/bin/perl -w
#<perl><start>
my $count = 0;   #<var><integer>
my $value;       #<var><integer>
my $newkey;      #<var><string>
my $oldkey;      #<var><string><null>
my $line;        #<var><string>
my @keyValue;    #<var><string>

while ($line = <STDIN>) {        #<reduce>
 chomp ($line);
 $keyValue = split ("\t",$line);
 $newkey = $keyValue[0];
 $value  = $keyValue[1];

 if (!(defined($oldkey))) {
   $oldkey = $newkey;
   $count  = $value;
 }
 else {
   if ($oldkey eq $newkey) {
     $count = $count + $value;
   }
```

```
else {
     my $returnKey =
     $oldkey."\t"; #<var><string>
     my $returnValue =
     $count."\n"; #<var><string>
     print $returnKey.$returnValue;
     $oldkey = $newkey;
     $count  = $value;
   }
 }
}
my $returnKey =
$oldkey."\t"; #<var><string>
my $returnValue =
$count."\n"; #<var><string>
print $returnKey.$returnValue;

#<perl><end>
```

Code example. WordCount Reducer (II)

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

public static class WordCountReducer
    extends Reducer<Text, Text, Text,
    Text> {
  int count  = 0;
  @Override
  public void reduce(Text key, Iterable<
      Text> values,
  Context context) throws IOException,
      InterruptedException {

    //<java><start>
    int value;
    String newkey;
    String oldkey = null;
    String line;
    String keyValue;
    for (Text val : values) {
      newkey    = key.toString();
      value     = Integer.parseInt(val.
          toString());
      if (!(oldkey!= null)) {
        oldkey  = newkey;
        count   = value;
      }
```

```
      else{
        if (oldkey.equals(newkey) ) {
          count  = count + value;
        }
        else{
          String returnKey    = oldkey;
          String returnValue  = String.
              valueOf(count);
          context.write(new Text(returnKey
              ),
          new Text(returnValue));
          oldkey  = newkey;
          count   = value;
        }
      }
    }
    String returnKey    = oldkey;
    String returnValue  = String.valueOf(
        count);
    context.write(new Text(returnKey),new
        Text(returnValue));
    //<java><end>

  }
}
```

Test cases

- Tested using several **NLP (Natural Language Processing)** modules:
  1. *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...
  2. *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...
  3. *Named Entity Classification (NEC)*: It is the process of **classifying entities** by means of classes such as “People”, “Organizations”, “Locations”, or “Miscellaneous”.
- Hardware platform characteristics:
  - Virtual cluster installed at the Galicia Supercomputing Center (CESGA), with 64 nodes.
  - Intel Xeon E5520 processor.
  - The Hadoop version is the 1.1.2, while the Java and Perl versions are 1.7.0 and 5.10.1 respectively.
  - The input data is plain text from Wikipedia (2.1 GB).

Execution times and speed up

Execution times

- The sequential execution time for the **NEC** module (the most time consuming one) is in *Perl* **457.4 hours**. In *Java* is **84.2 hours**.
- Using **64 cores** the execution time is, with Perl and Hadoop Streaming, **15.5 hours**. With Perldoop and Hadoop we have an execution time of **1.5 hours**.

Speed up

- The speed up in the case of **64 cores** is of **29.4**$\times$ in the case of Hadoop Streaming and **57.9**$\times$ in the case of Perldoop and Hadoop.
- This means that, with 64 cores, we **almost** reach the **perfect speed up** of 64$\times$.

Performance improvement

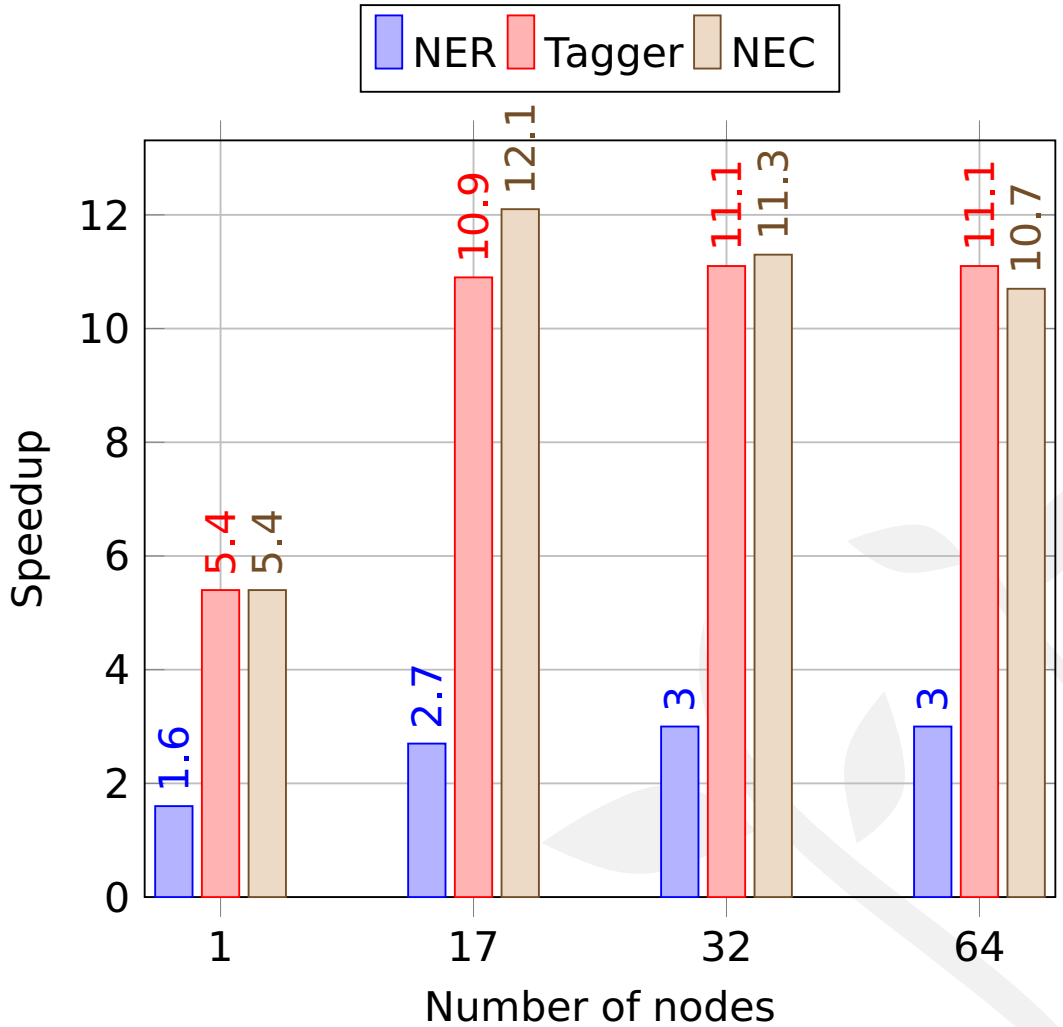


Figure: Performance improvement of the Java modules generated by Perldoop using Hadoop with respect to the use of Perl and Hadoop Streaming.

NLP conclusions

Conclusions

1. **Perl** is widely used in several areas (NLP, Bioinformatics, . . .) for its unrivaled ability to use regular expressions.
2. **Perldoop**[5]: Automatically translate MapReduce Perl codes into Hadoop-ready Java codes .
3. Tool tested with some NLP modules, but it is a general purpose tool for MapReduce programs.
4. New Java modules clearly outperforms the Perl ones, reaching **speedups** up to **12**×.

[5] J. M. Abuín, J. C. Pichel, T. F. Pena, P. Gamallo, and M. García, "Perldoop: Efficient Execution of Perl Scripts on Hadoop Clusters," in **IEEE International Conference on Big Data**, 2014, pp. 766 – 771

NLP conclusions

Future Work

1. Try to avoid the use of Java Templates.
2. Extend the tool to translate MapReduce Perl codes into Spark ready codes.
3. Test the tool with new applications.
4. Try to improve this model by defining a formal grammar for the translation process.

Iterative methods

Iterative methods

- Numeric methods are the **basis** of most **scientific simulation software**.
- Solve **linear systems** with **thousands of equations and unknowns**.
- They require a **big** quantity of **computation time**.
- **Classic HPC** solutions use MPI, OpenMP, CUDA, . . . have a good performance, but lack of **fault tolerance**.

Big Data solution

Big Data equation systems solving

- Hadoop can perform **iterative processes**, but with the consequent **loss of performance** because it has to **write and read** intermediary results to/from disk.
- Apache **Spark** can handle iterative processes in a mos efficient way thanks to its **in-memory** approach and the use of **RDDs**.
- Implementation of the **conjugate gradient** method and comparison to a **C++** implementation with **MPI**.

Input matrices

Tag	Dimensions	Size
D1	20000 × 20000	3.35 GB
D2	30016 × 30016	7.55 GB
D3	40000 × 40000	13.41 GB
D4	50016 × 50016	20.96 GB

Table: Main characteristics of the input square matrices.

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$

$\mathbf{p}_0 := \mathbf{r}_0$

$k := 0$

repeat

$$\alpha_k := \frac{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}$$

$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$

$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$

if r_{k+1} is sufficiently small then exit loop

$$\beta_k := \frac{\mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}$$

$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$

$k := k + 1$

end repeat

The result is $\mathbf{x}_{k+1}$

Dense Matrix - Vector multiplication

Figure: Conjugate gradient pseudocode.

Execution times and speed up

Execution times

- The sequential time is similar for the C++ and Java cases with D4, **228** minutes.
- Using **32 cores**, the Spark version has an execution time of **43.19** minutes, and the C++ with MPI **27.58** minutes.
- Generally, execution times with MPI are between **1.5**× and **1.8**× better than the ones with Spark.

Speed up

- For the case of D4 with **32 cores** we reach a speed up of **8.33**× with MPI and **5.26**× with Spark.
- Speed ups are also between **1.5**× and **1.8**× better in MPI than the ones with Spark.

Iterative methods conclusions

Conclusions

1. It is **possible** to use Big Data infrastructures to solve this kind of methods.
2. The performance is **not as good** as the one obtained with classic HPC methods.
3. But we have the advantage of the **fault tolerance**.

Future work

1. Develop **other iterative methods** that can fit into this paradigm.
2. Try to integrate this methods into the **Spark MLlib linear algebra package**.

Contents

Final conclusions

Conclusions

- It is **possible** to use Big Data for classic HPC scientific applications.
- The performance obtained can **outperform MPI** in some cases (genomics alignment) but it is more complicated in others (iterative methods).
- Hadoop and Spark give us fault tolerance. This can be useful in programs that take a very long time to obtain a final result.

Future work

- Apply Big Data technologies to **other scientific applications**.
- **Improve** all the software we developed for our current lines of work.
- Use the HDFS native library ***libhdfs.so*** to take advantage of use the HDFS with **MPI** in all these applications.

References I

- L. Pireddu, S. Leo, and G. Zanetti, “SEAL: a distributed short read mapping and duplicate removal tool,” **Bioinformatics**, vol. 27, no. 15, pp. 2159–2160, 2011.
- D. Peters, X. Luo, K. Qiu, and P. Liang, “Speeding up large-scale next generation sequencing data analysis with pBWA,” **Journal of Applied Bioinformatics & Computational Biology**, vol. 1, no. 1, pp. 1–6, 2012.
- J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, “BigBWA: Approaching the Burrows–Wheeler aligner to Big Data technologies,” **Bioinformatics**, 2015.
- M. Ding, L. Zheng, Y. Lu, L. Li, S. Guo, and M. Guo, “More convenient more overhead: the performance evaluation of Hadoop streaming,” in **ACM Symp. on Research in Applied Computation**, 2011, pp. 307–313.
- T. White, **Hadoop: The Definitive Guide**, 3rd ed. O’Reilly Media, Inc., 2012.

References II

J. M. Abuín, J. C. Pichel, T. F. Pena, P. Gamallo, and M. García, "Perldoop: Efficient Execution of Perl Scripts on Hadoop Clusters," in **IEEE International Conference on Big Data**, 2014, pp. 766 – 771.

Thank you for your attention

josemanuel.abuin@usc.es
https://jmabuin.wordpress.com
https://github.com/jmabuin