Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

Big Data "Intro to R" Bootcamp

Afternoon Session

AMANDA WHITE, TONY BLADEK, LANDON SEGO, RYAN HAFEN

Pacific Northwest National Laboratory

24 July 2014

Agenda

Schedule

Morning (9:00 a.m.-12:00 p.m.)

- Bootcamp introduction-administrivia (0.5 hrs.)
- Session 1: R Introduction/Fundamentals (2 hrs.)
- Session 2: Tessera Introduction (0.5 hrs.)

Lunch (12:00 p.m.-1:00 p.m.)

Graphs 101 (LLNL) (1:00-1:30 p.m.)

Afternoon (1:30 p.m.-4:30 p.m.)

- Session 3: Introduction to Tessera tools with data (1.0 hrs.)
- Session 4: Using Tessera with Hadoop to analyze large data (1.0 hrs.)
- Session 5: Using Trelliscope with large data (0.5 hrs)
- Session 6: Summary/Feedback (0.5 hrs.)

Bootcamp Afternoon Session

- In the morning session we
  - Explored the R tool for statistical data analysis
  - Reviewed basic R functions and data types
  - Learned about R contributed packages and how to get them
  - Learned about Tessera tools (including DataDR and Trelliscope) for big data analysis in R
    - Open source and freely available
    - Developed by data scientists
    - Works with a variety of hardware platforms from desktop to large clusters

- **Afternoon Session Goal**:
  To gain hands-on experience using R, DataDR and Trelliscope on two realistic datasets

Session 3

- Session 3: Introduction to Tessera tools with an example dataset from the 1994 census
  (1:30 – 2:30 pm)

Session 3 Objectives

- In this session we will
  - Learn about DataDR data objects
  - Create and explore Distributed Data Frames
  - Explore how to use divide and recombine methods for exploratory data analysis
  - Create Trelliscope displays
  - Use Trelliscope to visually explore data

Dataset 1: Adult Census Data

- US Census data from 1994
- Fields include
  - age
  - race
  - gender
  - education level
  - income bin (<=$50K, >$50K)
  - hours per week worked
  - job class (e.g. private, government, self-employed, etc)
- 32561 records
- Data available
  - Packaged with the DataDR library (http://tesseradata.org/datadr)
  - Also at UCI Machine Learning Repository: http://archive.ics.uci.edu/ml/datasets/Adult

Set Up Environment and Load Data (Activity 2.1.1)


```
# load libraries
library(datadr)
library(trelliscope)

# look at the dataset
str(adult)
```

```
'data.frame':  32561 obs. of  16 variables:
 $ age           : int  39 50 38 53 28 37 49 52 31 42 ...
 $ workclass     : Factor w/ 9 levels "?","Federal-gov",..: 8 7 5 5 5 5 5 7 5 5 ...
 $ fnlwgt        : int  77516 83311 215646 234721 338409 284582 160187 209642 45781 ...
 $ education     : Factor w/ 16 levels "10th","11th",..: 10 10 12 2 10 13 7 12 13 10 ...
 $ educationnum  : int  13 13 9 7 13 14 5 9 14 13 ...
 $ marital       : Factor w/ 7 levels "Divorced","Married-AF-spouse",..: 5 3 1 3 3 3 ...
 $ occupation    : Factor w/ 15 levels "?","Adm-clerical",..: 2 5 7 7 11 5 9 5 11 5 ...
 $ relationship  : Factor w/ 6 levels "Husband","Not-in-family",..: 2 1 2 1 6 6 2 1 ...
 $ race          : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...
 $ sex           : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 1 2 1 2 ...
 $ capgain       : int  2174 0 0 0 0 0 0 0 14084 5178 ...
 $ caploss       : int  0 0 0 0 0 0 0 0 0 0 ...
 $ hoursperweek  : int  40 13 40 40 40 40 16 45 50 40 ...
 $ nativecountry: Factor w/ 42 levels "?","Cambodia",..: 40 40 40 40 6 40 24 40 40...
 $ income        : Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1 2 2 2 ...
 $ incomebin     : num  0 0 0 0 0 0 0 1 1 1 …
```

Create Distributed Data Frame (Activity 2.1.2)

- Transform adult dataset into a Distributed Data Frame (DataDR data type)

```
# express data as a local "distributed data frame"
adultDdf <- ddf(adult, update=TRUE)

adultDdf
Distributed data object of class 'kvMemory' with attributes:

'ddo' attribute | value
----------------+-----------------------------------------
 keys           | keys are available through getKeys(dat)
 totStorageSize | 2.12 MB
 totObjectSize  | 2.12 MB
 nDiv           | 1
 splitSizeDistn | use splitSizeDistn(dat) to get distribution
 example        | use kvExample(dat) to get an example subset
 bsvInfo        | [empty] no BSVs have been specified

'ddf' attribute | value
----------------+-----------------------------------------
 vars           | age(int), workclass(fac), and 14 more
 transFn        | identity (original data is a data frame)
 nRow           | 32561
 splitRowDistn  | use splitRowDistn(dat) to get distribution
 summary        | use summary(dat) to see summaries

In-memory data connection
```

Explore Distributed Data Frame (Activity 2.1.3)

- A good place to start in an exploratory analysis is to look at the summary statistics

```
# look at data summary statistics
summary(adultDdf)

#results
 age                  workclass                   fnlwgt
 -------------------  --------------------------  -------------------
  missing :          0          levels : 9         missing :         0
      min :         17         missing : 0             min :     12285
      max :         90     > freqTable head <          max :   1484705
     mean :   38.58165            Private : 22696     mean : 189778.4
  std dev :   13.64043  Self-emp-not-inc :  2541   std dev :    105550
 skewness : 0.5587176         Local-gov :  2093  skewness : 1.446913
 kurtosis :  2.833714                  ? :  1836  kurtosis : 9.217672
 -------------------  --------------------------  -------------------
```

Divide Data on Education (Activity 2.1.4)

- A typical analysis path is to divide data based on some variable in the dataset

```
# divide the DDF by the variable "education"
byEducation <- divide(adultDdf, by = "education")

getKeys(byEducation)
[[1]]
[1] "education=10th"
[[2]]
[1] "education=11th"
[[3]]
[1] "education=12th"
[[4]]
[1] "education=1st-4th"
[[5]]
[1] "education=5th-6th"
...
```

Divide Data on Education (Activity 2.1.5)

- Look at data associated with one value of 'education'

```
# look at subsets of the DDF
head(byEducation[["education=Bachelors"]][[2]])
```

```
  age         workclass fnlwgt educationnum              marital
1  39         State-gov  77516           13        Never-married
2  50 Self-emp-not-inc  83311           13   Married-civ-spouse
3  28           Private 338409           13   Married-civ-spouse
4  42           Private 159449           13   Married-civ-spouse
5  30         State-gov 141297           13   Married-civ-spouse
6  23           Private 122272           13        Never-married
       occupation  relationship                 race    sex capgain caploss
1    Adm-clerical Not-in-family                White   Male    2174       0
2 Exec-managerial       Husband                White   Male       0       0
3  Prof-specialty          Wife                Black Female       0       0
4 Exec-managerial       Husband                White   Male    5178       0
5  Prof-specialty       Husband Asian-Pac-Islander   Male       0       0
6    Adm-clerical     Own-child                White Female       0       0
...
```

Use Recombine (Activity 2.1.6)

- Use recombine function to calculate summary statistics on each division and combine all results into a single table

```
# compute mean and standard dev of hoursperweek
# for each subset and recombine into one table
edMeanAndVar <- recombine(byEducation,
    apply = function(x) {
       list(mean=mean(x$age, na.rm=TRUE),
        stdev=sqrt(var(x$age, na.rm=TRUE)))
    },
    combine = combRbind()
)

#look at results
edMeanAndVar

 education val.mean val.stdev
1          10th 37.42980  16.72071
2          11th 32.35574  15.54548
3          12th 32.00000  14.33463
4       1st-4th 46.14286  15.61562
...
```

Divide Data by Work Class (Activity 2.1.7)

- Divide data according to work class

```
# divide by work class
byWorkClass <- divide(adultDdf, by="workclass")

getKeys(byWorkClass)

[[1]]
[1] "workclass=?"

[[2]]
[1] "workclass=Federal-gov"

[[3]]
[1] "workclass=Local-gov"

[[4]]
[1] "workclass=Never-worked"
```

Trelliscope Displays

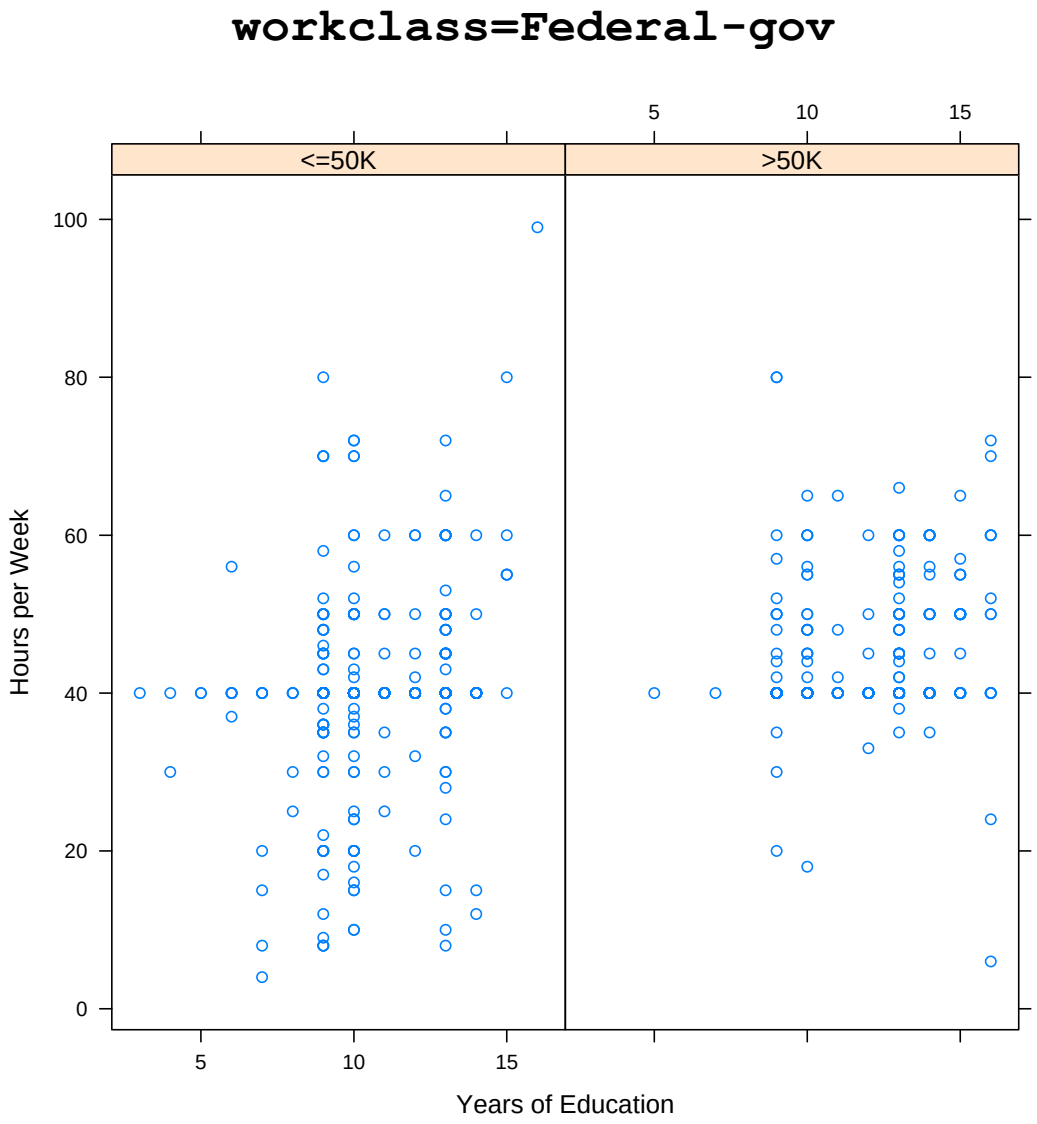
- In this activity we will make a couple of simple Trelliscope displays
- Trelliscope
  - Provides a visual recombination approach to Divide and Recombine.
  - We provide a function to be applied to each subset of data which produces a plot.
  - Each plot is called a panel
  - A collection of panels is called a display
  - A collection of displays is called a “visualization database” (VDB)
  - See (http://tesseradata.org/trelliscope) for more details about Trelliscope

Plot Function (Activity 2.1.8)

- Develop a function to make a plot, to be used on all data divisions

```
# panel function: education vs hours per week,
# conditioned on income category
pf1 <- function(x)
   xyplot(hoursperweek ~ educationnum | income,
      data = x,
      xlab="Years of Education",
      ylab="Hours per Week"
   )

# test it on one data division
pf1(byWorkClass[["workclass=Federal-gov"]][[2]])
```

Plot Function Result



Cognostics

- To help interact with all the possible panels we can specify metrics to compute for each subset
- The metrics are called ‘cognostics’
- Cognostics specify sorting and filtering of the plots based on the metrics chosen
- An example is on the next slide. The metrics are summary statistics of the education level and hours per week (mean, max, min)

Create Cognostics Function (Activity 2.1.9)


```
# cognostics function
cf <- function(x) {
   list(
      fracAbove50K = cog(mean(x$incomebin),
         desc = "fraction above $50K income"),
      minEducation = cog(min(x$educationnum),
         desc = "min number of education years"),
      meanEducation = cog(mean(x$educationnum),
         desc = "mean number of education years"),
      maxEducation = cog(max(x$educationnum),
         desc = "max number of education years"),
      minnHoursPerWeek = cog(min(x$hoursperweek),
         desc="min hours per week worked"),
      meanHoursPerWeek = cog(mean(x$hoursperweek),
         desc="mean hours per week worked"),
      maxHoursPerWeek = cog(max(x$hoursperweek),
         desc="max hours per week worked")
   )
}
```

Test Cognostics Function (Activity 2.1.10)


```
# test cognostics function on 1 data division
cf(byWorkClass[["workclass=Federal-gov"]][[2]])

#results
$fracAbove50K
[1] 0.3864583
$minEducation
[1] 3
$meanEducation
[1] 10.97396
$maxEducation
[1] 16
$minnHoursPerWeek
[1] 4
$meanHoursPerWeek
[1] 41.37917
$maxHoursPerWeek
[1] 99
```

Create and Launch Trelliscope Display (Activity 2.1.11)

- Use the panel and cognostics functions to create a Trelliscope display

```
# create visualization database
vdbConn("vdb_census", autoYes = TRUE)

# create display with this panel and cognostics functions
makeDisplay(byWorkclass,
    name = "hours_and_education_by_workclass",
    desc = "hours and education by workclass",
    panelFn = pf1, cogFn = cf,
    width = 400, height = 400
)

# view the display
myport <- Sys.getenv("TR_PORT")
view(port=myport)
```

Add a Second Type of Plot to Trelliscope (Activity 2.1.12)

- Make boxplots of education for the same dataset

```
pf2 <- function(x)
   boxplot(educationnum ~ income, data = x,
      xlab="Income", ylab="Education")

# a second type of plot on the same data
makeDisplay(byWorkClass,
   name = "education_boxplot_by_workclass",
   desc = "education boxplot by workclass",
   panelFn = pf2, cogFn = cf,
   width = 400, height = 400
)

view(port=myport)
```

Challenge

- Part 1:
  Can you construct a dataset like **`byWorkclass`** divided on another variable?
  - Hint: use **`names(adultDdf)`** to see other variables
- Part 2:
  Now use one of the two existing panel and the cognostics functions to create a new Trelliscope display.

Challenge Solution

- Challenge solution: (one of many)

```
# divide by race field
byRace <- divide(adultDdf, by="race")

makeDisplay(byRace,
    name = "hours_and_education_by_race",
    desc = "hours and education by race",
    panelFn = pf1, cogFn = cf,
    width = 400, height = 400
)

# launch viewer
view(port=myport)
```

Session 3 Review

- At the end of this session you should understand
  - Why you would want to create a DDF and how to interact with it (Slides 8-9)
  - How and why to use the divide and recombine methods (Slides 10-13)
  - How to create Trelliscope displays (Slides 14-23)
  - How to use Trelliscope to visually explore data

Break: 2:30-2:40 p.m.

- Break: 2:30-2:40 p.m.

- Next:
  Session 4: Using Tessera with Hadoop to analyze large data

Session 4

- Session 4: Using Tessera with Hadoop to analyze large data (2:40 – 3:20 pm)

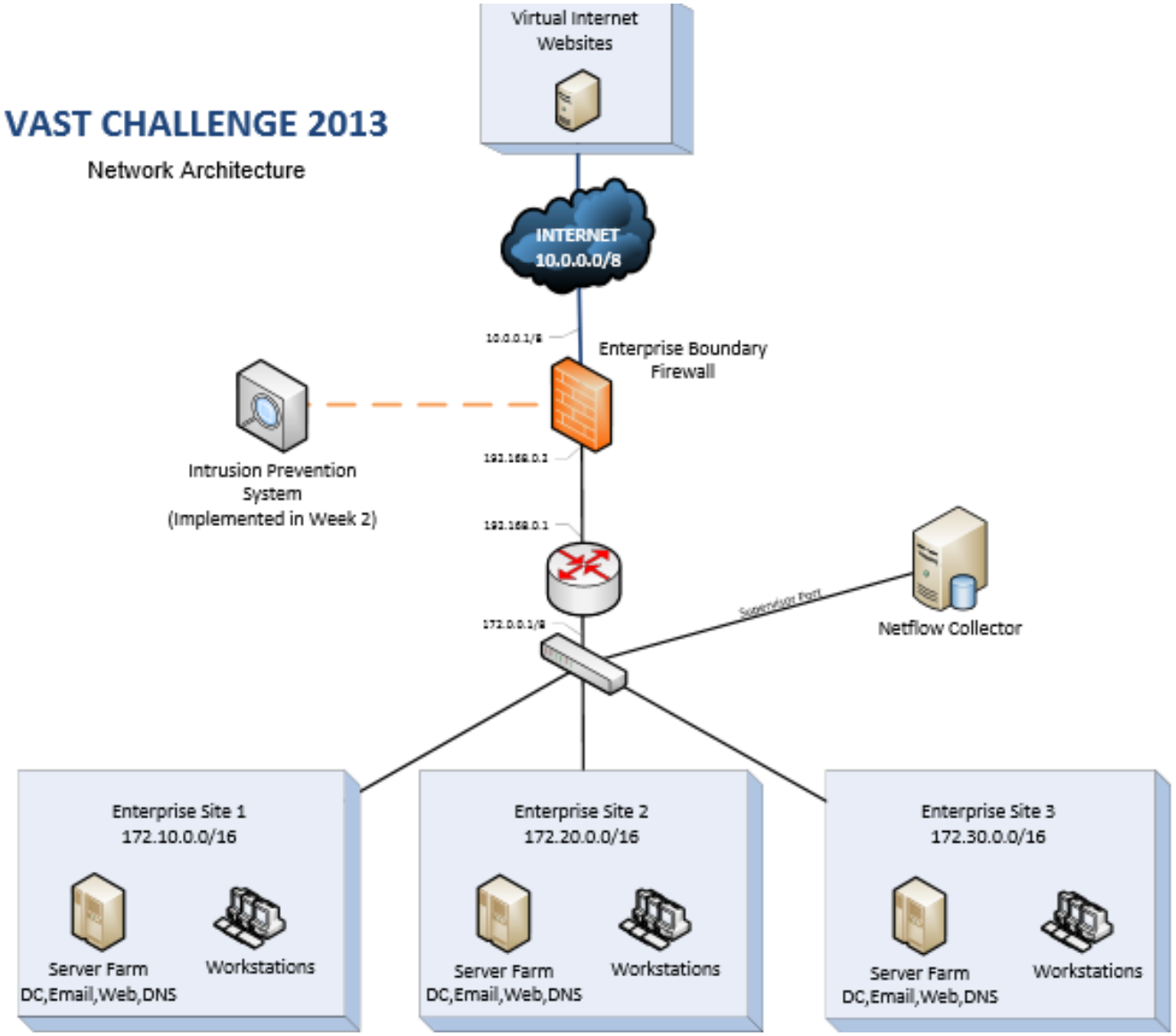
Session 4 Objectives

- In this session we will:
  - Learn how to use DataDR to analyze large datasets located in Hadoop Distributed File System (HDFS)
  - Learn how to load NetFlow dataset using distributed processing
  - Explore other DataDR analysis methods (e.g. drAggregate and drQuantile)

Dataset 2: Network Traffic

- **Visual Analytics Science and Technology (VAST) challenge**
- **Simulated network traffic dataset**
- **NetFlow Data**
  - Network flow data captures, to the extent feasible, the traffic moving across the network.
  - NetFlow data is captured at the firewall, so transactions that go from the company to the internet, or come from the internet into the company, are captured.
  - NetFlow data is a series of messages between two computers which are
    - Combined into a single flow record.
    - Records appear for each session where the handshake between the two computers is completed.
    - Includes a source and destination IP but
      - the designation of source and destination are not guaranteed to be correct.
    - Includes the total number of packets sent/received, total bytes sent/received, internal protocol used (the two most common are TCP and UDP), etc.
- Additional details of this example can be found at http://tesseradata.org/example-vast-challenge/

Data Exploration: VAST Data Challenge Network Diagram



Example 2 Environment Setup (Activity 2.2.1)

- Environment Setup

```
# load required packages and initialize Rhipe
library(datadr)
library(Rhipe)
library(cyberTools)
rhinit()

# set time zone to "UTC" for use with dates in the data
Sys.setenv(TZ = "UTC")

# set working directories on local machine and HDFS
setwd(”~")

# set working directory in HDFS
hdfs.setwd(Sys.getenv("HDFS_USER_VAST"))
```

Exploring HDFS (Activity 2.2.2)

- Look for data file in Hadoop Distributed File System

```
# make sure raw text data has been copied to HDFS
rhls("raw/nf")

# Example output (Content will vary per user)
permission   owner      group      size       modtime
1 drwxrwxrwx garfunkel supergroup 0          2014-04-28 20:27
2 -rw-rw-rw- garfunkel supergroup 2.766 gb   2014-04-24 19:57
file
1 /user/garfunkel/vast/raw/nf/_rh_meta
2 /user/garfunkel/vast/raw/nf/nf-week2.csv
```

Loading Dataset in DataDR

- Read NetFlow csv (format) Data to R Objects
  - Data formatting for analysis can be tedious
  - DataDR tries to make this as painless as possible.
  - Data needs to be stored as native R objects for most efficient processing.
    - Many native data structures in R allow for lots of flexibility
  - Specific case
    - All input (NetFlow) data is text and is stored in a Comma Separated Format (.csv)
    - DataDR provides an extension to a native R function (`read.csv`) to read this type of data (`drRead.csv`)

NetFlow Data

	A	B	C	D	E	F	G	H	I	
1	TimeSeconds	parsedDate	dateTimeStr	ipLayerProto	ipLayerProto	firstSeenSrcI	firstSeenDes	firstSeenSrcF	firstSeenDes	more
2	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.19	239.255.255.	29987	1900	
3	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.18	239.255.255.	29986	1900	
4	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.17	239.255.255.	29985	1900	
5	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.16	239.255.255.	29984	1900	
6	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.14	239.255.255.	29983	1900	
7	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.13	239.255.255.	29982	1900	
8	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.12	239.255.255.	29981	1900	
9	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.11	239.255.255.	29980	1900	
10	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.10	239.255.255.	29979	1900	
11	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.35	239.255.255.	29978	1900	
12	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.34	239.255.255.	29977	1900	
13	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.37	239.255.255.	29976	1900	
14	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.36	239.255.255.	29975	1900	
15	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.31	239.255.255.	29974	1900	
16	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.30	239.255.255.	29973	1900	
17	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.33	239.255.255.	29972	1900	
18	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.32	239.255.255.	29971	1900	
19	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.39	239.255.255.	29970	1900	
20	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.38	239.255.255.	29969	1900	
21	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.9	239.255.255.	29968	1900	
22	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.8	239.255.255.	29967	1900	
23	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.5	239.255.255.	29966	1900	
24	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.4	239.255.255.	29965	1900	
25	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.7	239.255.255.	29964	1900	
26	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.6	239.255.255.	29963	1900	
27	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.1	239.255.255.	29962	1900	
28	1365582756	4/10/13 8:32	20130410083236.3	17	UDP	172.20.2.3	239.255.255.	29961	1900	

Preparing to Read Data from File (Activity 2.2.3)

- Read NetFlow csv (format) Data to R Objects
  - Now lets make the previous operations into a function so we can reuse it as each row of data is read.

```
# make a date parsing function to use during data ingest
nfTransform <- function(x) {
   x$date <- as.POSIXct(as.numeric(as.character(x$TimeSeconds)),
      origin = "1970-01-01", tz = "UTC")
   x[,setdiff(names(x), c("TimeSeconds", "parsedDate",
      "dateTimeStr"))]
}

# initiate a connection to existing csv text file on HDFS
csvConn <- hdfsConn("raw/nf", type = "text")

# initiate a new connection where parsed NetFlow data will be stored
nfConn <- hdfsConn("nfRaw", autoYes=TRUE)
```

Read Data and Store as R Objects (Activity 2.2.4)

- **`drRead.csv`** similar to **`read.csv`** with same arguments
  - Where to store the outputs
  - How many rows to put in each block
  - Optional transformational function to apply to each block.
- Read is implemented in Map-Reduce: file contents are divided into chunks and sent different processors to ingest and store as R objects

```
# read in NetFlow data

nfRaw <- drRead.csv(csvConn, output = nfConn, postTransFn =
    nfTransform)
```

- Comparison: the data read/ingest step takes 5 min 53 sec when using local disk/single processor rather than HDFS/parallel processing

Distributed Data Objects (Activity 2.2.5)

- Now that we have read in the data, let's look at the structure

```
# look at the nfRaw object
nfRaw

#Results
Distributed data object of class 'kvHDFS' with attributes:

'ddo' attribute | value
----------------+-----------------------------------------------
 keys           | [empty] call updateAttributes(dat) to get this value
 totStorageSize | 1.39 GB
 totObjectSize  | [empty] call updateAttributes(dat) to get this value
 nDiv           | [empty] call updateAttributes(dat) to get this value
 splitSizeDistn | [empty] call updateAttributes(dat) to get this value
 example        | use kvExample(dat) to get an example subset
 bsvInfo        | [empty] no BSVs have been specified
```

Distributed Data Objects Continued (Activity 2.2.5)

- Note that nfRaw is a Distributed Data Frame (DDF)
  - **`nfRaw`** is a special R object that contains metadata and pointers to the actual data
  - Other attributes will be updated shortly.

```
#Results (continued from previous slide)

'ddf' attribute | value
----------------+-----------------------------------------------
 vars           | ipLayerProtocol(int) and 16 more
 transFn        | identity (original data is a data frame)
 nRow           | [empty] call updateAttributes(dat) to get this value
 splitRowDistn  | [empty] call updateAttributes(dat) to get this value
 summary        | [empty] call updateAttributes(dat) to get this value

hdfsConn connection
  loc=/user/simon/bootcamp/nfRaw; type=sequence
```

Distributed Data Objects (Activity 2.2.6)

- Getting Familiar with Distributed Data Objects
  - We can update the missing attributes using the **`updateAttributes`** function

```
# update attributes to calculate summary statistics
nfRaw <- updateAttributes(nfRaw)
```

The R data with summary statistics and other metadata are stored on HDFS

```
# nfRaw can be reloaded in any subsequent R session using the
# following command
nfRaw <- ddf(hdfsConn("nfRaw"))
```

Look at Data Summary (Activity 2.2.7)

- Let's look at the data again after updating the attributes
  - Note that it has a total size of 1.83 GB and there are 481 divisions

```
nfRaw

Distributed data object of class 'kvHDFS' with attributes:

'ddo' attribute | value
----------------+-----------------------------------------
 keys           | keys are available through getKeys(dat)
 totStorageSize | 1.39 GB
 totObjectSize  | 1.83 GB
 nDiv           | 481
 splitSizeDistn | use splitSizeDistn(dat) to get distribution
 example        | use kvExample(dat) to get an example subset
 bsvInfo        | [empty] no BSVs have been specified
```

Look at Data Summary Continued (Activity 2.2.7)

- Let's look at the data again after updating the attributes
  - Note that it has 23 million rows and 17 variables

```
# continued from previous slide

'ddf' attribute | value
----------------+-----------------------------------------
 vars           | ipLayerProtocol(int) and 16 more
 transFn        | identity (original data is a data frame)
 nRow           | 23258685
 splitRowDistn  | use splitRowDistn(dat) to get distribution
 summary        | use summary(dat) to see summaries

hdfsConn connection
  loc=/user/simon/bootcamp/nfRaw; type=sequence
```

Getting Familiar with DDFs (Activity 2.2.8)

- Let's look at some more aspects of the data.
  - We can see the number of rows and the variable names

```
# get total number of rows
nrow(nfRaw)

#Results
[1] 23258685

# see what variables are available
names(nfRaw)

#Results
 [1] "ipLayerProtocol"           "ipLayerProtocolCode"       "firstSeenSrcIp"
 [4] "firstSeenDestIp"           "firstSeenSrcPort"          "firstSeenDestPort"
 [7] "moreFragments"             "contFragments"             "durationSeconds"
[10] "firstSeenSrcPayloadBytes"  "firstSeenDestPayloadBytes" "firstSeenSrcTotalBytes"
[13] "firstSeenDestTotalBytes"   "firstSeenSrcPacketCount"   "firstSeenDestPacketCount"
[16] "recordForceOut"            "date"
```

Getting Familiar with DDFs Continued (Activity 2.2.9)

- Let's look at some more aspects of the data.
  - Look at the structure of the first subset

```
# look at the structure of the first key-value pair
str(nfRaw[[1]])
```

```
# Partial Results

List of 2
 $ : chr "attempt_201402250850_1647_m_000003_0"
 $ :'data.frame':   50000 obs. of  17 variables:
  ..$ ipLayerProtocol          : int [1:50000] 6 6 6 6 6 6 6 6 6 6 ...
  ..$ ipLayerProtocolCode      : chr [1:50000] "TCP" "TCP" "TCP" "TCP" ...
  ..$ firstSeenSrcIp           : chr [1:50000] "10.38.217.48" "10.38.217.48" ...
  ..$ firstSeenDestIp          : chr [1:50000] "172.10.0.4" "172.10.0.4" "172.10.0.4" ...
  ..$ firstSeenSrcPort         : int [1:50000] 45482 44204 46121 44205 45483 44206 46604 ...
  ..$ firstSeenDestPort        : int [1:50000] 80 80 80 80 80 80 80 80 80 80 ...
  ..$ moreFragments            : int [1:50000] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ contFragments            : int [1:50000] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ durationSeconds          : int [1:50000] 5 9 3 9 5 9 2 9 5 9 ...
```

Getting Familiar with DDFs Continued (Activity 2.2.10)

- The `summary` function provides a nice overview of the variables

```
# look at summaries (computed from updateAttributes)
summary(nfRaw)

# Partial Results

 ipLayerProtocol      ipLayerProtocolCode      firstSeenSrcIp           firstSeenDestIp
 --------------------  --------------------  ----------------------  ----------------------
  missing :          0          levels : 3          levels : 1390           levels : 1277
      min :          1         missing : 0         missing : 0             missing : 0
      max :         17   > freqTable head <    > freqTable head <      > freqTable head <
     mean :   6.089594     TCP : 23062987    10.138.214.18 : 1300759   172.30.0.4 : 8122427
  std dev : 0.9961443     UDP :   191395    10.170.32.181 : 1259035   172.10.0.4 : 4652570
 skewness :  10.78649   OTHER :     4303    10.170.32.110 : 1257747   172.20.0.4 : 4341038
 kurtosis :  118.5448                        10.10.11.102 : 1251990  172.20.0.15 : 4029911
 --------------------  --------------------  ----------------------  ----------------------

...
```

Getting Familiar with DDFs Continued

- In general, a good way to start to look at large data sets in R is to look at summary statistics. The data provided with a DDF is a good place to start.

- Visualizing the data will provide many more insights into the data which will cover in the next session.

drAggregate Function (Activity 2.2.11)

- The drAggregate function counts frequencies of any combination of variables specified
- Can be used to get an idea of meaningful or interesting data divisions for further analysis

```
#aggregate by minute and IP for just "bigIPs"
bigTimeAgg <- drAggregate(~ timeMinute + firstSeenDestIp,
    data = nfRaw,
    preTransFn = function(x) {
        x <- subset(x, firstSeenDestIp %in% bigIPs)
        x$timeMinute <- as.POSIXct(trunc(x$date, 0, units="mins"))
        x
    })

# sort the result by IP and time
bigTimeAgg <- bigTimeAgg[order(bigTimeAgg$Freq, decreasing=TRUE),]
```

Aggregated Data (Activity 2.2.12)


```
# look at the first few rows
head(bigTimeAgg)

# results
                  timeMinute firstSeenDestIp   Freq
6409356 2013-04-11 12:55:00      172.30.0.4 200790
6409355 2013-04-11 12:54:00      172.30.0.4 193490
6409354 2013-04-11 12:53:00      172.30.0.4 191602
6409358 2013-04-11 12:57:00      172.30.0.4 189878
6413807 2013-04-14 15:06:00      172.30.0.4 174662
6409353 2013-04-11 12:52:00      172.30.0.4 174432
```

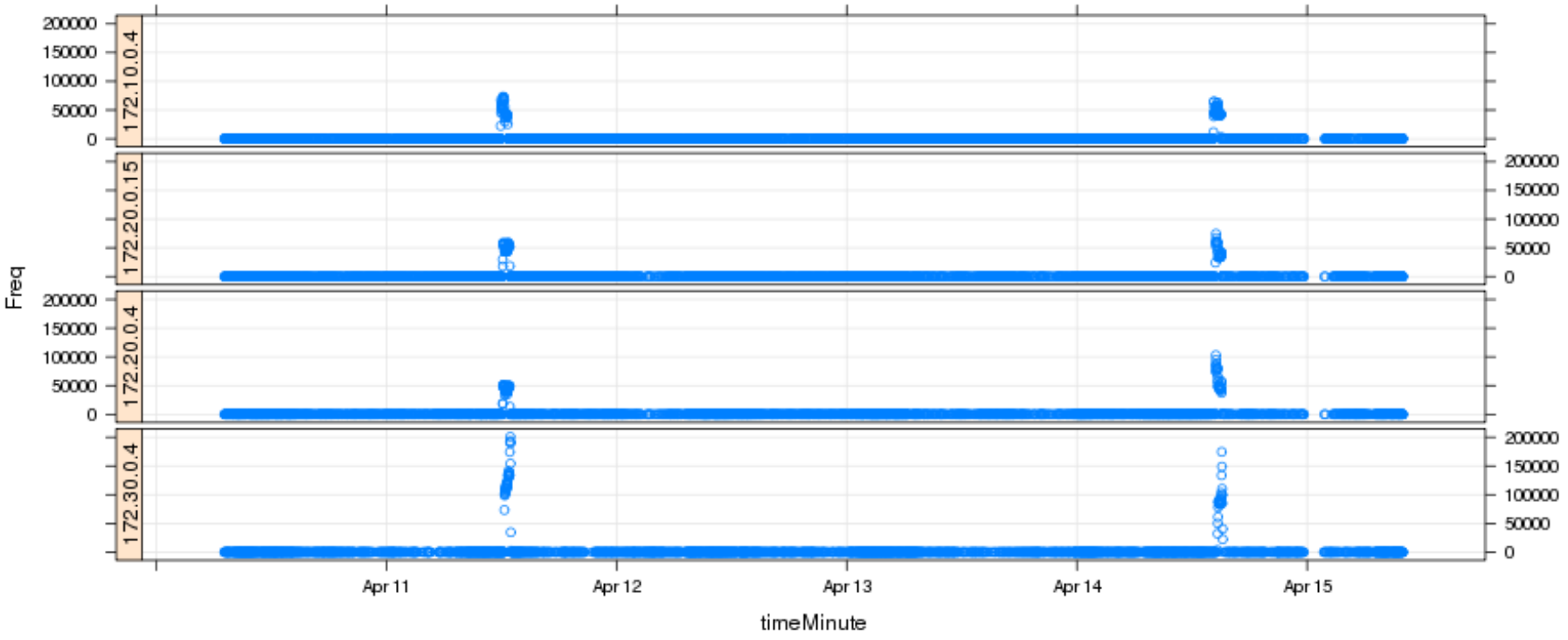
Plot Time-Aggregated Data (Activity 2.2.13)

- Plot the Time Series by Host IP

```
# convert timeMinute column to a time variable
bigTimeAgg$timeMinute <- as.POSIXct(bigTimeAgg$timeMinute,
    tz = "UTC")

# a few http servers
httpIPs <- c("172.20.0.15", "172.20.0.4", "172.10.0.4",
    "172.30.0.4")

# plot the time series of minute-counts by destination IP
xyplot(Freq ~ timeMinute | firstSeenDestIp,
        data = bigTimeAgg[bigTimeAgg$firstSeenDestIp %in%
          httpIPs,],
        layout = c(1, 4), as.table = TRUE,
        strip = FALSE, strip.left = TRUE,
        between = list(y = 0.25),
        type = c("p", "g"))
```

Plot of Destination IP Connections Over Time

- Traffic occurs in two bursts at the same time for each host
- Looks like a Denial of Service Attack.



Other uses for drAggregate

- This example used **`drAggregate`** to count sessions per minute for each source IP
- What other aggregations might be of interest?

Other uses for drAggregate

- This example used **`drAggregate`** to count sessions per minute for each source IP
- What other aggregations might be of interest?
  - Count sessions per minute (or any time interval) for each protocol code
  - Count sessions for each source and destination pair
  - Sum the bytes per time interval for each source or destination IP
  - Mean session bytes per time interval for each source or destination IP (requires 2 calls to **`drAggregate`**)

drQuantile Function (Activity 2.2.14)

- Another 'division agnostic' method is using the **drQuantile** plot to calculate data quantiles, either overall or conditioned on another variable
- Good for identifying data outliers
- Use **drQuantile** to explore the distribution of connection duration by source host type

```
# compute the distribution of connection duration
# by source host type
dsqSrcType <- drQuantile(
   nfRaw, var = "durationSeconds", by = "type",
   preTransFn = function(x) {
      suppressMessages(library(cyberTools))
      mergeHostList(x[,c("firstSeenSrcIp",
         "durationSeconds")], "firstSeenSrcIp")
   }
)
```

drQuantile Output (Activity 2.2.15)

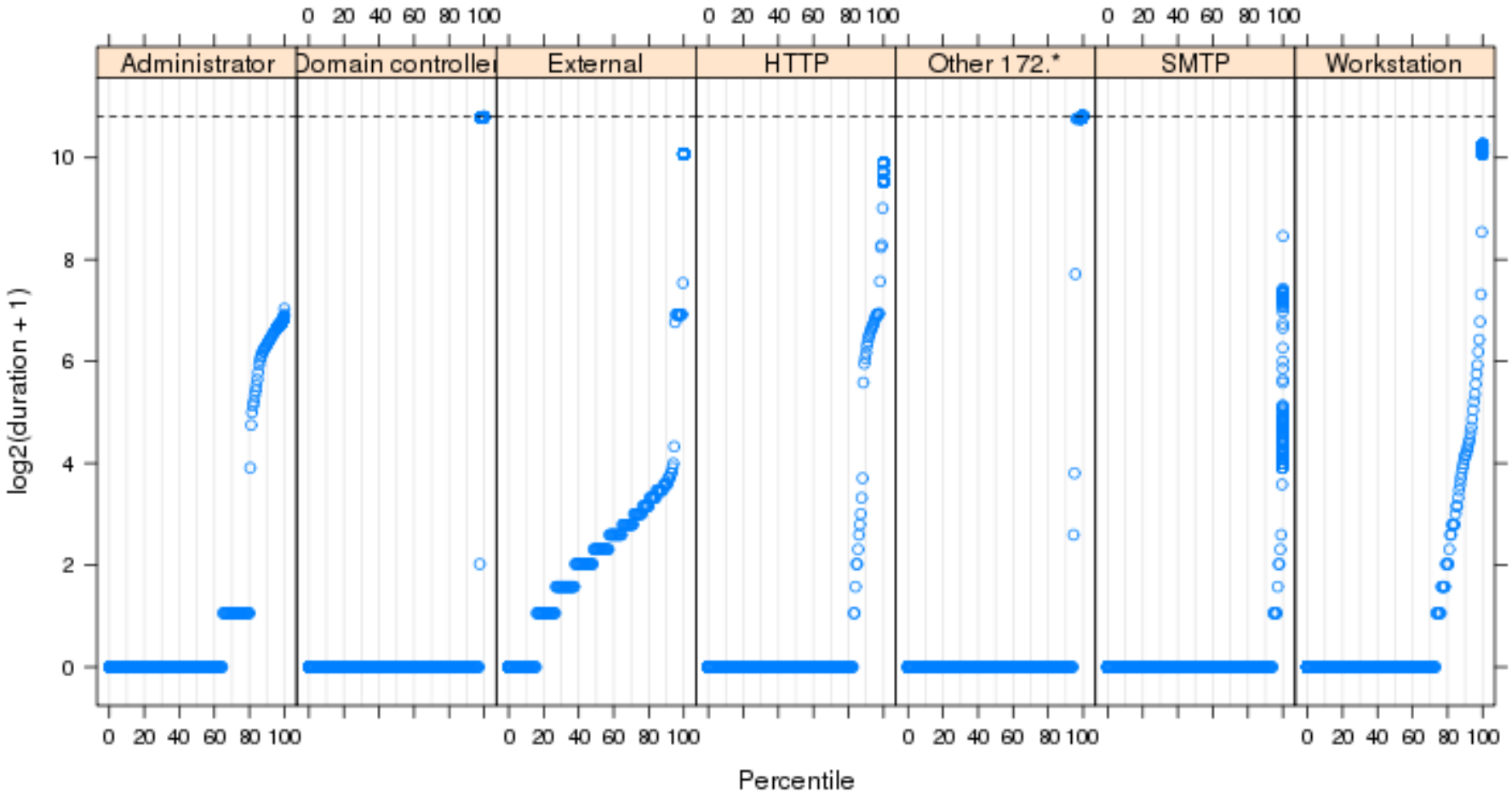

```
# look at the drQuantile output
head(dsqSrcType)

# Result
           fval q          group
1 0.0000000000 0 Administrator
2 0.0004035513 0 Administrator
3 0.0008071025 0 Administrator
4 0.0012106538 0 Administrator
5 0.0016142050 0 Administrator
6 0.0020177563 0 Administrator
```

Construct a Quantile Plot (Activity 2.2.16)

- Plot frequency quantiles by host type (e.g. HTTP, SMTP, etc)

```
# plot the distribution of connection duration by source host
# type
xyplot(log2(q + 1) ~ fval * 100 | group,
   data = dsqSrcType, type = "p",
   xlab = "Percentile",
   ylab = "log2(duration + 1)",
   panel = function(x, y, ...) {
      panel.abline(v = seq(0, 100, by = 10), col = "#e6e6e6")
      panel.xyplot(x, y, ...)
      panel.abline(h = log2(1801), lty = 2)
   },
   layout = c(7, 1)
)
```

Quantile Plot



Session 4 Review

- At the end of this session you should understand
  - How and why you would want to use DataDR with HDFS (Slides 32-38)
  - How to use division-agnostic functions in DataDR (Slides 45-54)

Break: 3:20-3:30 p.m.

- Break: 3:20-3:30 p.m.

- Next:
  Session 5: Creating and using Trelliscope displays with large data

Session 5

- Session 5: Creating and using Trelliscope displays with large data (3:30 pm – 4:00 pm)

Session 5 Objectives

- Explore the use of Trelliscope on a large complex dataset

Trelliscope for Big Data

- Trelliscope can provide an intuitive and convenient way to sort and filter through large datasets to identify data trends
- In this section we will develop a few Trelliscope displays for NetFlow data and see how they can be used to answer questions about the data

Division by Inside Host (Activity 2.3.1)

- Division by Inside Host
  - We now want to explore the behavior of all hosts inside the company network.
  - We can now go from division agnostic analysis to dividing based on internal host IP
  - First we want to remove the DDoS IP address during the attacks

```
# variables to filter on
bigTimes <- sort(unique(bigTimeAgg$timeMinute[
    bigTimeAgg$Freq > 1000]))
badIPs <- c("10.138.214.18", "10.17.15.10", "10.12.15.152",
    "10.170.32.110", "10.170.32.181", "10.10.11.102",
    "10.247.106.27", "10.247.58.182", "10.78.100.150",
    "10.38.217.48", "10.6.6.7", "10.12.14.15", "10.15.7.85",
    "10.156.165.120", "10.0.0.42", "10.200.20.2",
    "10.70.68.127", "10.138.235.111", "10.13.77.49",
    "10.250.178.101")
```

Division by Inside Host (Activity 2.3.2)

- Now divide the data by inside host using newly created variables.

```
# divide data by host IP
nfByHost <- divide(nfRaw, by = "hostIP",
           preTransFn = function(x) {
               suppressMessages(library(cyberTools))
               x$timeMinute <- as.POSIXct(trunc(x$date,
                   0, units = "mins"))
               x <- subset(x,
                   !(timeMinute %in% bigTimes &
                   firstSeenSrcIp %in% c(httpIPs, badIPs) &
                   firstSeenDestIp %in% c(httpIPs, badIPs)))
               if(nrow(x) > 0) {
                   return(getHost(x))
               } else {
                   return(NULL)
               }
           },
           output = hdfsConn("nfByHost", autoYes=TRUE)
)

nfByHost <- updateAttributes(nfByHost)
```

Division by Inside Host (Activity 2.3.3)

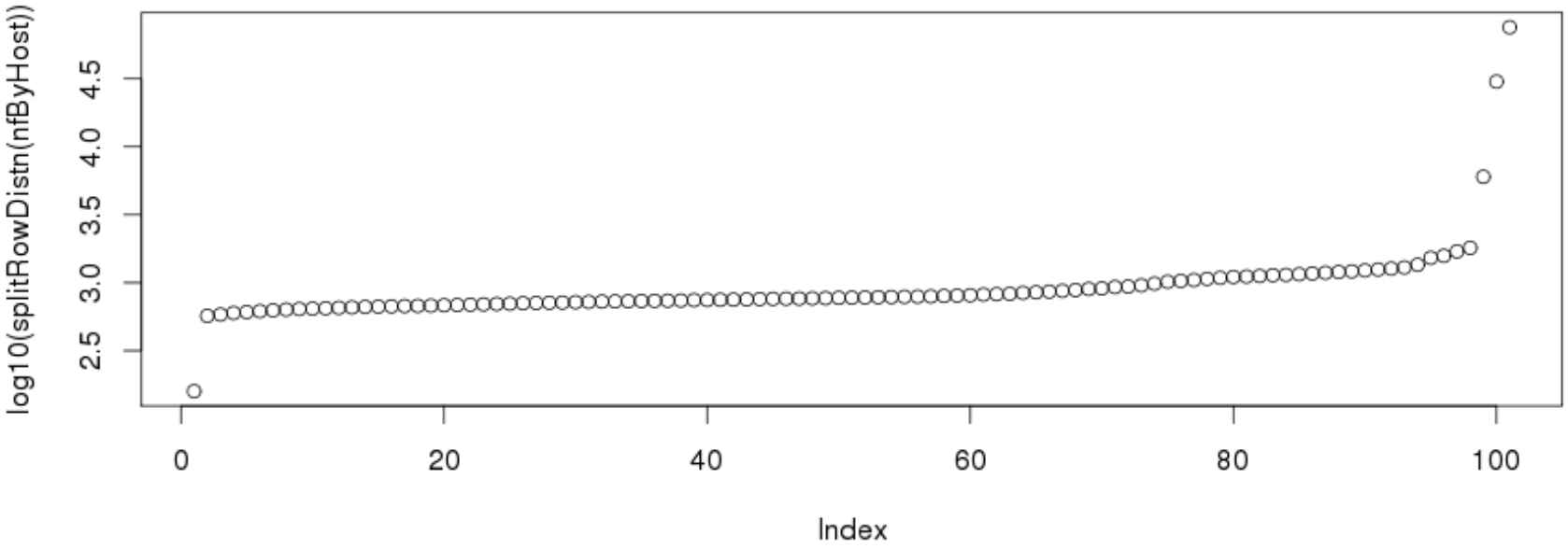
- Look at the result
  - Note it is much smaller and could be handled on a single machine.

```
# print nfByHost
nfByHost

# Partial Results
Distributed data object of class 'kvHDFS' with attributes:

'ddo' attribute | value
----------------+-------------------------------------------
 keys           | keys are available through getKeys(dat)
 totStorageSize | 135.82 MB
 totObjectSize  | 181.82 MB
 nDiv           | 1223
 splitSizeDistn | use splitSizeDistn(dat) to get distribution
 example        | use kvExample(dat) to get an example subset
 bsvInfo        | [empty] no BSVs have been specified
```

Division Size Distribution (Activity 2.3.4)

- Plot number of rows per division, note there are a few very large divisions

```
# look at the distribution of number of rows in
nfByHost plot(log10(splitRowDistn(nfByHost)))
```



Time-Aggregated Recombination (Activity 2.3.5)

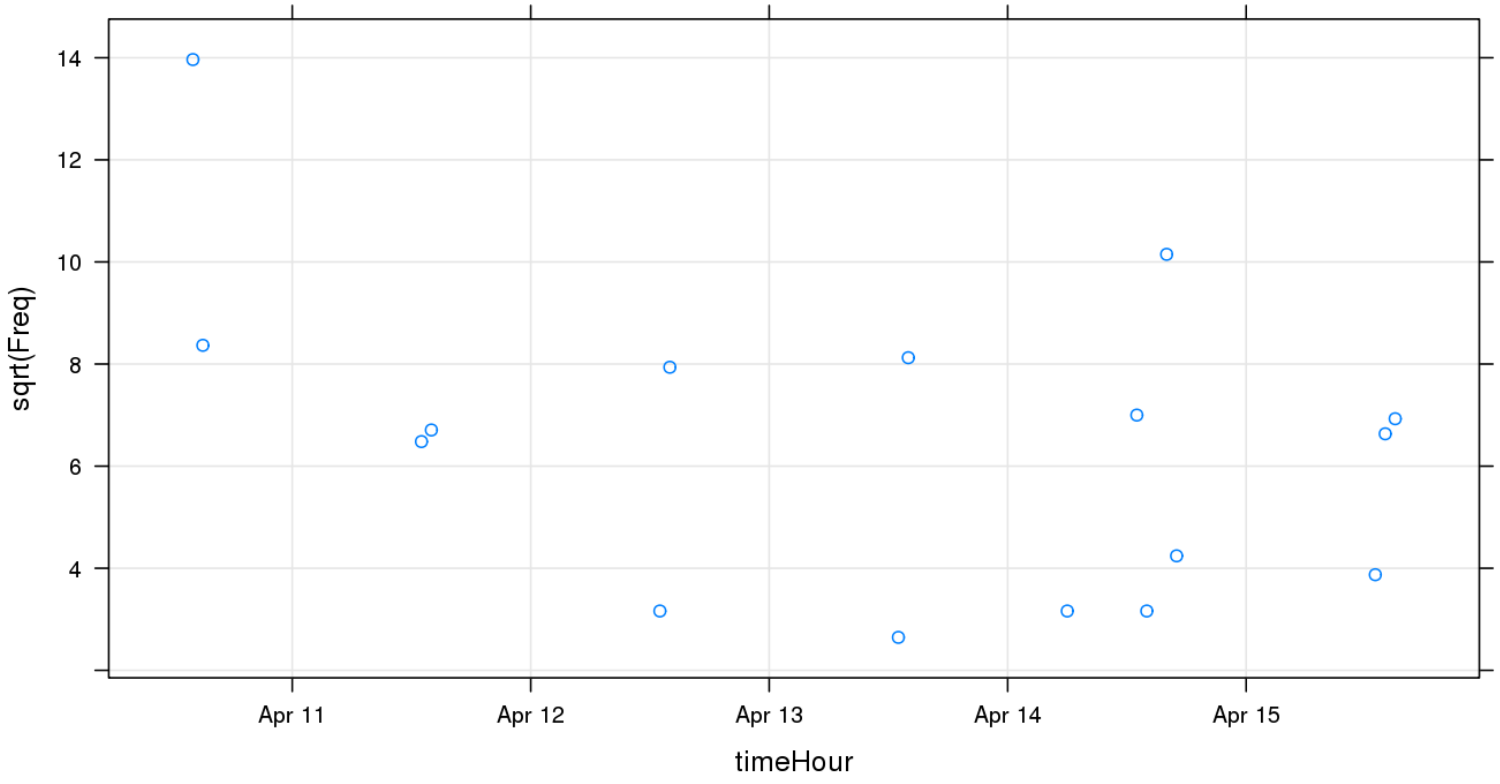
- Let's tabulate the number of connections by the hour
  - We can do this by calling the 'recombine()' function which will apply a function to each subset of the data and will combine the results using the function 'combDdo().
  - The result is a distributed data object (ddo).

```
# roll data up to counts by hour for each host
hostTimeAgg <- recombine(nfByHost,
    apply = function(x) {
        timeHour <- as.POSIXct(trunc(x$date, 0,
            units = "hours"))
        res <- data.frame(xtabs(~ timeHour))
        res$timeHour <- as.POSIXct(res$timeHour)
        res
    },
    combine = combDdo()
)
```

NetFlow Trelliscope Displays (Activity 2.3.6)


```
# load trelliscope library
library(trelliscope)

# initiate a visualization database (VDB) connection
vdbConn("vdb_vast", autoYes=TRUE)

# panel function for simple time series plot
timePanel <- function(x) {
   xyplot(sqrt(Freq) ~ timeHour, data = x,
       type = c("p", "g"))
}

# test on subset
timePanel(hostTimeAgg[[1]][[2]])
```

Panel Function Output



NetFlow Cognostics Function (Activity 2.3.7)

- The metrics are total number of connections, median and standard deviation of number of hourly counts.

```
# cognostics function for simple time series plot
timeCog <- function(x) {
   IP <- attr(x, "split")$hostIP
   curHost <- hostList[hostList$IP == IP,]
   list(
      hostName = cog(curHost$hostName, desc = "host name"),
      type = cog(curHost$type, desc = "host type"),
      nobs = cog(sum(x$Freq), "log 10 total number of connections"),
      timeCover = cog(nrow(x),
          desc = "number of hours containing connections"),
      medHourCt = cog(median(sqrt(x$Freq)),
      desc = "median square root number of connections"),
      madHourCt = cog(mad(sqrt(x$Freq)),
          desc = "median absolute deviation square root number of
connections"),
      max = cog(max(x$Freq),
          desc = "maximum number of connections in an hour")
   )
}
```

Test Cognostics Function on One Division (Activity 2.3.8)

- Test cognostics function on one division of the data

```
# test on subset
timeCog(hostTimeAgg[[1]][[2]])

$hostName
[1] "wss2-259.bigmkt2.com"
$type
[1] "Workstation"
$nobs
[1] 795
$timeCover
[1] 16
$medHourCt
[1] 6.670727
$madHourCt
[1] 3.057091
$max
[1] 195
```

Create Trelliscope Display (Activity 2.3.9)

- We can now make a display by providing panel and cognostics functions as well as additional information like a display name and description.

```
# create the trelliscope display
makeDisplay(hostTimeAgg,
    name = "hourly_count",
    group = "inside_hosts",
    desc = "time series plot of hourly counts of connections
for each inside host",
    panelFn = timePanel,
    cogFn = timeCog,
    width = 800, height = 400,
    lims = list(x = "same", y = "same")
)

# launch display
myport <- as.numeric(Sys.getenv("TR_PORT"))
view(port=myport)
```

Another Trelliscope Display (Activity 2.3.10)

- One more division we can make is to divide the data into incoming and outgoing connections.

```
# aggregate hourly counts by "incoming", "outgoing"
hostTimeDirAgg <- recombine(nfByHost,
    apply = function(x) {
        x$timeHour <- as.POSIXct(trunc(x$date, 0,
            units = "hours"))
        res <- data.frame(xtabs(~ timeHour + srcIsHost,
            data = x))
        res$timeHour <- as.POSIXct(res$timeHour)
        res$direction <- "incoming"
        res$direction[as.logical(
            as.character(res$srcIsHost))] <- "outgoing"
        subset(res, Freq > 0)
    },
    combine = combDdo())
```

Plot Incoming and Outgoing Sessions (Activity 2.3.11)

- Now make a similar panel function that separates incoming and outgoing sessions

```
# new slightly different time panel
timePanelDir <- function(x) {
   xyplot(
      sqrt(Freq) ~ timeHour,
      groups = direction,
      data = x,
      type = c("p", "g"),
      auto.key = TRUE
   )
}
```

Cognostics Function: Incoming vs Outgoing (Activity 2.3.12)

- Construct a new cognostics function that calculates all the same metrics, but divided by incoming and outgoing sessions

```
# new cognostics function
timeCog2 <- function(x) {
  IP <- attr(x, "split")$hostIP
  curHost <- hostList[hostList$IP == IP,]
  ind.incoming <- x$direction == "incoming"

  cog.values <- list(
    hostName = cog(curHost$hostName, desc = "host name"),
    type = cog(curHost$type, desc = "host type"),
     incomingNobs = cog(sum(x$Freq[ind.incoming]),
                        "log 10 total number of incoming connections"),
     outgoingNobs = cog(sum(x$Freq[!ind.incoming]),
                        "log 10 total number of outgoing connections"),
     incomingTimeCover = cog(sum(ind.incoming), desc = "number of hours containing incoming connections"),
     outgoingTimeCover = cog(sum(!ind.incoming), desc = "number of hours containing outgoing connections"),
     incomingMedHourCt = cog(median(sqrt(x$Freq[ind.incoming]), na.rm=TRUE),
                 desc = "median square root number of incoming connections"),
     outgoingMedHourCt = cog(median(sqrt(x$Freq[!ind.incoming]), na.rm=TRUE),
                     desc = "median square root number of outgoing connections"),
     incomingMadHourCt = cog(mad(sqrt(x$Freq[ind.incoming])),
                     desc = "median absolute deviation square root number of incoming connections"),
     outgoingMadHourCt = cog(mad(sqrt(x$Freq[!ind.incoming])),
                 desc = "median absolute deviation square root number of outgoing connections"),
    incomingMax = cog(max(c(0, x$Freq[ind.incoming])), desc = "maximum number of incoming connections in an hour"),
    outgoingMax = cog(max(c(0, x$Freq[!ind.incoming])), desc = "maximum number of outgoing connections in an hour")
  )
  cog.values[unlist(lapply(cog.values, is.na))] <- -1
  cog.values
}
timeCog2(hostTimeDirAgg[[1]][[2]])
```

Make Another Trelliscope Display (Activity 2.3.13)

- Use the new panel and cognostics functions to construct a new display with the data in `hostTimeDirAgg`
- This time traffic is broken out into incoming and outgoing connections

```
# create the display
makeDisplay(hostTimeDirAgg,
            name = "hourly_count_src_dest",
            group = "inside_hosts",
            desc = "time series plot of hourly counts of connections
for each inside host by source / destination",
            panelFn = timePanelDir,
            width = 800, height = 400,
            cogFn = timeCog2,
            lims = list(x = "same", y = "same"))

# view display
view(port=myport)
```

Trelliscope Challenge

- There is a set of workstations that were infected with malware and now form a botnet*. This botnet has started a recurring Denial of Service attack against an external IP.

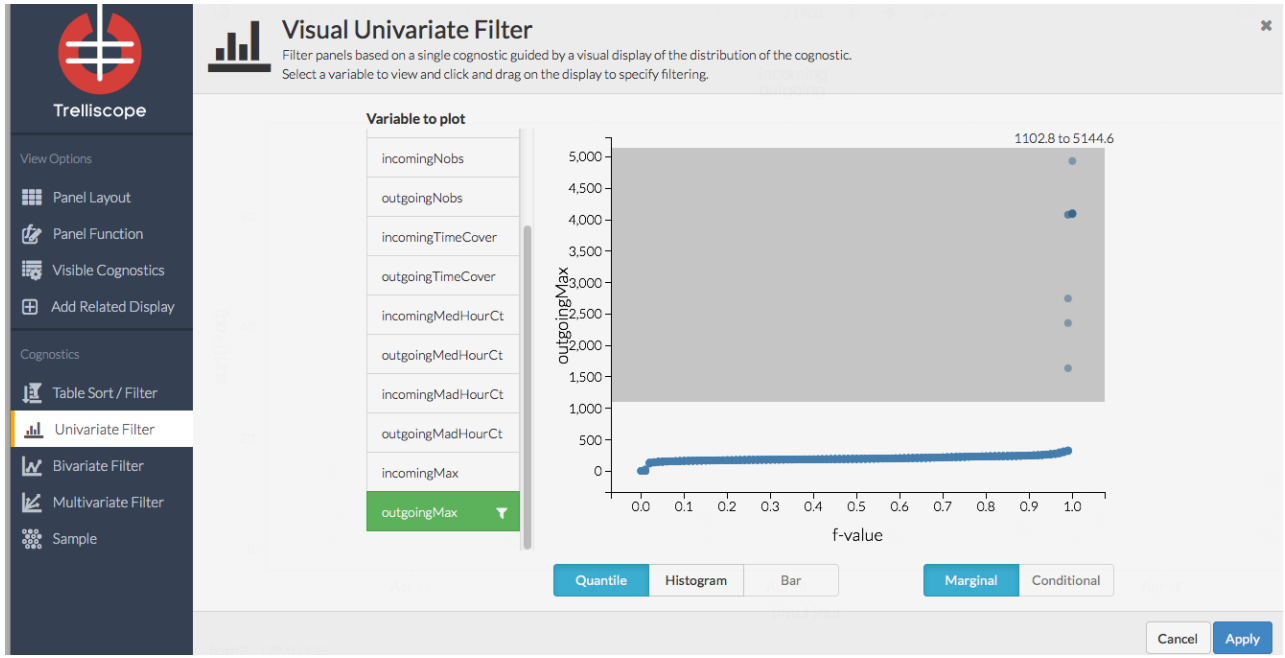
  Can you find the machines in question?

  - Hint: look for high numbers of outgoing connections

*"A botnet is a collection of Internet-connected programs communicating with other similar programs in order to perform tasks. This can be as mundane as keeping control of an Internet Relay Chat (IRC) channel, or it could be used to send spam email or participate in distributed denial-of-service attacks." (http://en.wikipedia.org/wiki/Botnet)

Challenge Solution

- In Trelliscope:
  - Use the Table Sort/Filter screen to select only **`Workstation`** from the **`type`** column
  - Then use the Univariate Filter screen, select the **`outgoingMax`** variable, then draw a box on the plot to select the outliers on the upper right



Session 5 Review

- At the end of this session you should understand
  - How to use Trelliscope to visually explore a large dataset
  - How to use cognostics to sort and filter Trelliscope displays to find relevant data

Session 6: Summary, Feedback and Wrap up

- Session 6: Summary, Feedback and Wrap up (4:00 p.m. – 4:30 p.m.)

Bootcamp Summary

Now that we've done the sessions you

- ✓ have an understanding for when and why you might use R for statistical analysis data
- ✓ feel comfortable with the basic functions of R
- ✓ understand how R can use functions from contributed packages, such as the Tessera R packages (DataDR and Trelliscope)
- ✓ understand the scalability that you get when you use a divide and recombine methodology as implemented in the DataDR package
- ✓ know how you might use visualization in data exploration
- ✓ understand how these tools might be applied to a dataset, such as a realistic (but simulated) large data set of network traffic data (Netflow)

Resources:

- Tessera main website: http://tesseradata.org
- DataDR: http://tesseradata.org/datadr/
- Trelliscope: http://tesseradata.org/trelliscope/
- Intro to R Bootcamp code and description: http://tesseradata.org/docs-r-intro-bootcamp
- VAST Challenge Data exploration: http://tesseradata.org/example-vast-challenge/

Feedback

Contact Information

- PNNL
  - Landon Sego, landon.sego@pnnl.gov
  - Ryan Hafen, ryan.hafen@pnnl.gov
  - Amanda White, amanda.white@pnnl.gov