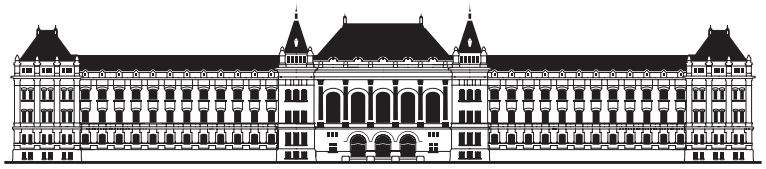

Budapest University of Technology and Economics
Department of Measurement and Information Systems
Fault Tolerant Systems Research Group

Critical Architectures Laboratory
Spring Semester 2015/2016

Monitoring Data Analysis

Syllabus
v1.0

Author: Ágnes Salánki
(salanki@mit.bme.hu)

March 9, 2016

Contents

1 Introduction **2**

2 Background **2**

2.1 System under analysis: our educational cloud 2

2.2 Data sources 3

2.3 Basic concepts of data analysis 3

2.4 Toolset: the R language 4

3 Analysis workflow **4**

3.1 Step 1: Transformation of small data 5

3.2 Step 2: Distributed data processing on MapReduce base 6

3.3 Step 3: Visualization 8

4 List of Questions **9**

5 Exercises **11**

1 Introduction

Infrastructure analytics is a special field of quantitative analysis where the outcome of the analysis process aims to support systems engineers with fine-tuning their domain knowledge. The input of the analysis is usually logs collected at several levels (OS, application, etc.) of different components. Nowadays, the components of an IT infrastructure are well-instrumented, resulting large amount of heterogeneous data.

The goal of this laboratory is (i) getting familiar with the basic techniques used for transformation and cleaning of small and large data sets (ii) practicing creation of good and informative visualizations (iii) getting an idea how interpretation of temporal analysis results support typical tasks of systems engineers, e.g. resource management and capacity planning.

The primary language of the laboratory is R. The basics of the language (value assignment, control structures, data types) can be acquired e.g. via the interactive DataCamp course[1] (which provides an R console in your browser so you do not have to install the R interpreter or any IDE on your machine) but feel free to use any tutorial you find useful. You will find a detailed list about R functions whose confident usage will be required to pass the prerequisite test and finish the laboratory on time.

2 Background

This section will briefly summarize the necessary theoretical background of the laboratory.

2.1 System under analysis: our educational cloud

The use case of this laboratory is going to be monitoring data originated from our educational cloud (VCL as *Virtual Computing Labs*). It contains nine physical hosts as worker nodes, each having 32 GB of RAM and 8 CPU cores [2].

Before the release of a VM type, it is configured by a minimally „promised" and maximally allowed memory amount and number of CPU cores. There is no CPU affinity set up, i.e. the mapping of logical cores to physical ones may change during the usage of the VM. Memory overcommitment is prohibited in the system, thus, the sum of minimally reserved memory of running VMs never exceeds the memory capacity of the physical machine. The VM type is available for the students three weeks before their deadline.

During the reservation period, the students send requests containing a VM type and the desired length of the reservation, which is usually maximized in a couple of hours. A request can be parametrized as: (i) an *immediate request* becoming active right after the reservation is acknowledged; in this case, after deployment onto the physical host, the VM starts loading; (ii) a *future request* being served at a specified time. The VCL scheduler examines the resource amount of the VM to be reserved and the current workload of the hosts. Each host has a static limit for every VM type, maximizing the number of VMs able to run on the host simultaneously. If there is no host which has enough spare capacity to run the requested VM, then it will be rejected.

[1] `https://www.datacamp.com/courses/free-introduction-to-r`

While using the system: if a reservation request is accepted, the VM is assigned to an appropriate host and loading starts. If students decide to finish their work earlier or the time expires, the VM is deleted and the resources are released.

2.2 Data sources

Two types of data was collected and stored in our educational cloud: (i) *reservation logs* containing information about the VM requests and (ii) *monitoring data* originated from the virtualization platform being sampled with a 20 second frequency.

Attributes characterizing a single log observation are the following:

request information: time and date of the request, requested VM type, desired length and timing (*immediate* or a start time in case of *future*) of the reservation;

system state: boolean flag indicating whether any of the hosts had enough capacity to serve the request;

reservation information: how much time the *load* took (time spent until the permission comes to students to log-in onto the remote machines) and how the reservation was closed (time expired, the user logged out and released the resources, etc.).

The attributes stored at the virtualization platform level are the basic resource utilization metrics of the host subsystems: the average CPU and memory usage, number of bytes read and written via the network interface or onto the local disk of the machine in the observed period (here the last 20 sec). Since VMs are not instrumented for monitoring/logging, we lack a more detailed analysis: e.g. there is no exact information about the resource usage ratio among the VMs and the management functions.

Our analysis aims at answering a very simple question of capacity planning: *according to our experiences in the observed period, (i) what are the recommended limits of virtual machines and (ii) what is the maximal number of reservations our system can serve reliably?*

2.3 Basic concepts of data analysis

The goal of data analysis projects is extracting data-based information about an observed system or phenomenon. In an ideal case, at the end of the project we have a deeper insight of how our system works (*what is happening?)* and have some hypotheses about the cause-effect relationships *(why could it happen this way?)*.

The analysis usually consists of two steps: *exploratory* and *confirmatory* phases (exploratory and confirmatory data analysis, *EDA* and *CDA*, respectively).

The exploratory data analysis phase consists of activities related to the exploration of the system. It answers the most basic questions related to the marginal and joint distributions of variables (the marginal distribution of each individual variable and basic relationships between them). Since one- and two-dimensional visualization techniques are excellent (intuitive, fast) tools for extracting this information, the exploratory analysis in practice means the plotting of, and inspection into, many graphical plots.

At the end of the EDA phase, we already have some ideas, so-called hypotheses about the basic phenomena in the system. For example "the distribution family of the processing

time is a two-modal Gaussian". However, to prove (or publish) these, ad-hoc ideas are not enough, we need statistically significant results. This is the main task of the CDA phase. Primary tools here are the statistical tests (z-test, chi-square test, etc.).

This laboratory focuses on the exploratory phase, thus, concepts of statistical testing will not be tested.

2.4 Toolset: the R language

R is a popular analytical framework, originally tailored for statistics and visualization. It has more than 7000 packages, built-in functions support the analyst at every step from data cleaning through manipulation to reporting and publishing of results.

We will use only a very narrow subset of R functions, related to data manipulation, distributed data processing, visualization and reporting. This syllabus contains the required theoretical background of the laboratory: the basic visualization types, the concepts of wide and long data frames and the MapReduce programming paradigm.

However based on our experiences, solid knowledge of particular R packages (functions) can boost your performance during the analysis (and allow to finish your exercises in time), thus, familiarity with the following functions is required for the laboratory:

reshape2:: dcast, melt – transformation between wide and long data tables[2],

ggplot2:: qplot – sophisticated (static) data visualization[3],

plyr:: ddply – category-dependent computations[4],

stats:: lm – linear model fitting[5],

rmr2:: mapreduce, to|from.dfs, keyval, c.keyval – distributed computations[6],

base:: as.POSIXct, strptime – conversion between date and string[7],

utils:: read|write.csv – loading and saving data in a tabular format[8].

3 Analysis workflow

The first part of the laboratory (you should spend approx. one and a half hour on it) is about data transformation and exploration, the second section (approx. one hour) focuses on distributed computations, the third part is less coding and ideal to improve your interpretation skills (approx. one hour). This section briefly summarizes the main steps of your workflow.

[2] http://seananderson.ca/2013/10/19/reshape.html
[3] http://blog.echen.me/2012/01/17/quick-introduction-to-ggplot2/
[4] http://seananderson.ca/2013/12/01/plyr.html
[5] http://www.statmethods.net/stats/regression.html
[6] http://home.mit.bme.hu/~ikocsis/notes/2013/10/23/(r)hadoop-sandbox-howto/
[7] http://www.cyclismo.org/tutorial/R/time.html
[8] http://www.r-tutor.com/r-introduction/data-frame/data-import

3.1 Step 1: Transformation of small data

During transformation, our data set can be available in several different forms, depending how many empty cells (NA stands for ”not available” values in analytics terminology) the data table has or how redundant it is. There are two distinguished formats we get familiar with in detail: the *long* and the *wide* data format.

Long data frames have as few columns as possible, minimally three: the ID of the object, a feature name (what we have measured, observed, etc.) and the value itself. For example, ESX logs are typically in a long format:

```
>head(esx_log)
Timestamp               variable        host_ID Instance Value
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "1"    "1.57"
"2014.03.21.7:05:00"    "mem.usage.average" "394" ""     "31.33"
"2014.03.21.7:05:00"    "net.transmitted"   "394" ""     "1.99"
"2014.03.21.7:05:00"    "net.received"      "394" ""     "1.67"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "5"    "2.68"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "7"    "1.48"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "2"    "1.37"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "3"    "0.46"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "4"    "1.89"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "0"    "1.97"
```

Listing 1: ESX logs are typically stored in a long format

Wide data frames have one ID column and every feature appears in a separate column: one observation is characterized with a single row in our data table.

For example, the reservation log of the VCL is stored in a wide format, where a row represents a reservation query:

```
>head(reservations, n = 10)
reservation_id time_of_reservation start_of_reservation
   end_of_reservation length_of_reservation VM_type host_ID
5337 2014.01.24 15:20 2014.01.24 15:27 0.1786 Windows7 394
5338 2014.01.24 15:32 2014.01.24 15:33 0.0867 Windows7 396
5339 2014.01.24 15:33 2014.01.24 16:25 0.9272 Windows7 397
5340 2014.01.27 7:48 2014.01.27 16:00 8.2022 Windows7 397
5341 2014.01.27 11:31 2014.01.27 11:46 0.3050 ITLab1 MIT2 Client 396
5342 2014.01.28 7:07 2014.01.28 7:09 0.0475 ITLab1 MIT2 Client 396
5343 2014.01.28 7:30 2014.01.28 7:32 0.1044 ITLab1 MIT2 Client 397
5344 2014.02.01 11:32 2014.02.01 11:51 0.3300 ITLab1 MIT2 Client 396
5345 2014.02.01 11:41 2014.02.01 11:51 0.2011 NFS Server 397
```

Listing 2: The reservation log stored in a wide format

In general, the long format is much more flexible; in case of entering a new feature in our system (e.g. our instrumentation has changed and we monitor more metrics of the application) we only add a new line in the right format without any configuration problems. On the other hand, the wide data format allows us to handle features absolutely separately.

reshape2

melt and *dcast* are the most useful transformation functions for changing between long and wide format. Please take a look at their help page[9].

3.2 Step 2: Distributed data processing on MapReduce base

MapReduce is a general programming paradigm allowing parallel processing of large data sets. The "divide and conquer" concept of MapReduce – that we write simple functions executed on small data chunks and at the end we somehow summarize the results yielded from the individual functions – should be familiar from other domains, where distributed, massively parallelizeable algorithms run.

Two main steps of a MapReduce job are – not surprisingly – Map and Reduce. The typical task of the Map phase is usually some filtering, while the summarization is performed during the Reduce. Note: the input of the Map is only a randomly assigned data chunk, the Reduce phase handles usually a more homogeneous data subset. MapReduce implementations nowadays (e.g. Hadoop) take care of the whole management workflow (split the data into chunks, assign the computation jobs to them, collecting and merging the results of the Map phase etc.) and thus, the developers only have to write the two worker functions.

For example, we have a larger ESX log file containing resource utilization metrics in a long format. We are curious how our resources are utilized, thus, we would like to extract *the maximum utilization of each resources*.

Fig. 1 shows what happens in each phase of a possible implementation of this exercise, where the resources are considered as keys. (The original figure credit goes to [1].)

Mappers process the data chunk by row, emits the results of a filtering in form of <*key, value*> pairs, where the resource is the *key*. Note that the we abstract the timestamp variable entirely, decreasing the amount of data to be sent further to Combiner jobs.

Combiners summarize the values belonging to one key, practically performing a Reduce on the small data set. It reduces further the data set further (note again the communication overhead in a distributed system). In simpler cases the Map and Combiner functions are merged.

Partitioners aggregate values by keys, transmit the results of the Mapper/Combiner jobs in a <*key, value list*> format.

Reducers perform the final summarization (or other computations) for an individual key.

rmr2

The rmr2 package is a MapReduce implementation, where Map and Reduce functions can be written in R. The package is able to connect to a Hadoop distribution, and – and this is the most useful feature for us – can run on a single compute node. Thus, we are able to get familiar with its functions without installing a whole Hadoop cluster into our analysis VM.

Please take a look at a possible solution for the exercise above.

[9] A detailed description can be found here: `http://seananderson.ca/2013/10/19/reshape.html`

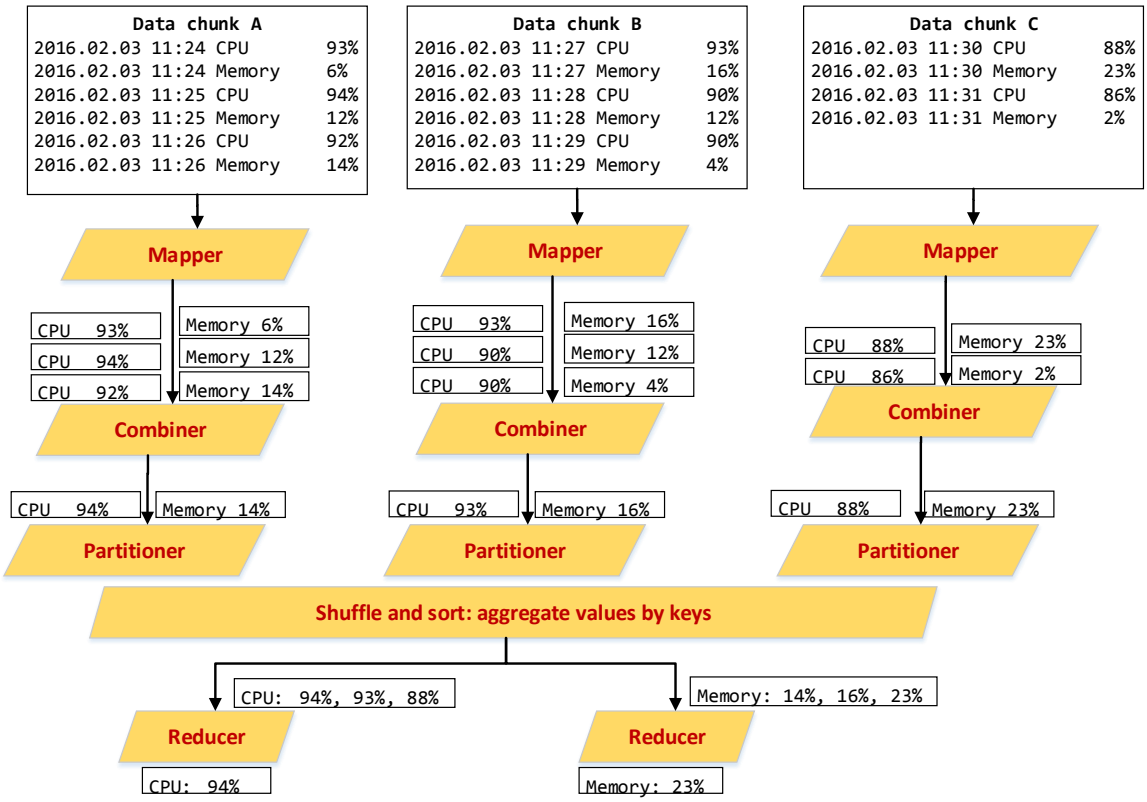

Figure 1: A possible solution for the maximum resource utilization exercise

```
library(rmr2)
library(stringr)
rmr.options(backend = "local")

  a <- mapreduce(input="VCL_sample.csv",
                 input.format="csv",

                 map=function(k,v){
                   v <- v[v[, 4] == "", ]
                   keyval(key=v[, c(2, 3)],
                          val=v[, 5])},

                 reduce=function(id, numbers.list){
                   keyval(key=id,
                          val=max(numbers.list))}
  )()

  b <- from.dfs(a)
  b
```

Listing 3: rmr2 code for the maximal resource utilization problem

3.3 Step 3: Visualization

Faithful and quick visualization is a key during EDA.

Note: *visual analytics* and *information visualization* are different concepts.

Information visualization – nice examples are e.g. in newspapers – is the art of storytelling with charts. Business analysts, designers, UX experts work on the best possible presentation of some *information already found*. In other words, infographics[10] are the result of an analysis, they contain the most compelling or useful or impressive relationships found in the data. The recipients of this information are people who have never seen the raw data.

On the other hand, the analyst – who ideally has a solid knowledge of the context and spent a couple of hours with data munging – would like to focus on discovering patterns on a plot. She knows exactly the input data set was, what axes X and Y contain, etc. Thus, in these cases, performance beats aesthetics in priority. The goal of our plots in this laboratory is to gain a better understanding of the system under analysis. Therefore, do not expect plots, which – at least with their default settings – will be publication-ready or even nice-looking.

EDA typically uses low-dimensional visualizations, the most common plot types are the following.[11]

- Histograms and boxplots for one-dimensional numerical variables;
- Barcharts for one-dimensional categorical variables;
- Scatterplots for visualization of two numerical variables;
- Mosaic plots for visualization of two categorical variables.

Sometimes high dimensional plot types are useful as well like scatterplot matrices and parallel coordinates, however, simpler dimension increasing methods such coloring and grouping are worth considering before you use more complicated visualizations.

The grammar of graphics and the ggplot2 package

Wilkinson's „The grammar of graphics" (1999) is a definitive book on the area of data visualization. Its concept of general geometric objects (bars, lines, points, etc.) and their aesthetic features (size, color, etc.) opened a new way of thinking among visualization people. It became natural in the 2000s and is present in most visualization tools nowadays (Tableau, IBM Watson Analytics Visualization Dashboard, Microsoft Power BI).

ggplot2 is an open-source implementation of the grammar of graphics paradigm [3]. During the laboratory, its usage is recommended but not required, you can use any visualization framework.[12]

[10] Nice examples here: `http://goo.gl/8O6buk` and here: `http://goo.gl/Crc7uW`

[11] A great source to learn the basic visualizations is the Visual Analysis chapter here: `http://goo.gl/5blwrK`

[12] The concept can be learned from several sources, you may want to check out `http://blog.echen.me/2012/01/17/quick-introduction-to-ggplot2/` or `http://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html`

4 List of Questions

1. The reservation logs have typically a wide format. How would it look like in a long format? Which R function would you use to do transformation?

```
>head(reservations, n = 10)
reservation_id time_of_reservation start_of_reservation
   end_of_reservation length_of_reservation VM_type host_ID
5337 2014.01.24 15:20 2014.01.24 15:27 0.1786 Windows7 394
5338 2014.01.24 15:32 2014.01.24 15:33 0.0867 Windows7 396
5339 2014.01.24 15:33 2014.01.24 16:25 0.9272 Windows7 397
5340 2014.01.27 7:48 2014.01.27 16:00 8.2022 Windows7 397
5341 2014.01.27 11:31 2014.01.27 11:46 0.3050 ITLab1 MIT2 Client 396
5342 2014.01.28 7:07 2014.01.28 7:09 0.0475 ITLab1 MIT2 Client 396
5343 2014.01.28 7:30 2014.01.28 7:32 0.1044 ITLab1 MIT2 Client 397
5344 2014.02.01 11:32 2014.02.01 11:51 0.3300 ITLab1 MIT2 Client 396
5345 2014.02.01 11:41 2014.02.01 11:51 0.2011 NFS Server 397
```

Listing 4: The reservation log stored in a wide format

2. The ESX logs have typically a long format. How would it look like in a wide format? Which R function would you use to do the transformation?

```
>head(esx_log)
Timestamp               variable        host_ID Instance Value
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "1"   "1.57"
"2014.03.21.7:05:00"    "mem.usage.average" "394" ""    "31.33"
"2014.03.21.7:05:00"    "net.transmitted"   "394" ""    "1.99"
"2014.03.21.7:05:00"    "net.received"      "394" ""    "1.67"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "5"   "2.68"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "7"   "1.48"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "2"   "1.37"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "3"   "0.46"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "4"   "1.89"
"2014.03.21.7:05:00"    "cpu.usage.average" "394" "0"   "1.97"
```

Listing 5: The reservation log is stored in a wide format

3. Assuming the above mentioned format of the reservation logs, how would you visualize
   - how many VMs the individual hosts have run overall the two years?
   - the marginal distribution of reservation lengths?
   - the marginal distribution of reservation lengths along the individual hosts?
   - how the reservation affinity of a specific VM type has changed in time?
   - how many instances the individual hosts have run from each VM type?
4. Assuming the above mentioned format of the reservation logs, how would you use the *ddply* function to compute the median reservation length for each VM type?
5. Let's assume that we have an extremely long ESX log in the long table used above and we would like to compute, which happens more frequently: that the memory usage of

a specific host is higher than its CPU usage or the opposite? Write MapReduce-based pseudocode to solve the problem, indicating the exact input and output type of each step.

6. How would you figure it out whether there is a strong the linear relationship is between the CPU and memory usage of a host.

7. You would like to transform the reservation log into a data frame containing four columns: *Timestamp, host_ID, VM_type and Number* where a row represents how many VMs a particular host have run in a particular minute. How would you do that? List the steps or write pseudocode to transform the *df.1* into the *df.2*.

```
>df.1
reservation_id start_of_reservation end_of_reservation VM_type
   host_ID
5338 2014.01.24 15:32 2014.01.24 15:33 Windows7 396
5339 2014.01.24 15:32 2014.01.24 15:34 Windows7 397
5339 2014.01.24 15:33 2014.01.24 15:35 Windows7 397
5341 2014.01.24 15:33 2014.01.24 15:35 ITLab1 MIT2 Client 396

>df.2
Timestamp host_ID VM_type Number
2014.01.24 15:32 396 Windows7 1
2014.01.24 15:32 397 Windows7 1
2014.01.24 15:33 396 Windows7 1
2014.01.24 15:33 396 ITLab1 MIT2 Client 1
2014.01.24 15:33 397 Windows7 2
2014.01.24 15:34 396 ITLab1 MIT2 Client 1
2014.01.24 15:34 397 Windows7 2
2014.01.24 15:35 396 ITLab1 MIT2 Client 1
2014.01.24 15:35 397 Windows7 1
```

Listing 6: Example input and output

Environment

The VM you will work with has the following setup:

- R 3.2.2 `apt-get install r-base r-base-dev`
- RStudio 0.99 `http://www.rstudio.com/products/rstudio/download/`
- required packages: *ggplot2*, *reshape2*, *plyr*, *knitr*, *rmarkdown*, *Rcpp*, *RJSONIO*, *bitops*, *digest*, *functional*, *stringr*, *plyr*, *reshape2*, *caTools*
- rmr-3.3.1. packages `http://github.com/RevolutionAnalytics/RHadoop/wiki/Downloads`, easy install via GUI
- test scripts: `http://home.mit.bme.hu/~ikocsis/notes/2013/10/23(r)hadoop-sandbox-howto`

5 Exercises

To avoid the ad-hoc, unnecessary steps during the laboratory (which are quite typical in every data analysis project), the exercises are this time a little more specific than usual. You find the skeleton of the laboratory in .Rmd files, please, follow that guide and eventually submit your report in that format (too) on GitHub.

Acknowledgments

The data you will use during the laboratory originates from our VCL system, being maintained by our VCL gurus, Áron Tóth and Dávid Cseh.

References

[1] Jimmy Lin and Chris Dyer. Data-intensive text processing with MapReduce. *Synthesis Lectures on Human Language Technologies*, 3(1):1–177, 2010.

[2] Ágnes Salánki, Gergő Kincses, László Gönczy, and Imre Kocsis. Data analysis based capacity planning of VCL clouds. *3rd International IBM Cloud Academy Conference (ICACON 2015)*, 2015.

[3] Hadley Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.