SPARK

T. VERBEIREN

9/7/2014

CONTENTS

ME, MYSELF AND I

PhD on Neural Networks

ICT & Management Consulting

ML and Data Management

Visual Analytics @ KU Leuven

ESAT

ExaScience life lab

DATA INTUITIVE

Data Visualization Lab

INTRODUCTION

DISTRIBUTION IS HARD ...

2 world views:

- **H**igh **P**erformance **C**omputing
- **H**igh **T**hroughput **C**omputing

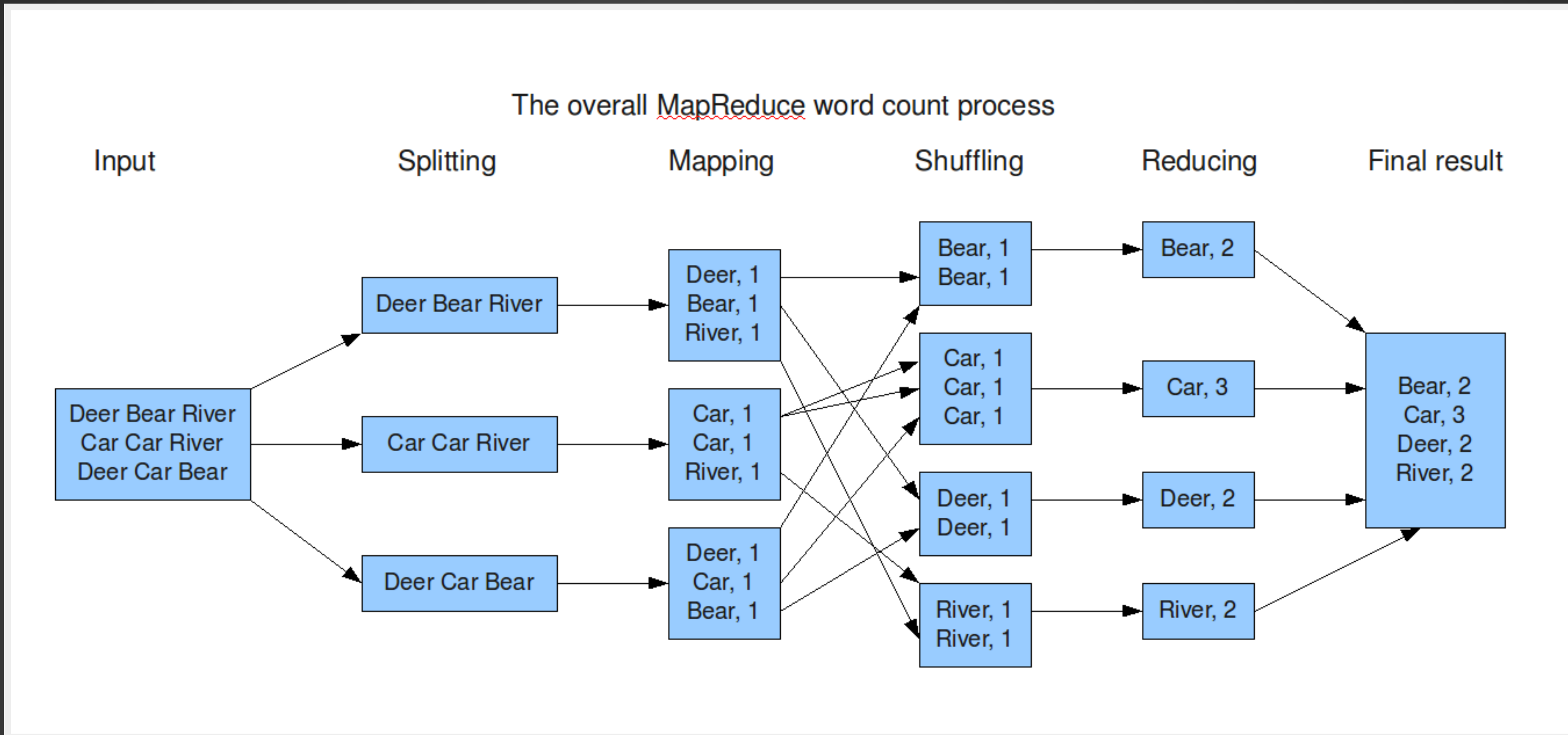
MAP / REDUCE

`MAP`

`REDUCE`

(Nothing special ?!)

WORD COUNT IN M/R


The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

```java
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");
```

WHAT IF …

… WE COULD JUST WRITE

```
val y = x map () reduce ()
```

… THIS COULD BE EXTENDED

```
val y = x map () filter() map () flatMap () reduce ()
```

WHAT WOULD BE NEEDED?

1. Platform
2. Parallel abstraction mechanism
3. Language support

SPARK

- Berkeley University
- Apache Project
- v1.0 released on May 30d 2014
- Written in Scala
- Supported by DataBricks
- Used by ...

LANGUAGE SUPPORT

Language Support

Python

```
lines = sc.textFile(...)
lines.filter(lambda s: "ERROR" in s).count()
```

Scala

```
val lines = sc.textFile(...)
lines.filter(x => x.contains("ERROR")).count()
```

Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains("error");
  }
}).count();
```

Standalone Programs

- Python, Scala, & Java

Interactive Shells

- Python & Scala

Performance

- Java & Scala are faster due to static typing
- …but Python is often fine

PLATFORM

Built for low-latency

ABSTSTRACTION MECHANISM

Resilient **D**istributed **D**atasets

RDDS

Immutable Collection

Accepting **transformations** and **actions**

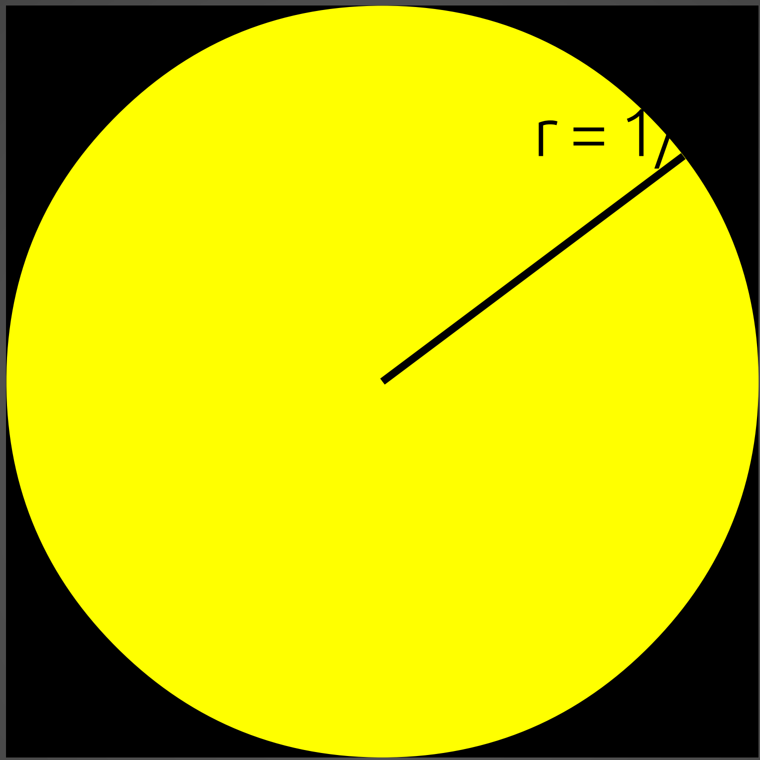
TRANSFORMATIONS

- map
- filter
- sample
- union/intersection
- groupByKey
- reduceByKey
- join
- …

ACTIONS

- `reduce`
- `collect`
- `count`
- `take(n)`
- `saveAsTextFile`
- …

THE FIRST EXAMPLE



$$P(\text{hitting circle}) \approx \text{Surface circle} = \frac{\pi}{4}$$

```
import sc._

val N = 10000000

// Generate a sequence of numbers and distribute
val par = parallelize(1 to N)

// Generate a point in 2D unit square
def randomPoint:(Double,Double) = {
    val x = Math.random()
    val y = Math.random()
    (x,y)
}
// Check if a point lies in the unit circle
def inCircle(point:(Double,Double)):Int = {
    if (point._1*point._1 + point._2*point._2 < 1) 1 else 0
}
```

```scala
// List of hits yes/no
val inCircleList = par map(i => inCircle(randomPoint))

// Return the first 5 elements from the RDD
inCircleList take 5

// Get info about the RDD
inCircleList.toDebugString

// The number of hits
val total = inCircleList reduce (_+_)

// Probability of hitting the circle *4 = Pi
val S = 4. * total / N
```

From the Spark examples page

```
val count = parallelize(1 to N).map{i =>
  val x = Math.random()
  val y = Math.random()
  if (x*x + y*y < 1) 1 else 0
}.reduce(_ + _)
println("Pi is roughly " + 4.0 * count / N)
```

Hadoop M/R in Spark

```
// Read a file from Hadoop FS, (e.g. Ulysses / Project Gutenberg)
// and process it similar to Hadoop M/R
val file = textFile("Joyce-Ulysses.txt")

// Convert to an array of words in the text
val words = file.flatMap(_.split(" "))

// Map to (key,value) pairs
val mapped = words map (word => (word,1))

// Sort and group by key,
// Result is of form (key, List(value1, value2, value3, ...))
val grouped = mapped sortByKey() groupByKey()

// The length of the values array yields the amount
val result = grouped map {case (k,vs) => (k,vs.length)}
// But where is the *reduce*?
```

Be careful with *definitions* of `map` and `reduce`!

```scala
// Read a file from Hadoop FS, (e.g. Ulysses / Project Gutenberg)
// and process it similar to Hadoop M/R
val file = textFile("Joyce-Ulysses.txt")

// Convert to an array of words in the text
val words = file.flatMap(_.split(" "))

// Map to (key,value) pairs
val mapped = words map (word => (word,1))

// Sort and group by key,
// Result is of form (key, List(value1, value2, value3, ...))
val grouped = mapped sortByKey() groupByKey()

// The length of the values array yields the amount
// val result = grouped map {case (k,vs) => (k,vs.length)}
val result = grouped map {case (k,vs) => (k, vs reduce (_+_))}
```

In Spark, we would use:

```
val file = textFile("Joyce-Ulysses.txt")
val words = file.flatMap(_.split(" "))
val mapped = words map (word => (word,1))
val result = mapped reduceByKey(_+_)
result collect
```

… AND REPL

… AND WEB INTERFACE

Spark Master at spark://ly-1-00:7077

URL: spark://ly-1-00:7077
Workers: 4
Cores: 96 Total, 96 Used
Memory: 373.7 GB Total, 128.0 GB Used
Applications: 1 Running, 5 Completed
Drivers: 0 Running, 0 Completed

Workers

Id	Address	State	Cores	Memory
worker-20140626200902-ly-1-00.exascience.org-47436	ly-1-00.exascience.org:47436	ALIVE	24 (24 Used)	93.4 GB (32.0 GB Used)
worker-20140626200903-ly-1-01.exascience.org-60945	ly-1-01.exascience.org:60945	ALIVE	24 (24 Used)	93.4 GB (32.0 GB Used)
worker-20140626200903-ly-1-09.exascience.org-48016	ly-1-09.exascience.org:48016	ALIVE	24 (24 Used)	93.4 GB (32.0 GB Used)
worker-20140626200904-ly-2-13.exascience.org-56964	ly-2-13.exascience.org:56964	ALIVE	24 (24 Used)	93.4 GB (32.0 GB Used)

Running Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20140627203341-0005	Spark shell	96	32.0 GB	2014/06/27 20:33:41	toniv	RUNNING	49 min

Completed Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20140627194920-0004	Spark shell	96	32.0 GB	2014/06/27 19:49:20	toniv	FINISHED	43 min
app-20140626213510-0003	Spark shell	96	32.0 GB	2014/06/26 21:35:10	toniv	FINISHED	14 min
app-20140626211353-0002	Spark shell	96	32.0 GB	2014/06/26 21:13:53	toniv	FINISHED	21 min
app-20140626202327-0001	Spark shell	96	32.0 GB	2014/06/26 20:23:27	toniv	FINISHED	50 min
app-20140626201124-0000	Spark shell	96	32.0 GB	2014/06/26 20:11:24	toniv	FINISHED	11 min

... AND DISTRIBUTED MEMORY CACHING

```
val file = textFile("Joyce-Ulysses.txt")
val words = file.flatMap(_.split(" "))
val mapped = words map (word => (word,1))
// Cache the RDD for later use
val cached = mapped cache()
// Use the cached version
val result = cached reduceByKey(_+_)
// Oops, nothing happens?
result.collect
// Laziness... oh my
result.collect

// Count how many times the word 'the' occurs in the text
cached filter {case(word,v) => word=="the"} reduceByKey(_+_) collect
```

… AND INTERACTIVE USE

... AND THE ECOSYSTEM

- Spark SQL: http://spark.apache.org/sql/
- Spark Streaming: http://spark.apache.org/streaming/
- BlinkDB: http://blinkdb.org/
- MLlib: http://spark.apache.org/mllib/
- GraphX: http://spark.apache.org/graphx/
- SparkR: http://amplab-extras.github.io/SparkR-pkg/

NOT COVERED

BETTER ?
FASTER ?
EASIER ?

…

RDDS UNDER THE HOOD

http://dl.acm.org/citation.cfm?id=2228301

CONFIGURATION & PERFORMANCE

https://spark.apache.org/docs/latest/configuration.html

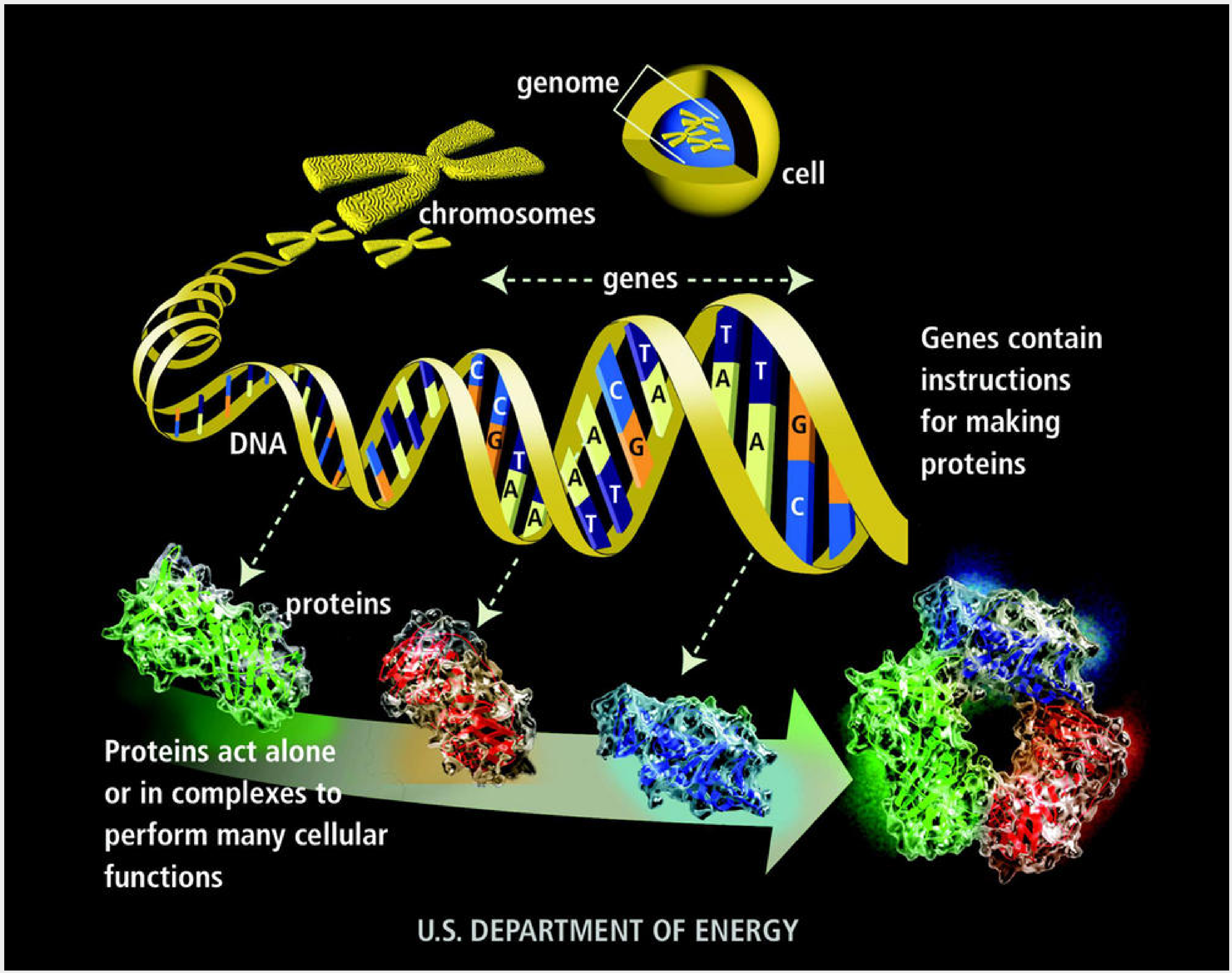
https://spark.apache.org/docs/latest/tuning.html

INSTALLATION & DEPLOYMENT

https://spark.apache.org/docs/latest/cluster-overview.html

EXAMPLE(S)

GENOMIC DATA


genome
cell
chromosomes
genes
DNA
Genes contain instructions for making proteins
proteins
Proteins act alone or in complexes to perform many cellular functions
U.S. DEPARTMENT OF ENERGY

3 billion base pairs (3.2×10^9)

Packaged in chromosomes

~ 3GB for one human

Analysis requires a lot of processing power and storage

Transcription Factors: proteins that bind on region

Coverage: basepair occurence in sequencing

Coverage data:

Chromosome	Position	Sequencing coverage
19	11004	1
19	11005	2
19	11006	2
19	11007	2
19	11008	3
19	11009	3

Transcription Factor data:

```
> awk 'BEGIN {srand()} !/^$/ { if (rand() <= .00001) print $0}' bedfile.bed

chr1    70529738    70529754    Maf     .   -
chr1    161676477   161676495   Pou2f2  .   -
chr1    176484690   176484699   AP-1    .   -
chr10   6020071     6020084     CTCF    .   -
chr11   1410823     1410838     NF-Y    .   -
chr16   4366053     4366067     YY1     .   +
chr17   77824593    77824602    BAF155  .   +
chr19   10947006    10947013    Rad21   .   -
chr19   49342112    49342121    SIX5    .   +
chr22   39548908    39548922    Irf     .   +
chr7    100048475   100048485   Egr-1   .   -
chr8    119123364   119123374   YY1     .   +
chr8    128562635   128562649   p300    .   -
chr9    14315969    14315982    Egr-1   .   -
chrX    101409366   101409384   CTCF    .   +
```

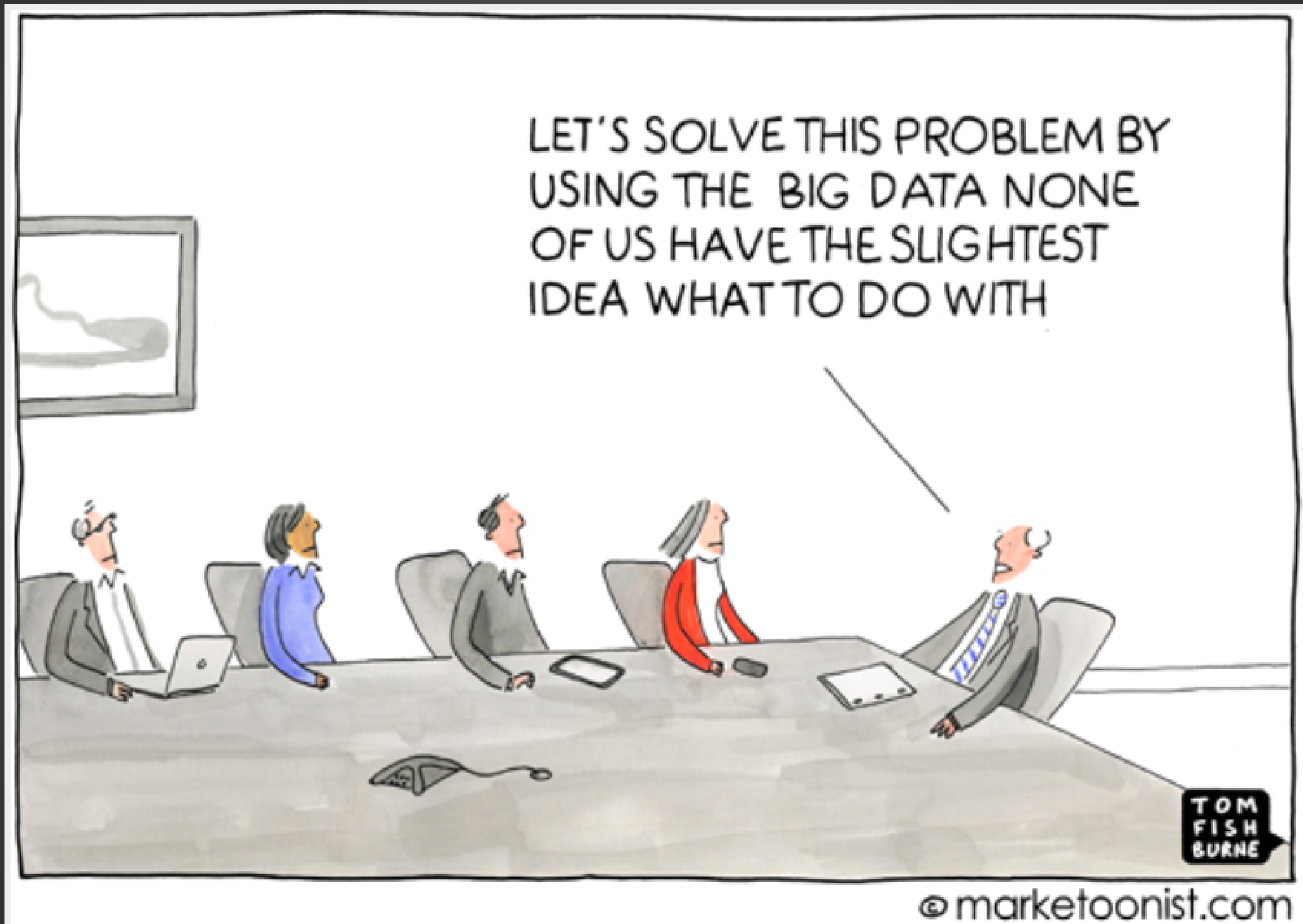
```
// Load files from HDFS
val covFile = sc.textFile("NA12878.chrom19.SLX.maq.SRP000032.2009_07.coverage
val bedFile = sc.textFile("201101_encode_motifs_in_tf_peaks.bed",8)

// Class to hold records from coverage data
class covData(val chr: String, val pos: Int, val cov: Int) {
    def this(line: Array[String]) {
     this(line(0).toString, line(1).toInt, line(2).toInt)
    }
}

// Class to hold records from Transcription Factor data
class tfsData(val chr: String, val pos1: Int, val pos2:Int, val tf: String) {
    def this(line: Array[String]) {
     this(line(0).toString, line(1).toInt, line(2).toInt, line(3).toString)
    }
}
```

```scala
// Turn input files into an RDD of objects
val cov = covFile.map(_.split("\\s+")).map(new covData(_))
val tfs = bedFile.map(_.split("\\s+")).map(new tfsData(_))

// Count the number of items in both datasets
cov.count
tfs.count

// Cache in memory
val ccov = cov cache
val ctfs = tfs cache

// Count once for the caching to occur
ccov.count
ctfs.count
```

```scala
// Turn coverage data into K/V pairs
val kvcov = ccov.map(x => (x.pos,(x.cov))).cache
// Turn TF data into K/V pairs
val kvtfs = ctfs.filter(x => x.chr == "chr19").map(x => (x.pos1,(x.pos2,x.tf)

// Activate the caching of the coverage data
kvcov.count

// Join both datasets together by key
val cjoined = kvcov.join(kvtfs)

// Waaaw, that's fast! In fact, nothing happened yet.
// select 5 entries to see the result but reformat first
val flatjoined = cjoined map { case(x,(y,(z,zz))) => (x,z,zz,y) }
flatjoined take 5

flatjoined.toDebugString
```

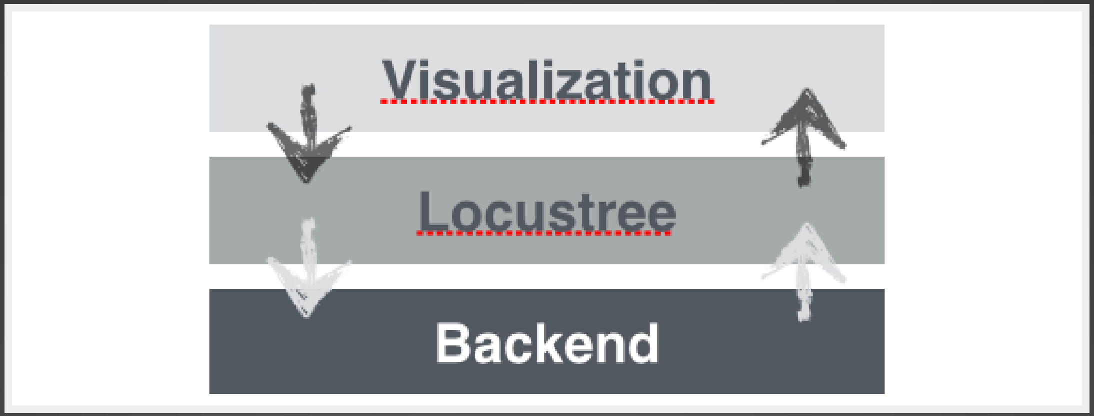
VISUALIZATION OF BIG DATA

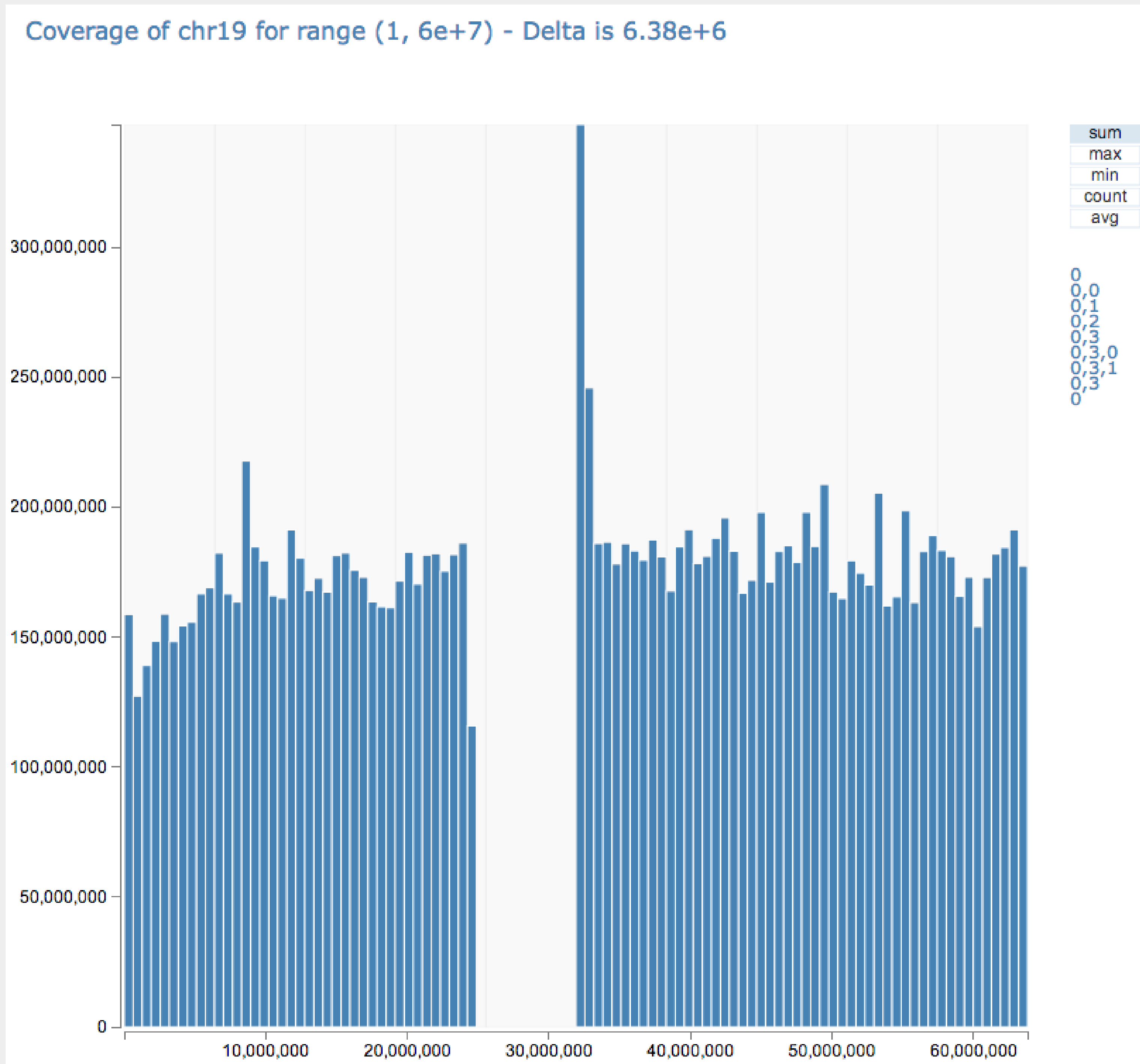
VISUAL ANALYTICS



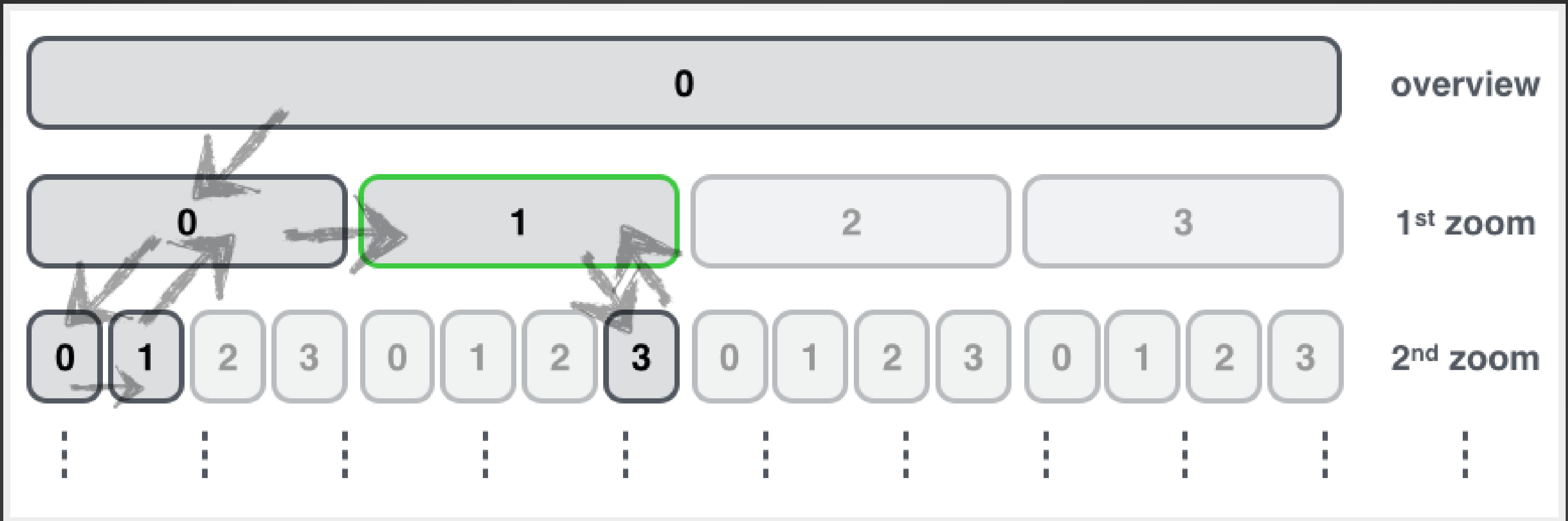
Visualize information on a chromosome

Create bins and aggregate within bins


Visualization
Locustree
Backend



Coverage of chr19 for range (1, 6e+7) - Delta is 6.38e+6
300,000,000
250,000,000
200,000,000
150,000,000
100,000,000
50,000,000
0
10,000,000
20,000,000
30,000,000
40,000,000
50,000,000
60,000,000
sum
max
min
count
avg
0
0,0
0,1
0,2
0,3
0,3,0
0,3,1
0,3
0

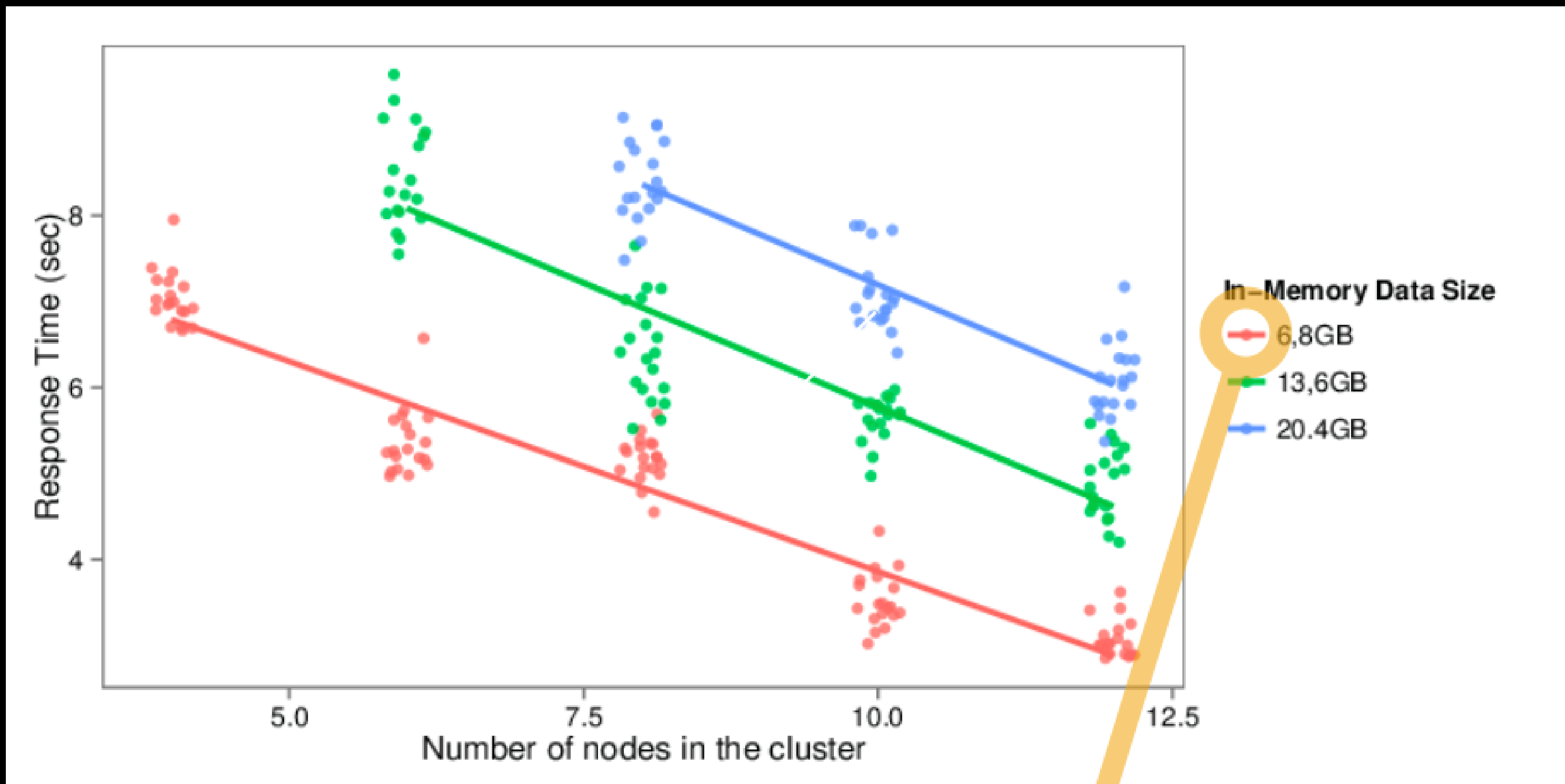

0
overview
0
1
2
3
1st zoom
0
1
2
3
0
1
2
3
0
1
2
3
0
1
2
3
2nd zoom

Lazy functional tree zipper

Part of `treeDraw`:

```
d3.select(window).on("keydown", function() {
  d3.event.preventDefault();
  switch (d3.event.keyCode) {
    case 38:
      zoomOut(tz,treeDraw);
      break ;
    case 40:
      zoomIn(0,tz,treeDraw);
      break;
    case 37:
      panLeft(tz,treeDraw);
      break;
    case 39:
      panRight(tz,treeDraw);
      break;
  };
});
```

```scala
object CovQuery extends SparkJob with NamedRddSupport {

  type Range = (Int, Int)
  type TreePath = List[Int]

  val B = 100      // number of bins, also N
  val Z = 10       // zoom level with every step

  override def validate(sc: SparkContext, config: Config): SparkJobValidation

  override def runJob(sc: SparkContext, config: Config): Any = {

    // a map with the persistent RDDs
    val listOfPersistentRDDs = sc.getPersistentRDDs

    val chrCache = namedRdds.get[DataPoint](myRDD).get

    // Info about the interval
```



275 million datapoints

Paper submitted to LDAV 2014

Online Submission ID: 109

An Architecture For Interactive Big Data Visualization

Category: Research

Abstract

We present an architecture that allows for interactive visualization of large amounts of data. Visualization of such data often requires pre-processing by aggregating over intervals. Rather than pre-processing the data, our architecture employs a scaling out approach such that adding compute resources has a direct effect on aggregation and visualization performance. We describe a proof-of-concept implementation of this architecture using readily available Open Source technologies. Scaling parameters are derived in order to show that scaling out, in a way transparent to the visualization expert, is a viable option for big data visualization.

Index Terms: Computer Graphics [I.3.3]: Picture/Image Generation—Display algorithms Computer Graphics [I.3.3]: Picture/Image Generation—Distributed/network graphics Methodology and Techniques [I.3.6]: Graphics data structures and data types— [Hardware Architecture]: I.3.1—Parallel processing Life and Medical Sciences [J.3]: Biology and genetics—

Another approach to the interactive zooming and panning in large amounts of data is by pre-processing the data and converting it to an in-memory or on-disk representation that can be rendered sufficiently fast [23, 15, 16]. Pre-processing data, e.g. generating data for different zoom levels, requires significantly more storage space than the original data in itself. Additionally, when considering realtime streaming data, online processing is required. In that case, pre-processing and storing intermediate data is not feasible at all.

1.3 Aggregation

The data visualization mantra *"overview first, zoom and filter, then details-on-demand"* suggests the requirement for different levels of data aggregation [25]. Since we are bounded by a limited display size, at the overview level, the complete data set is aggregated to fit into one view. The aggregation is a function of the underlying data: sum, maximum/minimum, count, value range, average, etc. Additional zoom levels apply similar aggregation functions to smaller and smaller subsets of the data. This means that by zooming in, the

THE END

Some links:

- @tverbeiren
- Slides: https://github.com/tverbeiren/BigDataBe-Spark
- Spark Home: https://spark.apache.org/
- Data Visualization Lab: http://datavislab.org
- ExaScience Life Lab: http://www.exascience.com/
- Data Intuitive: http://data-intuitive.com