RHIPE: A Distributed Environment for the Analysis of Large and Complex Datasets

Saptarshi Guha

March 9, 2010

Analysis of Very Large Data Sets

How to compute with? How to store?

- Parallelized external memory algorithms that use *all* the data

  Requires a different approach for several classes of statistical routines

- Summaries are not enough
- Need a more detailed analysis

Analysis of Very Large Data Sets

- Analyze subsets of the data. Apply numerical routines to each subset. Visualize a representative sample of the subsets.
- Divide and Recombine

Easy to spread computation on subsets across a cluster

Tools for Storing & Computing with Large Data

- **Hadoop DFS**: Open source software to store data across a cluster
  - A replicated, fault tolerant distributed file system
  - Data is partitioned into blocks and replicated across the cluster
- **Hadoop MapReduce**: An open source implementation of Google's MapReduce
  - A programming approach to handling massive (GB,TB,. . .) data
  - A more powerful and distributed version of R's `Map` and `Reduce` (also `tapply`)

Fault tolerance is nice: computers can fail, but your data is still retrievable, MapReduce jobs carry on !

Bringing Hadoop to R

Existing High Performance Computing packages(e.g. `multicore, rmpi, snow`) very convenient but offers no support for large datasets (except `bigmemory`[1])

How to compute with gigabytes of data across a cluster?

[1] which is convenient for data frames

Bringing Hadoop to R

RHIPE

- **R** and **H**adoop **I**ntegrated **P**rocessing **E**nvironment

Bringing Hadoop to R

RHIPE

- **R** and **H**adoop **I**ntegrated **P**rocessing **E**nvironment
- a natural environment to code mapreduce algorithms from within R
- a way to store and analyze datasets with millions of objects

RHIPE

Based on Hadoop Streaming source

Features

- Can read and write scalar vectors(all types) including NA for string and numeric

RHIPE

Based on Hadoop Streaming source

Features

- Can read and write scalar vectors(all types) including NA for string and numeric
- Preserves attributes for R objects: factors, matrices, arrays, data frames and S3 classes

RHIPE

Based on Hadoop Streaming source

Features

- Can read and write scalar vectors(all types) including NA for string and numeric
- Preserves attributes for R objects: factors, matrices, arrays, data frames and S3 classes
- Interop: uses Protocol Buffers: data written using RHIPE can be read using Python, C++, Java and many others (and vice versa)

RHIPE

Based on Hadoop Streaming source

Features

- Can read and write scalar vectors(all types) including NA for string and numeric
- Preserves attributes for R objects: factors, matrices, arrays, data frames and S3 classes
- Interop: uses Protocol Buffers: data written using RHIPE can be read using Python, C++, Java and many others (and vice versa)
- Can read and write to various formats

RHIPE

Based on Hadoop Streaming source

Features

- Can read and write scalar vectors(all types) including NA for string and numeric
- Preserves attributes for R objects: factors, matrices, arrays, data frames and S3 classes
- Interop: uses Protocol Buffers: data written using RHIPE can be read using Python, C++, Java and many others (and vice versa)
- Can read and write to various formats
- Can create files (e.g PDF) and they will be collected for you

RHIPE

Based on Hadoop Streaming source

Features

- Can read and write scalar vectors(all types) including NA for string and numeric
- Preserves attributes for R objects: factors, matrices, arrays, data frames and S3 classes
- Interop: uses Protocol Buffers: data written using RHIPE can be read using Python, C++, Java and many others (and vice versa)
- Can read and write to various formats
- Can create files (e.g PDF) and they will be collected for you
- Most errors are returned directly to the console - easier to debug

Case Study - VoIP

- Network packets for Voice over IP collected for 48 hrs
- Packets can be partitioned into *calls*
- Each call has two directions called *semi-calls*

Case Study - VoIP

- Network packets for Voice over IP collected for 48 hrs
- Packets can be partitioned into *calls*
- Each call has two directions called *semi-calls*

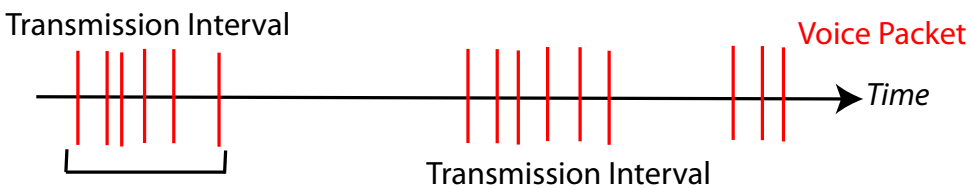
70 gigabytes of text data spread across 8 machines

Case Study - VoIP

Can we analyze this?

- Several types of analyses, we need to **store the data**
- How to compute across the data?

- *Transmission Interval*
- *Jitter*
- *Traffic Rate*: Bits per 30 seconds

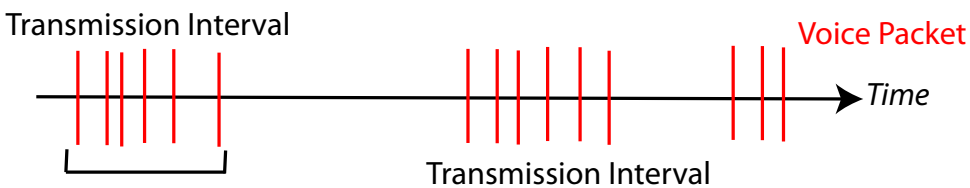


Case Study - VoIP

Can we analyze this?

- Several types of analyses, we need to **store the data**
- How to compute across the data?

- *Transmission Interval*
- *Jitter*
- *Traffic Rate*: Bits per 30 seconds



Question 1 Does the jitter depend on the traffic rate?

Case Study - VoIP

Convert source 70 gigabytes raw text data to R data frames

- Data frame has two columns: time of packet and silence flags
- As many rows as number of packets in a semi-call (< 5000)

Case Study - VoIP

Copy text data to HDFS

```
rhput('/home/sguha/pres/voip/text/20040312-105951-0.iprtp.out','/pres/voip/text')
```

Use RHIPE to convert text data :

```
1079089238.075950 IP UDP 200 67.17.54.213 6086 67.17.50.213 15074 0
...
```

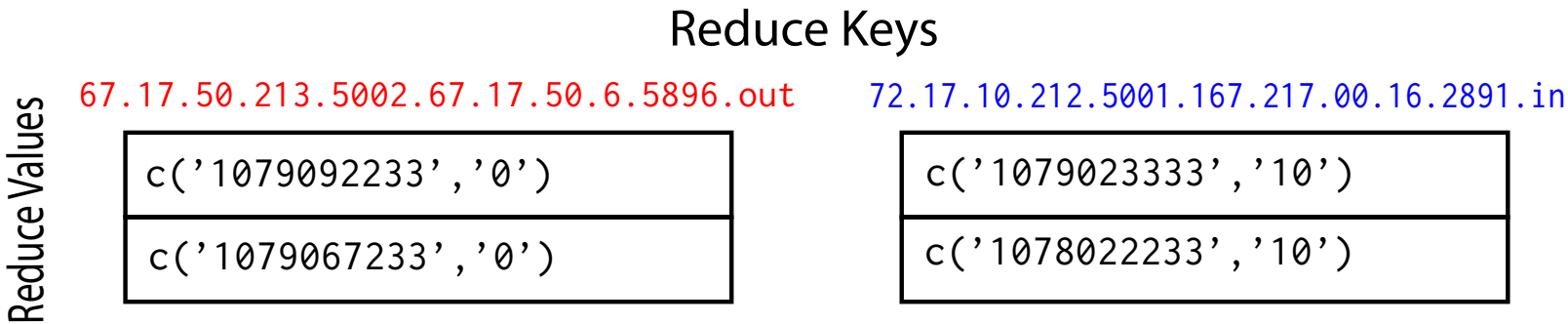
to R data frames

```
input <- expression({
  ## create components (direction, id.ip,id.port) from Sys.getenv("mapred.input.file")
  v <- lapply(seq_along(map.values),function(r) {
    value0 <- strsplit(map.values[[r]]," +")[[1]]
    key <- paste(value0[id.ip[1]],value0[id.port[1]],value0[id.ip[2]]
                 ,value0[id.port[2]],direction,sep=".")
   rhcollect(key,value0[c(1,9)])
  })})
```

Key	Value
67.17.50.213.5002.67.17.50.6.5896.out	c('1079092233','0')
72.17.10.212.5001.167.217.00.16.2891.in	c('1079023333','10')
67.17.50.213.5002.67.17.50.6.5896.out	c('1079067233','0')
72.17.10.212.5001.167.217.00.16.2891.in	c('1078022233','10')

Case Study - VoIP

```
1   reduce <- expression(
2        pre = {   mydata<-list() }
3       ,reduce = {  mydata <- append(mydata,reduce.values)}
4       ,post = {
5         mydata <- do.call("rbind",mydata)
6         colnames(mydata) <- c("time","rtpPT")
7         mydata <- mydata[order(mydata[,'time']),,drop=F]
8         mydata <- data.frame(time = as.numeric(mydata[,'time']),
9                              rtpPT = as.integer(mydata[,'rtpPT']))
10        rhcollect(reduce.key,mydata)
11      })
```



Case Study - VoIP

We can run this from within R:

```
mr<-rhmr(map=input,reduce=reduce, inout=c('text','map'),
         ifolder='/pres/voip/text', ofolder='/pres/voip/df',jobname='create'
         ,mapred=list(mapred.reduce.tasks=5))
mr <- rhex(mr)
```

which takes 40 minutes for 70 gigabytes across 8 computers(72 cores)
Saved as 277K data frames(semi-calls) across 14 gigabytes
We can retrieve semi-calls:

```
rhgetkey(list('67.17.50.213.5002.67.17.50.6.5896.out'),paths='/pres/voip/df/p*')
## a list of lists(key,value pairs)
[[1]][[1]]
[1] "67.17.50.213.5002.67.17.50.6.5896.out"
[[1]][[2]]
        time rtpPT
1 1079092233     0
2 1079092233    19
3 1079092233    19
```

VoIP: Computing Summaries

- For each call, compute the start time,end time and number of packets for each component semi-call.
- Uses a Map and a Reduce (need to aggregate semi-call information at call level)

VoIP: Computing Summaries

- For each call, compute the start time,end time and number of packets for each component semi-call.
- Uses a Map and a Reduce (need to aggregate semi-call information at call level)

87 seconds across 8 computers

```
m<-expression({
 lapply(seq_along(map.values),function(i){
   ## make key from map.keys[[i]]
   value<-if(tmp[11]=="in")
     c(in.start=start,in.end=end,in.dur=dur,in.pkt=n.pkt)
   else
     c(out.start=start,out.end=end,out.dur=dur,out.pkt=n.pkt)
   rhcollect(key,value)
 })
})
r<-expression(
  pre={
  mydata<-list()
  ifnull <- function(r,def=NA) if(!is.null(r))  r else NA
 },reduce={
  mydata<-append(mydata,reduce.values)
 },post={
   mydata<-unlist(mydata)
   in.start<-ifnull(mydata['in.start'])
   .....
   out.end<-ifnull( mydata['out.end'] )
   out.start<-ifnull(mydata['out.start'])
   value<-c(in.start,in.end,in.dur,in.pkt,out.start,out.end,out.dur,out.pkt)
   rhcollect(reduce.key,value)
 })
```

VoIP: Creating New Objects

We need to create a database of transmission intervals and jitter for every the packet in the transmission interval.
Use RHIPE to compute and store transmission intervals and the jitter associated with packets as data frames(called *jitter objects*)

VoIP: Creating New Objects

We need to create a database of transmission intervals and jitter for every the packet in the transmission interval.
Use RHIPE to compute and store transmission intervals and the jitter associated with packets as data frames(called *jitter objects*)

6.5 minutes to create 21 gigabytes of data: 14 million transmission intervals across 8 computers

VoIP: Statistical Routines Across Subsets

Jitter consists of a gateway effect and network queuing component.
Need to remove the gateway effect

- Compute bisquare robust regression on some of the jitter objects and return residuals
- Only on those with more than 90 packets
- **Just a map**: Apply a function to every jitter object:

VoIP: Statistical Routines Across Subsets

Jitter consists of a gateway effect and network queuing component.
Need to remove the gateway effect

- Compute bisquare robust regression on some of the jitter objects and return residuals
- Only on those with more than 90 packets
- **Just a map**: Apply a function to every jitter object:

- Run a mapreduce across 14 million jitter objects
- If the jitter object has more than 90 packets compute the regression
- Save the residuals(data frame) and a few other variables(saved as attributes to the data frame)

3.8 million (14 gigabytes) regressions complete in **12 mins 30 secs**

VoIP: Statistical Routines Across Subsets

... Question 1 Does the jitter depend on the traffic rate?

- Need to partition the jitter objects, sample and compute $\sim$ 5000 regressions
- Combine the results

Ongoing research

Are results as accurate as a regression using the entire data?

Another Example

Dept. Homeland Security project for a rules based statistical algorithm to detect presence of keystrokes in an SSH connection.

- 145 gigabytes of data, 1.2 million connections
- Compute summaries of connections
- Apply keystroke algorithm to more than 1.1 million connections
- Visualize a small sample using displays created with R

Example Compute total bytes, total packets across all HTTP and SSH connections.

```
m <- expression({
  w <- lapply(map.values,function(r)
              if(any(r[1,c('sport','dport')] %in% c(22,80))) T else F)
  lapply(map.values[w],function(v){
    key <- if(22 %in% v[1,c('dport','sport')]) 22 else 80
    rhcollect(key, c(sum(v[,'datasize']),nrow(v)))
  })})

r <- expression(
  pre <- { sums <- 0 }
  reduce <-{
    v <- do.call("rbind",reduce.values)
    sums  <- sums+apply(v,2,sum)
 },post={
    rhcollect(reduce.key,c(bytes=sums[1],pkts=sums[2]))
  })
```

RHIPE on EC2

Easy to start an EC2 cluster using Cloudera's distribution and shell script.

RHIPE on EC2

Easy to start an EC2 cluster using Cloudera's distribution and shell script.
Start 20 `c1.xlarge` nodes and login

```
python hadoop-ec2 launch-cluster --env REPO=testing
                     --env HADOOP_VERSION=0.20 test2 20
python hadoop-ec2 login test2
R
```

Can automatically install various R packages on the computers

Timings on EC2

An application of syndromic surveillance across time and space
State of Indiana Bio-Terrorism project

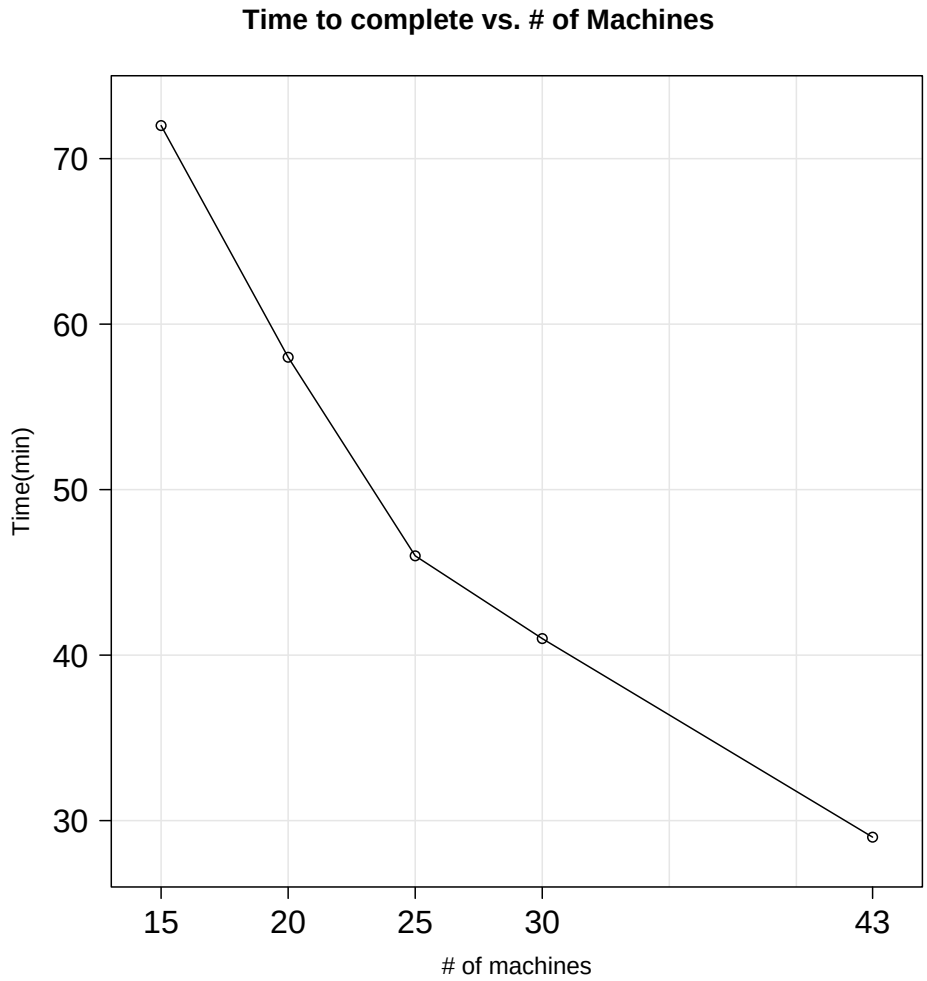
- Approximately 145 thousand simulations. Each simulation takes a list of variables as input
- Chunk size: 141 trials per task
- EC2 machine type: `c1.xlarge` each with 8 simultaneous R processes

Timings on EC2

An application of syndromic surveillance across time and space
State of Indiana Bio-Terrorism project

- Approximately 145 thousand simulations. Each simulation takes a list of variables as input
- Chunk size: 141 trials per task
- EC2 machine type: `c1.xlarge` each with 8 simultaneous R processes

```
library(Rhipe)
load("ccsim.Rdata")
rhput("/root/ccsim.Rdata","/tmp/")
setup <- expression({
  load("ccsim.Rdata")
  suppressMessages(library(survstl))
  suppressMessages(library(stl2))
})
chunk <- floor(length(simlist)/ 141)
z <- rhlapply(a,cc_sim, setup=setup,N=chunk,shared="/tmp/ccsim.Rdata"
              ,aggr=function(x) do.call("rbind",x),doLoc=TRUE)
rhex(z)
```


Time to complete vs. # of Machines
70
60
50
40
30
Time(min)
15
20
25
30
43
# of machines

Todo

- Improve error handling
- Rewrite using JNI?
- Better handling of the `combiner`
- Programatically inspect user code to determine dependencies
- Move to Avro serialization?
- Provide unique ID's to each mapper and reducer (for random number seeds)
- Improve documentation
- HBase/Hypertable integration ?

Lessons Learned

- Vector operations all the way, eschew `for`
- Not to expect *on-demand* performance with Hadoop
- MapReduce may not be the most efficient way to implement statistical algorithms
- For simulations, be sure running time of trial $>$ system overhead time