12-2016

Tutorial for Using the Center for High Performance Computing at The University of Utah and an example using Random Forest

Stephen Barton
Utah State University

TUTORIAL FOR USING THE CENTER FOR HIGH PERFORMANCE COMPUTING AT THE UNIVERSITY OF UTAH AND AN EXAMPLE USING RANDOM FOREST

by

Stephen Willis Barton

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

In

Statistics

Approved:

Dr. Adele Cutler	Dr. Richard Cutler
Major Professor	Committee Member

Dr. John Stevens
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2016

ABSTRACT

Tutorial for Using the Center for High Performance Computing at The

University of Utah and an example using Random Forest

by

Stephen Willis Barton, Master of Science

Utah State University, 2016

Major Professor: Dr. Adele Cutler
Department: Mathematics and Statistics

Random Forests are very memory intensive machine learning algorithms and most computers would fail at building models from datasets with millions of observations. Using the Center for High Performance Computing (CHPC) at the University of Utah and an airline on-time arrival dataset with 7 million observations from the U.S. Department of Transportation Bureau of Transportation Statistics we built 316 models by adjusting the depth of the trees and randomness of each forest and compared the accuracy and time each took. Using this dataset we discovered that substantial restrictions to the size of trees, observations allowed for each tree, and variables allowed for each split have little effect on accuracy but improve computation time by an order of magnitude.

Becoming familiar with the CHPC is significantly easier with the included tutorial at the end of the paper.

(32 pages)

ACKNOWLEDGMENTS

I would like to thank my wife for the tireless support, helpful hand, and willing ear that she gives me. The IT administrators at the Mathematics and Statistics Department have also been invaluable in setting up an RStudio Server for me to use. I would especially like to thank my committee members Drs. Richard Cutler and John Stevens, and most especially Adele Cutler, my major professor.

The support and resources from the Center for High Performance Computing at the University of Utah are gratefully acknowledged. Wim Cardoen was an indispensable resource in setting up the supporting code to run jobs on the cluster.

CONTENTS

ABSTRACT ii
ACKNOWLEDGMENTS iii
LIST OF FIGURES v
1. INTRODUCTION 1
2. METHODOLOGY 2
3. AIRLINE DATA 3
4. EXPERIMENTS 4
4.1. Predicting Late Flights Using Subset of 2008 Data With 5% Sample Size 4
4.2. Predicting Late Flights Using All of the 2008 Data and 5% Sample Size 6
4.3. Repeated Model Building 7
4.4. Random Forest Performance 8
5. CHPC TUTORIAL 10
5.1. Virtual Private Network 10
5.2. Transferring Files 11
5.3. Interactive GUI 13
5.4. Submitting Batch Scripts 17
5.4.1. Dummy Example of Slurm 18
5.4.2. Actual Random Forest example using Slurm 21
6. BIBLIOGRAPHY 22
APPENDIX 23
A.1. R Code 23
A.2. Configuration File 26
A.3. SLURM Batch File 27

LIST OF FIGURES

Figure 1 Maxnodes vs Error Rate with 5% sampling of subset of 1 million observations; log scale on horizontal axis 5

Figure 2 Maxnodes vs Error Rate with 5% sampling on full dataset; log scale on horizontal axis 7

Figure 3 Maxnodes vs Error Rate with mean and 95% confidence interval 8

Figure 4 The Cisco AnyConnect Secure Mobility Client 10

Figure 5 Confirmation of secure connection 10

Figure 6 CISCO's WebLaunch platform to download the client and automatically connect 11

Figure 7 Starting a new session with WinSCP 12

Figure 8 Click on the PuTTY button to open a terminal 13

Figure 9 FastX2 connection list before and after adding a connection 13

Figure 10 Create a new FastX2 SSH connection 14

Figure 11 Connection awaiting a session 14

Figure 12 Configure new session 15

Figure 13 Connection with active session 16

Figure 14 Virtual Desktop with the path to open a terminal window highlighted 16

Figure 15 Commands to open RStudio 17

Figure 16 Basic flowchart showing structure of clusters 17

1. INTRODUCTION

Random Forests (Breiman 2001) are standard tools for classification, in part because they work quite well “right out of the box”. The idea is to fit a forest of binary trees similar to those described in Breiman et al. (1984). However, at each node, instead of searching for the best possible split among all predictors, the search is done over a small random sample of predictors. Each tree is grown on a bootstrap sample of the data and predictions are obtained by passing a new observation down each tree and allowing the trees to vote for the predicted class of the new observation.

For classification, there are three tuning parameters the analyst must choose:

1. The number of randomly sampled predictors for each split (mtry).
2. The number of trees in the forest (ntree).
3. The size of the trees (maxnodes).

The number of trees is usually chosen to be as large as feasible because it is well known that increasing the number of trees will not cause the predictions to deteriorate but rather become more robust. The size of the trees is usually chosen to be as large as possible by growing each tree until the terminal nodes can not be split (either they are all from the same class or they all have identical predictor variables at the node). Using these two default choices, the only tuning parameter of any real concern is the number of randomly selected predictors at each node.

Random Forest is an example of “embarrassingly parallel” algorithms. In the traditional parallel computing paradigm, the data could be sent to a number of powerful machines which would each grow trees on the data. Finally, the trees would be combined to form a forest.

However, Random Forests are not easily adapted to today's Big Data world. To understand why it's important to understand how current computing environments use commodity machines to operate on Big Data. Commodity machines are low-cost computers that are housed in huge data centers. They regularly fail and are replaced. Current Big Data methodology breaks a large data set into small subsets and sends each subset to 3 or 4 commodity machines. In this way, the data are distributed over thousands of commodity machines, but each commodity machine has access to only a small subset of the data. The subsets are sent to 3 or 4 commodity machines to help reduce the chance that data will be lost due to failure. Each commodity machine works only on the piece of data it is given, and results are combined centrally. Much of the current research in statistical computing for Big Data has focused on how to compute standard statistical estimates such as regression coefficients when we combine results from the distributed framework.

Random Forests face two main challenges in using commodity machines. The first is that each tree is grown on a bootstrap sample of the data, which is as big as the original data. The second is that the trees are grown very deep, which requires a lot of memory.

This project investigates modifications of the Random Forest methodology to make Big Data Random Forests feasible for commodity machines. In order to perform the experiments required for this work, it was necessary to use the University of Utah Center for High Performance Computing (CHPC) facility. A major part of this project is a tutorial to help other statisticians at USU use the facility.

2. METHODOLOGY

One of the main difficulties for Random Forests for Big Data is that the data are dispersed among many commodity servers. Instead of trying to fight this, we consider fitting each tree in the forest to a small subset of the data instead of bootstrapping. Ideally, the subsets would be random. Current Big Data implementations (e.g. Hadoop) do not guarantee randomness, but investigating this aspect is beyond the scope of this project. The second thing we explore is the effect of changing the tree size in the forest, and finally, we explore changing the value of the number of randomly selected predictors at each node.

The R package randomForest (Liaw et al. 2002) was used to perform the experiments. The computations were not performed on commodity machines, but we simulated using them by sampling 5% of the data at random, without replacement, for each tree on high-performance server CPUs. The number of trees was held constant at 500.

3. AIRLINE DATA

The focus of the 2009 Joint Statistical Meetings' Data Exposition poster session, jointly sponsored by the Graphics Section and Computing Section was on Airline on-time performance (http://stat-computing.org/dataexpo/2009/). The departure and arrival details of all commercial flights from 1987 to 2008 in the US were made available in yearly segments because of the size of the dataset: nearly 120 million flights. The goal was to illustrate a graphical summary of a research question left up to the participants. We use these data to compare different Random Forest parameters for predicting whether a flight will be at least 15 minutes late.

The variables in the dataset (Table 1) describe flight details regarding departure time and airport, delays in departure and arrival and reasons for delays, time spent taxiing, and arrival time and airport. The 2008 data have 7,009,728 observations, with a file size of 657.5 Megabytes. There were 154,699 flights that were either canceled or diverted and have been removed from the dataset for this analysis. Airports, flight numbers, and tail numbers were also removed because they were categorical variables with hundreds of levels. The models were built on the following variables, ActualElapsedTime, AirTime, ArrTime, CRSArrTime, CRSDepTime, CRSElapsedTime, DayofMonth, DayOfWeek, DepDelay, DepTime, Distance, Month, TaxiIn, and TaxiOut.

Variable Name	Description	Variable Name	Description
Year	1987-2008	DepDelay	departure delay, in minutes
Month	1-12	Origin	origin IATA airport code
DayofMonth	1-31	Dest	destination IATA airport code
DayOfWeek	1 (Monday) - 7 (Sunday)	Distance	in miles
DepTime	actual departure time (local, hhmm)	TaxiIn	taxi in time, in minutes
CRSDepTime	scheduled departure time (local, hhmm)	TaxiOut	taxi out time in minutes
ArrTime	actual arrival time (local, hhmm)	Cancelled	was the flight cancelled?
CRSArrTime	scheduled arrival time (local, hhmm)	CancellationCode	reason for cancellation (A = carrier, B = weather, C = NAS, D = security)
UniqueCarrier	unique carrier code	Diverted	1 = yes, 0 = no
FlightNum	flight number	CarrierDelay	in minutes
TailNum	plane tail number	WeatherDelay	in minutes
ActualElapsedTime	in minutes	NASDelay	in minutes

Table 1 Variables in airline dataset and descriptions

4. EXPERIMENTS

4.1. Predicting Late Flights Using Subset of 2008 Data With 5% Sample Size

Our initial investigation was to determine the accuracy of smaller and more constrained trees on predicting flights with delays of 15 or more minutes. For classification problems such as this one, the default mtry, or the number of variables randomly sampled as candidates at each split, is the square root of the number of predictors, rounded down; in this experiment, with 14 predictor variables the algorithm randomly samples three of them. These models were restricted to a randomly selected subset of size one million to investigate the effect that mtry and maxnodes have on the error rate with such constraints. Figure 1 summarizes the results of 120 models. The most interesting observation was the immediate drop in error rate when mtry was increased from 6 to 7 variables with maxnodes=3. The disparity disappears by increasing the size of the trees by one node. More generally, as maxnodes increases, the models generated by mtry=3 take much longer to minimize error rate than those with higher mtry values.

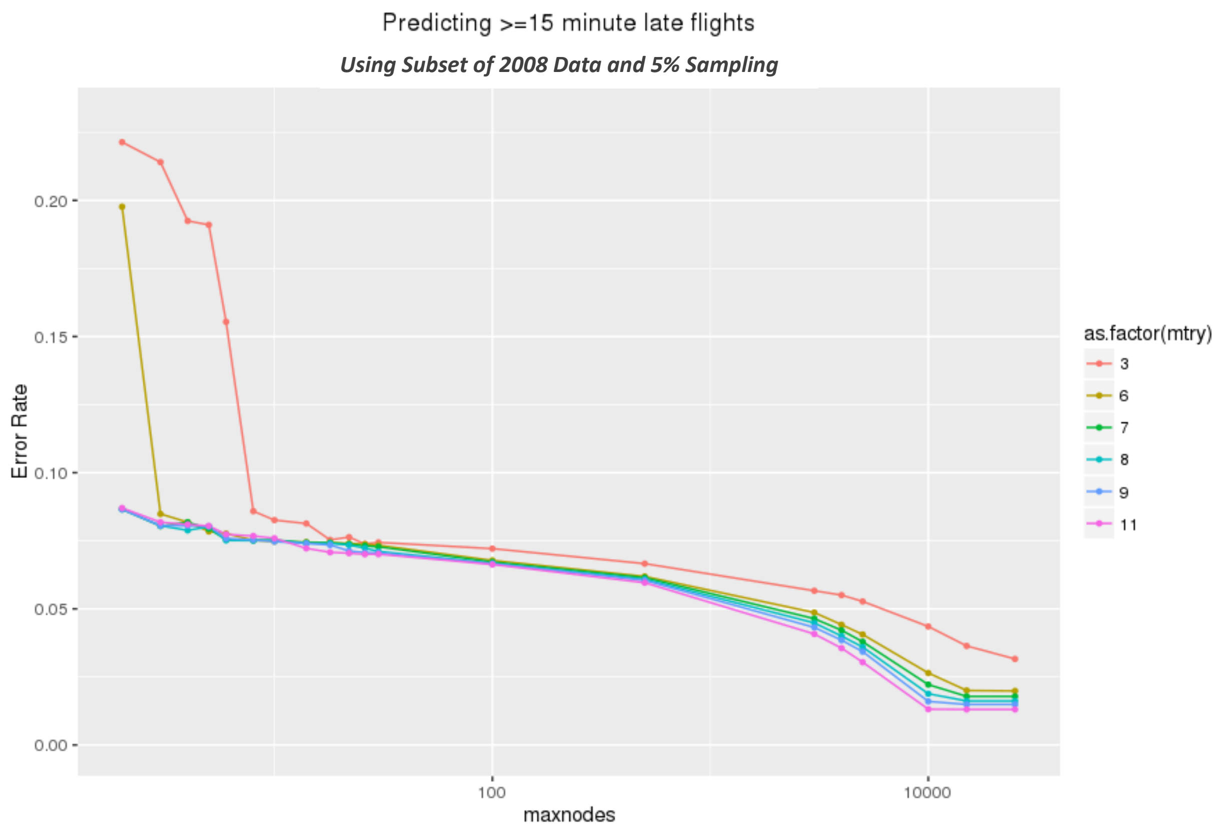


Figure 1 Maxnodes vs Error Rate with 5% sampling of subset of 1 million observations; log scale on horizontal axis

4.2. Predicting Late Flights Using All of the 2008 Data and 5% Sample Size

Extending the previous experiment to the whole year's dataset was much more complicated than simply changing the input data frame and response vector because of the size of the data and the amount of time required in computation. This experiment and the following one were performed through the Center for High Performance Computing at The University of Utah. The results of 312 models are summarized in figure 2.

The phenomenon between mtry=6 and mtry=7 from the earlier experiment seems to have shifted to between mtry=7 and mtry=8. More models were built and it is even more apparent using the whole year as shown in figure 2 the high volatility in the forests with smaller trees. After increasing the maximum number of nodes to at least ten the models become more stable. This is consistent with the previous experiment (fig. 1).

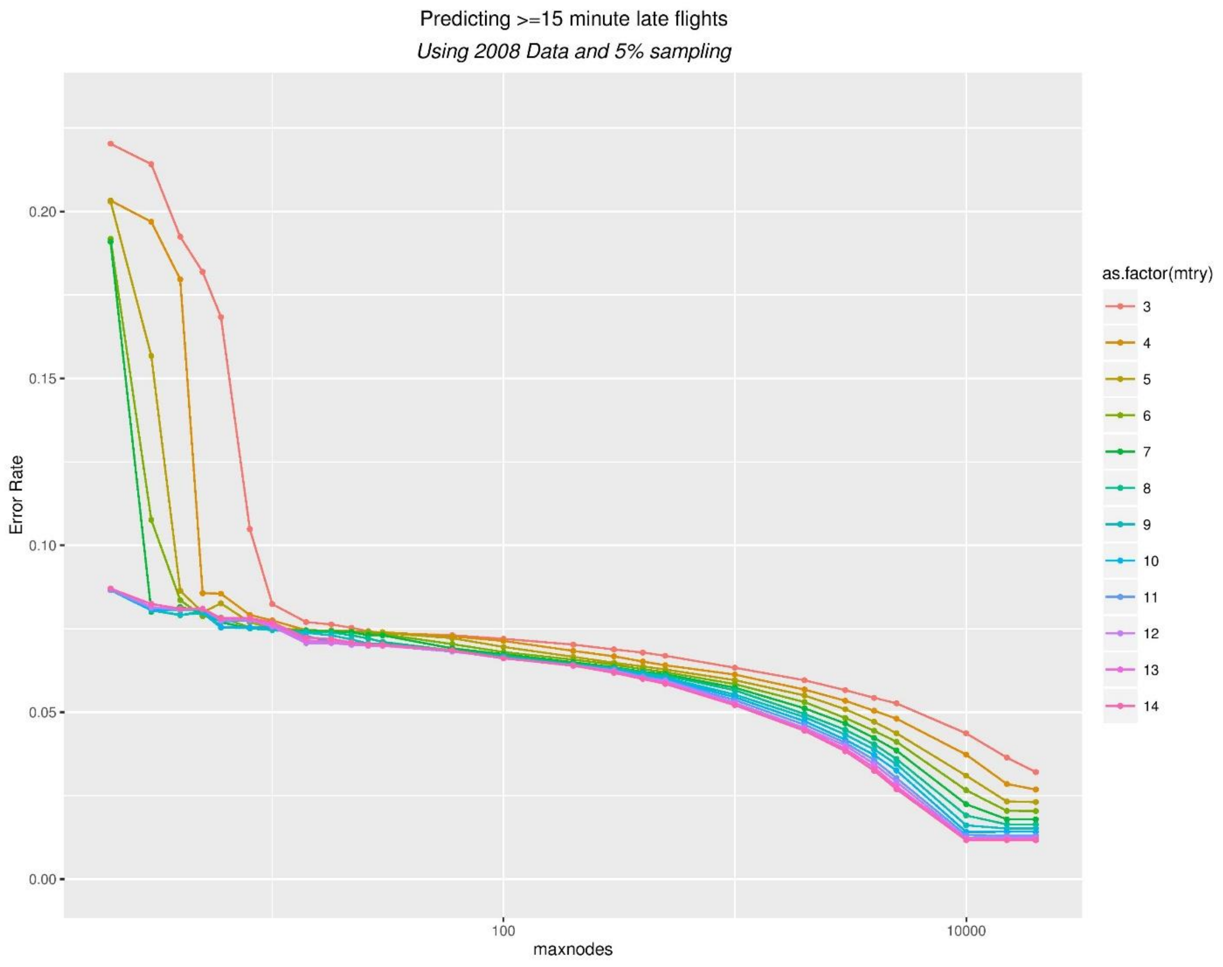


Figure 2 Maxnodes vs Error Rate with 5% sampling on full dataset; log scale on horizontal axis

4.3. Repeated Model Building

The strange pattern generated from the models with small trees and a lower number of mtry variables in Fig. 2 interested me. I therefore performed several repetitions of models with maxnodes less than 30 to determine if the oddity was persistent or based on the stochastic nature of the random forest algorithm. Figure 3 summarizes the results after five repetitions of each model showing the 95% confidence interval of the mean error rate. Notice how the high variability in the smallest (maxnodes=2) mtry=7 model could possibly explain the phenomenon from the previous experiments' results because of the wide range of error rates produced by the random forest models.

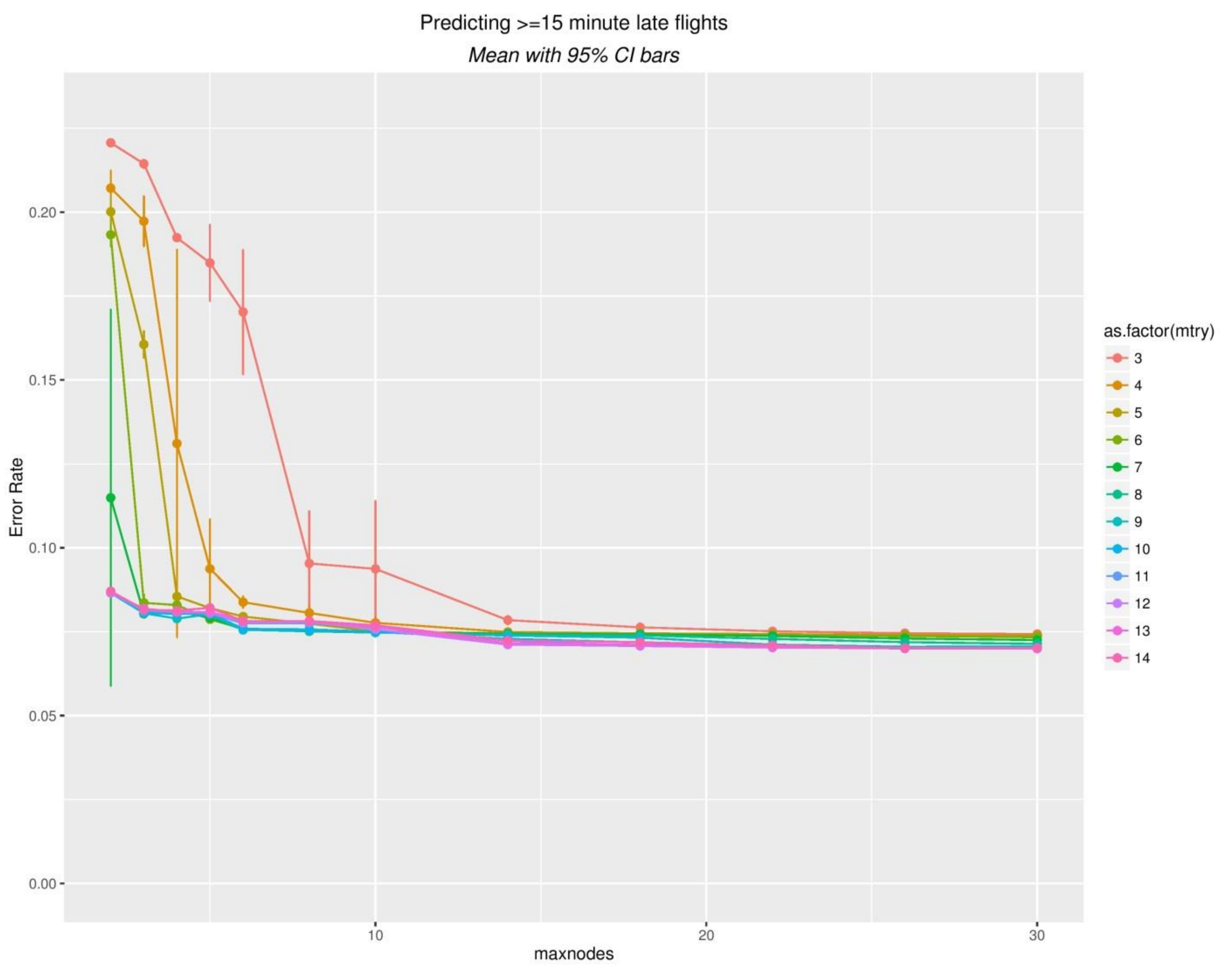


Figure 3 Maxnodes vs Error Rate with mean and 95% confidence interval

4.4.Random Forest Performance

Random forests are an example of an embarrassingly parallel algorithm because very little is required to distribute the workload across cores and nodes. One method of taking advantage of random forests' easily parallelizable algorithm is by dispersing the tree growing phase. Tree growth is done on bootstrap samples of the data and trees are constructed independently. The randomForest package includes a function to combine several trees or forests into one larger forest object thereby allowing multiple cores to build a forest.

I did not implement this method of forest building, however. I built whole forests on single cores and I took advantage of the multitude of cores and nodes at the CHPC to reduce the physical time required. The 312 models built took over 584 hours of total CPU time to complete. However, these were not built in serial; each node I used contained 32 cores so I ran 10 jobs simultaneously. The longest job took a total elapsed time of 3.25 hours.

The following four models were built one at a time using the CHPC. I grew them to their full depth. A model built using the one million observation subset and using 5% sampling from the one million observation subset took 7.75 minutes with an error rate of 4.85%. A similar model built using bootstrap sampling took 75 minutes with an error rate of 2.78%. Building a model using the full seven million observation dataset and 5% sampling took 70.33 minutes and produced an error rate of 3.25%. A random forest model with all default arguments would have taken over 10 hours if it had finished.

5. CHPC TUTORIAL

Your Principal Investigator (PI, usually your major professor) will need to request access for you, after which you will be given a uNID and can sign up for an account with CHPC. For further information on topics not covered in this tutorial go to chpc.utah.edu. Three pieces of software are required for this tutorial—a virtual private network client, an interactive GUI, and a program for transferring files.

5.1. Virtual Private Network

Use “Cisco AnyConnect Secure Mobility Client” application to connect to vpnaccess.utah.edu using your uNID and password in order to start a virtual private network (VPN), as in Fig. 4.

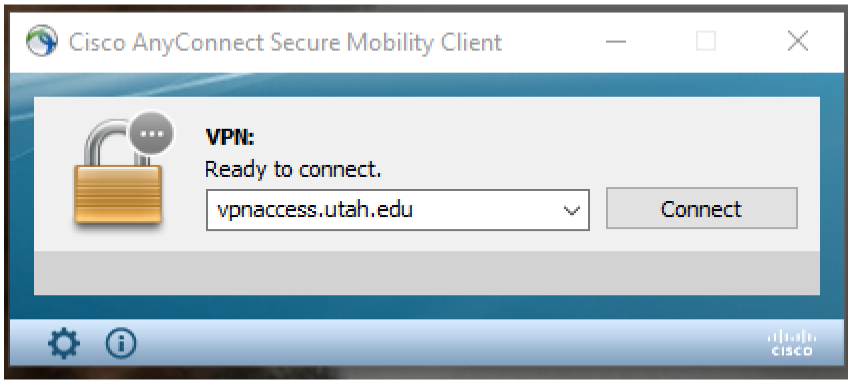


Figure 4 The Cisco AnyConnect Secure Mobility Client

On a Windows machine, the image in Fig. 5 will appear on your screen just above the taskbar when successfully connected.

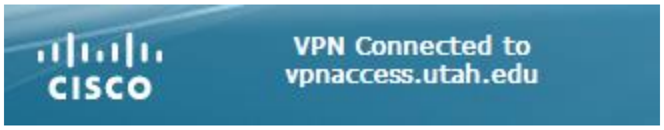
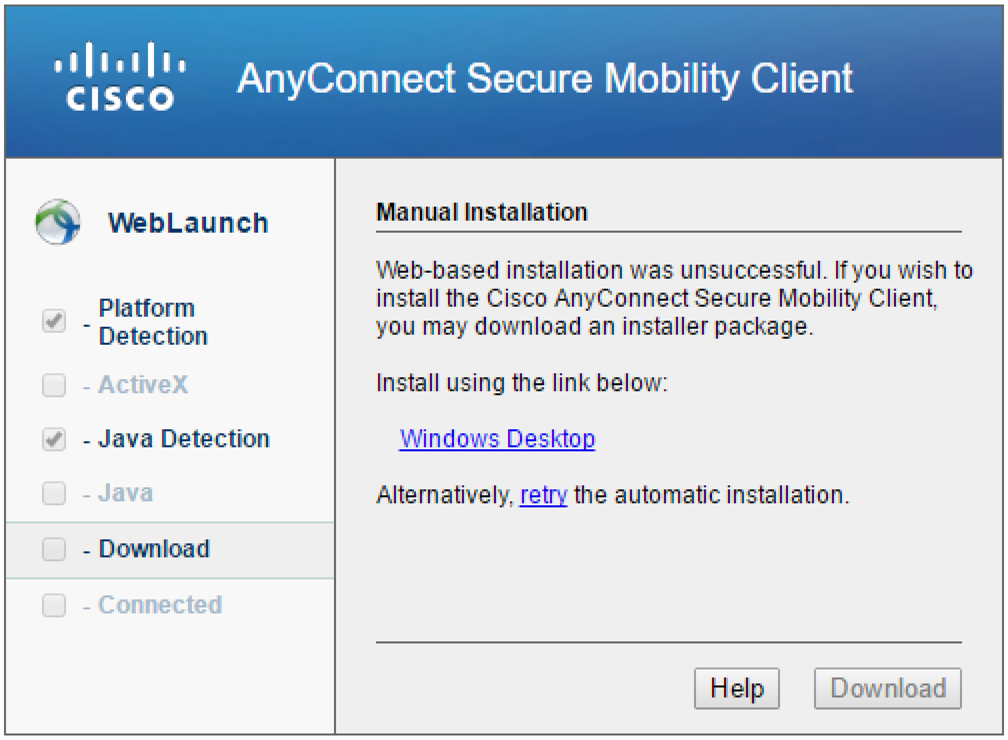


Figure 5 Confirmation of secure connection

If you don’t have the Client, using a browser log on to vpnaccess.utah.edu, click on “Start AnyConnect”. The web app will detect your browser and operating system, attempt to automatically install the client, and start a connection (see Fig.6). Depending on your browser you may need to manually download the VPN client.



AnyConnect Secure Mobility Client

WebLaunch

- ✓ - Platform Detection
- ☐ - ActiveX
- ✓ - Java Detection
- ☐ - Java
- ☐ - Download
- ☐ - Connected

Manual Installation

Web-based installation was unsuccessful. If you wish to install the Cisco AnyConnect Secure Mobility Client, you may download an installer package.

Install using the link below:

Windows Desktop

Alternatively, retry the automatic installation.

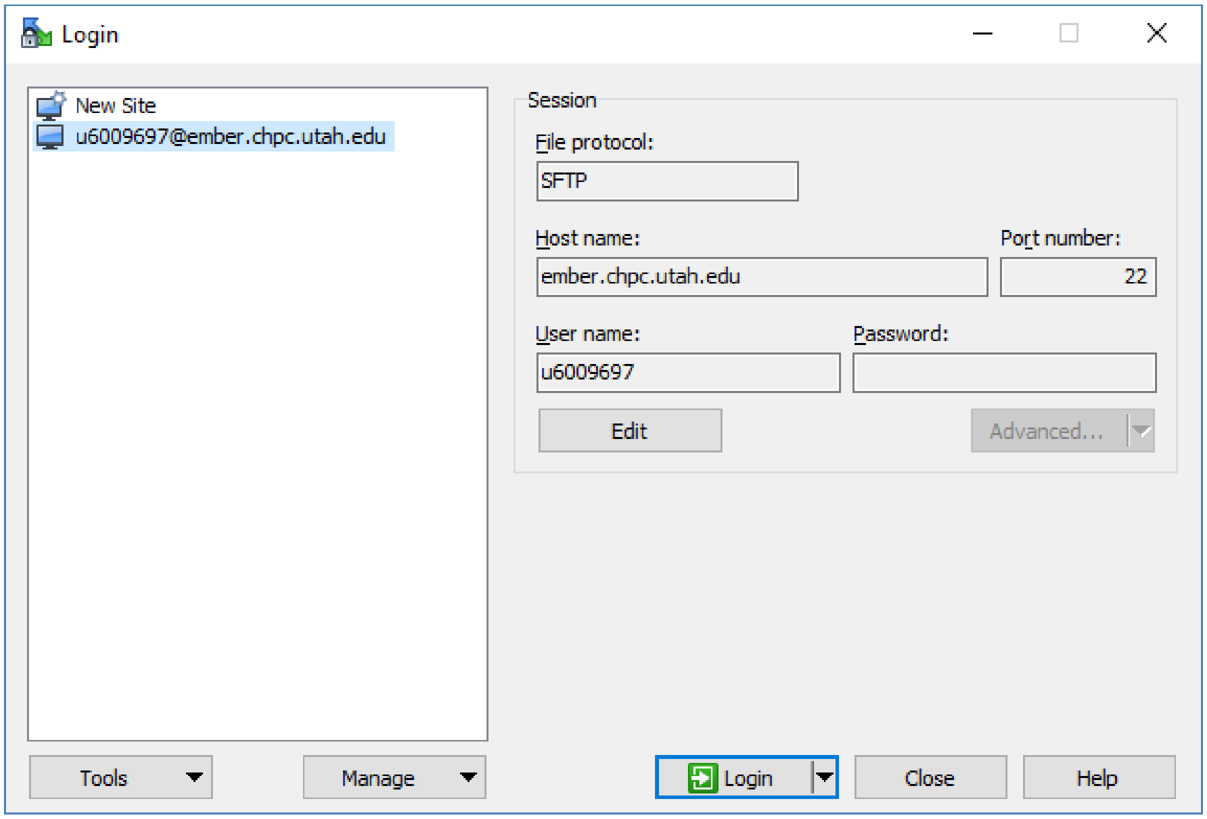
Help Download

Figure 6 CISCO's WebLaunch platform to download the client and automatically connect

Note that beginning 12/28/16, you will be required to use two-factor authentication. Visit http://it.utah.edu/2fa/#get-started to enroll. It is recommended that you register a mobile phone as your authentication device. There are apps for smart phones and tablets and you can get an automated phone call if you don't have either of those. Search for and download the Duo Mobile app in your phone's app store. Follow the onscreen instructions to register the phone. There are excellent step by step videos on the website for registering as well as logging in. Visit http://it.utah.edu/2fa/#training to view those videos.

5.2.Transferring Files

WinSCP is a simple click-and-drag application to transfer files from your own computer to a CHPC server. Direct your browser to https://winscp.net/eng/download.php, select either the full installation or a portable executable, and then either install or unzip it. Before running the program, follow the procedure to start a VPN to the University of Utah's network (see section 5.1 above). After opening the application you will be prompted for a host name, user name, and password. Again, if you are using the ember cluster, use ember.chpc.utah.edu in the host name, and your own uNID and password, as in Figure 7.



u6009697@ember.chpc.utah.edu

Session
File protocol: SFTP
Host name: ember.chpc.utah.edu
Port number: 22
User name: u6009697
Password:

Figure 7 Starting a new session with WinSCP

Once you are connected you have the ability to click and drag files from your computer to the remote drive. If you have written files on a Windows machine you will need to use the following command **`dos2unix [filename]`** to read it inside of a Linux environment. If you don't need the interactive GUI you can open a session in PuTTY from WinSCP by clicking on the ribbon button at the top of the application, as in Figure 8.

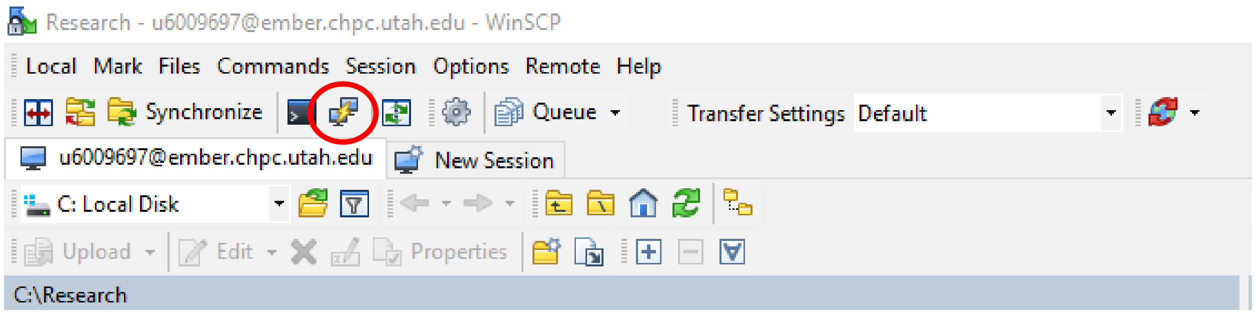

Research - u6009697@ember.chpc.utah.edu - WinSCP
Local Mark Files Commands Session Options Remote Help
Synchronize
Queue
Transfer Settings Default
u6009697@ember.chpc.utah.edu
New Session
C: Local Disk
Upload
Edit
Properties
C:\Research

Figure 8 Click on the PuTTY button to open a terminal

Files in your directory on the CHPC server are not backed up. Make sure to have copies on your own machine.

5.3. Interactive GUI

An interactive graphical user interface is useful for those who are unfamiliar with command line terminals. To install FastX2, in a browser go to http://ember1.chpc.utah.edu:3000 and click on the link labeled **Looking for the desktop client?** You can follow the CHPC instructions on their documentation website https://www.chpc.utah.edu/documentation/software/fastx2.php.

Once installed and opened, you can create a new SSH connection by clicking on the plus symbol in the upper right corner of the program (see Figure 9).

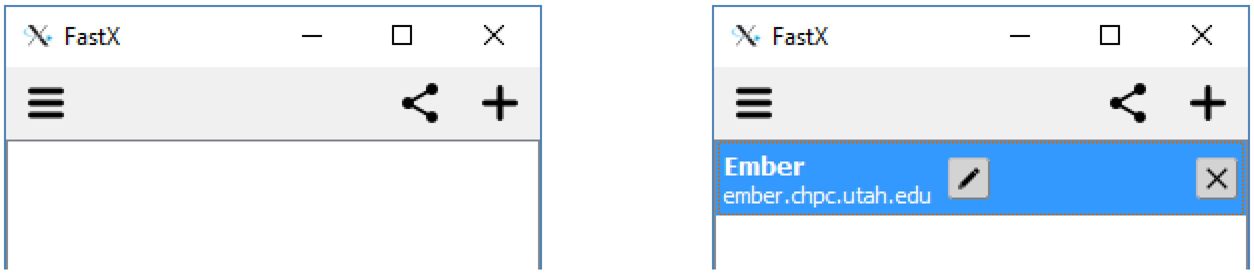
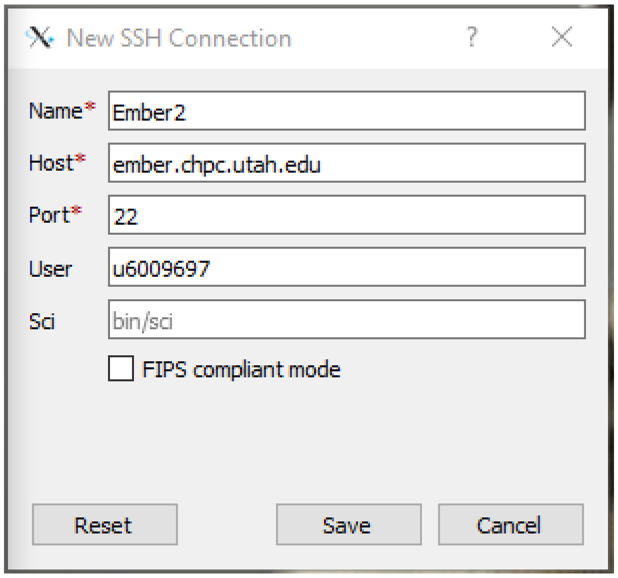


Figure 9 FastX2 connection list before and after adding a connection

USU has a partition on the Ember cluster, and therefore, I am using the Ember cluster for my tutorial. You must give the connection a name, but what you call it is unimportant. The Host field needs to be ember.chpc.utah.edu or the name of the cluster you want to use if it isn't Ember (see Figure 10).



Name* Ember2
Host* ember.chpc.utah.edu
Port* 22
User u6009697
Sci bin/sci
☐ FIPS compliant mode

Figure 10 Create a new FastX2 SSH connection

FastX2 will create another window where you can start a session (see Figure 11).

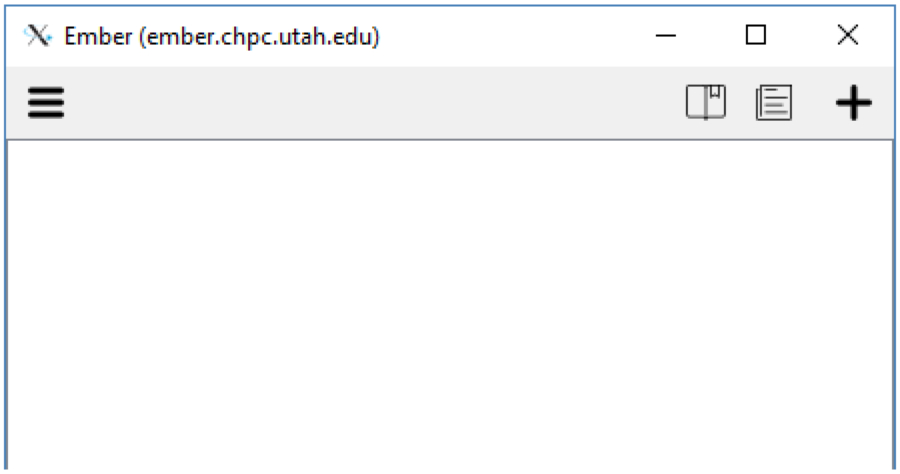
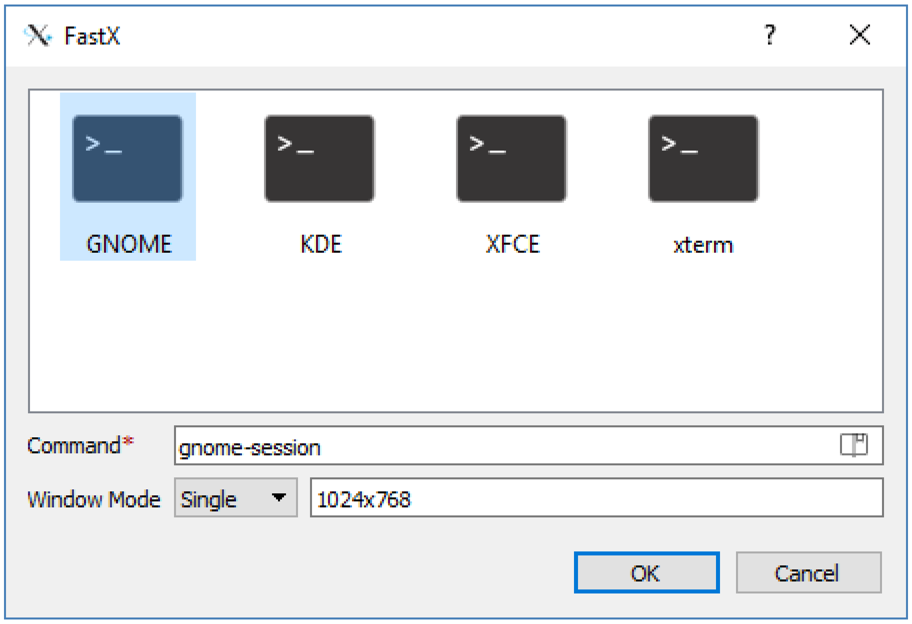


Figure 11 Connection awaiting a session

Start a new session by clicking on the plus icon in the upper right corner and selecting one of the interfaces (see Figure 12), I prefer GNOME because of its familiar look and feel. Once you select the desired one the command and window mode prompts will automatically fill in and you can click on OK.



Command* gnome-session
Window Mode Single 1024x768

Figure 12 Configure new session

Clicking OK in the dialog box (see Figure 12) will begin a session with a virtual desktop (see Figure 14) where you have a familiar look and feel and can access the terminal to open RStudio, submit batches to the cluster through the Simple Linux Utility for Resource Management (Slurm) job scheduler, and access the Internet. Figure 13 shows a connection with an active session.

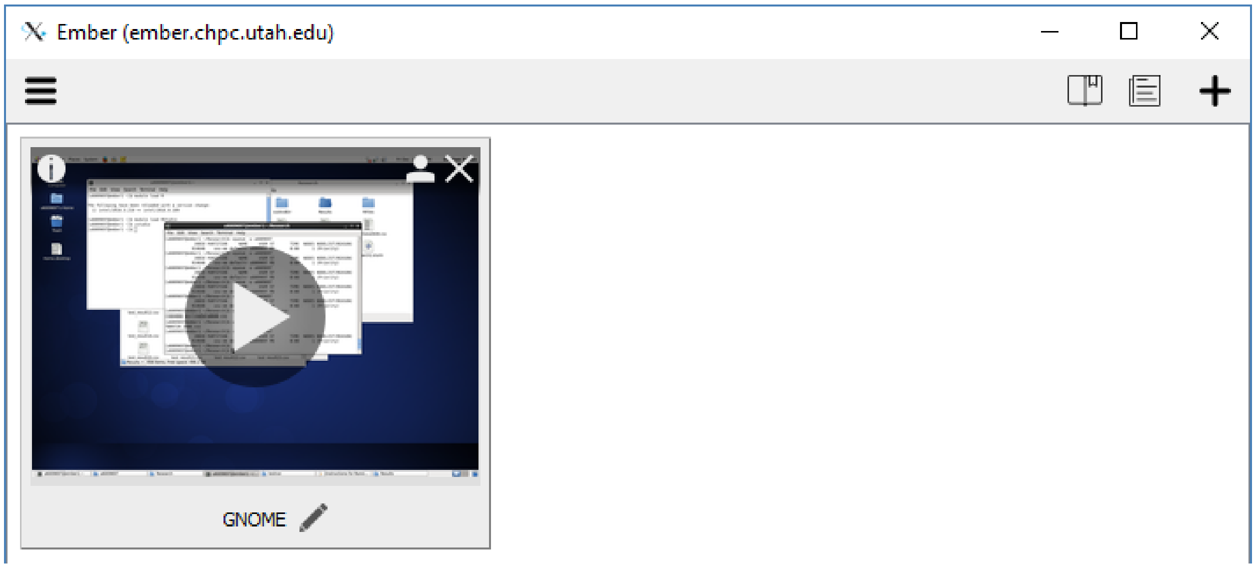


Figure 13 Connection with active session

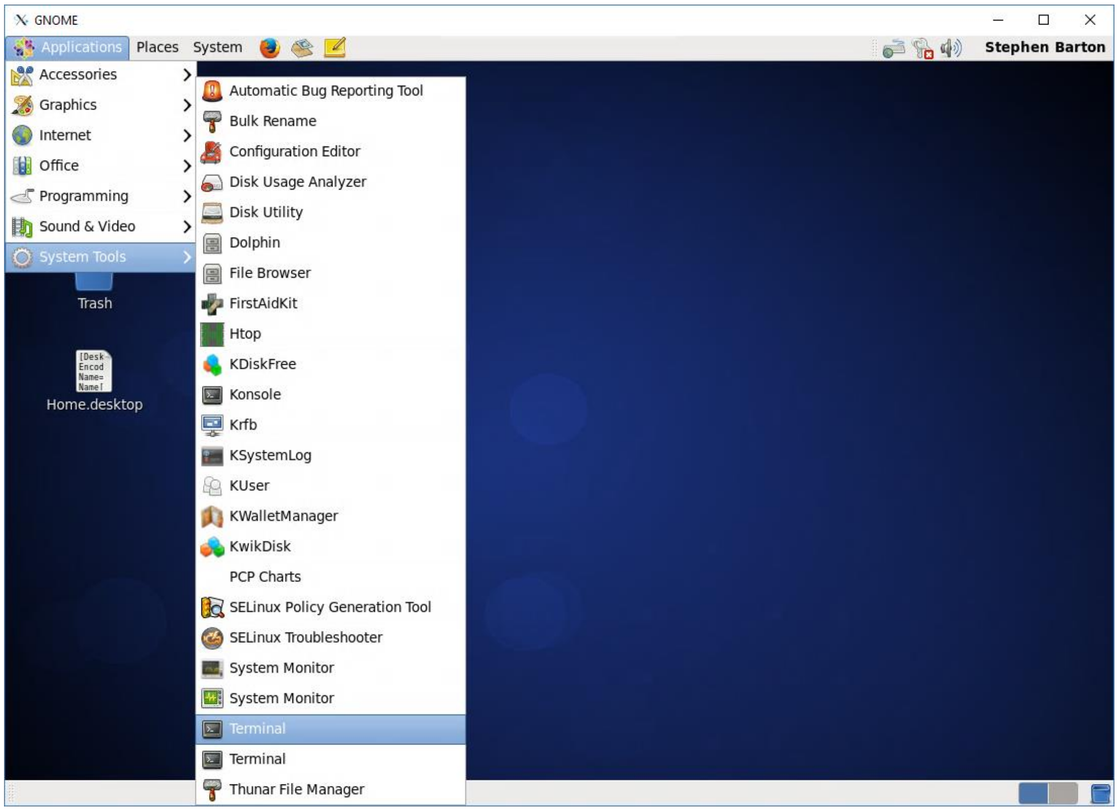
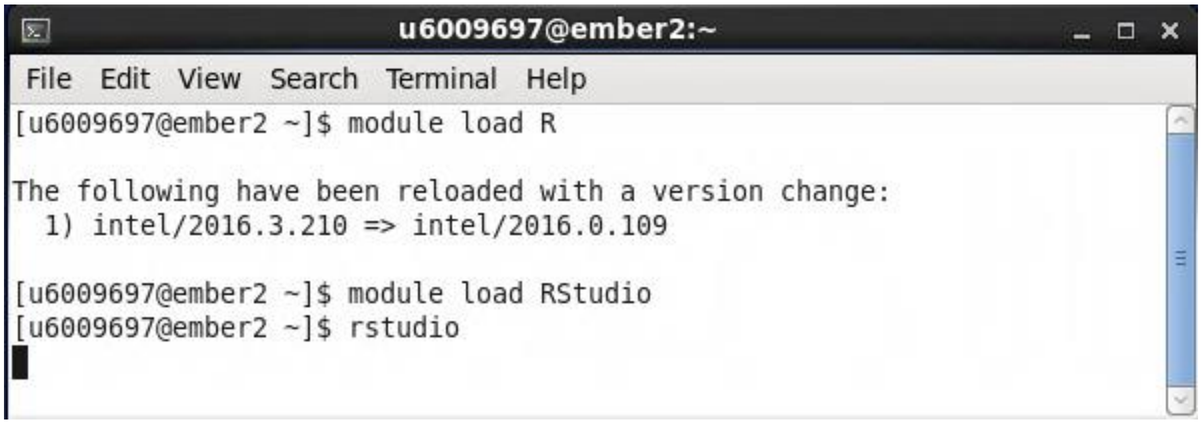


Figure 14 Virtual Desktop with the path to open a terminal window highlighted

You can open RStudio with shell commands within a terminal as shown in Figure 15. Remember the terminal is case sensitive.



```
[u6009697@ember2 ~]$ module load R

The following have been reloaded with a version change:
  1) intel/2016.3.210 => intel/2016.0.109

[u6009697@ember2 ~]$ module load RStudio
[u6009697@ember2 ~]$ rstudio
```

Figure 15 Commands to open RStudio

5.4.Submitting Batch Scripts

The computing power of the CHPC is best utilized by the compute nodes. Simply executing code on the virtual desktop in the interactive node will be only comparable to your own machine (see Figure 16).

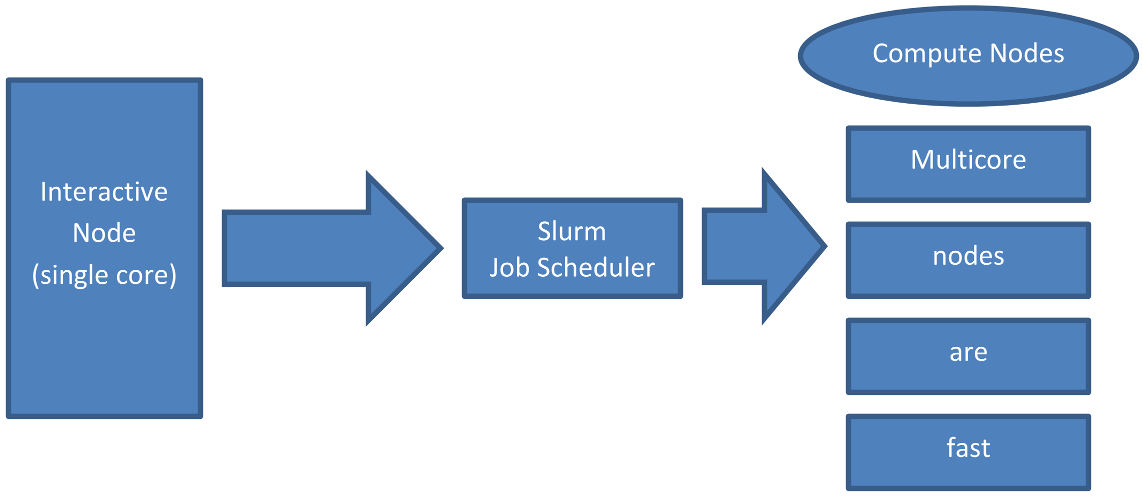


Figure 16 Basic flowchart showing structure of clusters

5.4.1. Dummy Example of Slurm

Simple Linux Utility for Resource Management, or Slurm, is a workload management application that CHPC uses to schedule jobs for the clusters and requires three files to run a batch:

- rscript.R - R script that holds all of the R commands
- my.conf - file that holds the command line arguments
- test.slurm - batch file that tells SLURM how to run the commands

The following is an example rscript.R

```
#!/usr/bin/env Rscript
library("R.utils")

sayHello <- function(){
   print('hello')
}

args <- cmdArgs()
cat("User command-line arguments used when invoking R:\n")
cat(str(args))

cat("\nInvoking sayhello\n")
sayHello()
```

Make note that you must load every library you wish to use. Furthermore, `cmdArgs()` is a very useful function for reading arguments from the command line into the script. It is not necessary, but makes it possible for incrementing through an index or a vector of parameters, for example, without needing to change the code for each repetition.

The following is an example configuration file, my.conf

```
0  Rscript /uufs/chpc.utah.edu/common/home/u6009697/R_Test/rscript.R 1 2
1  Rscript /uufs/chpc.utah.edu/common/home/u6009697/R_Test/rscript.R 21 31
```

Each line in my.conf is its own separate command line. Each of these lines runs on a separate core of the node on the partition that you pass it to. The USU partition (usu-em) consists of 16 nodes with 32 cores per node. This means you can have up to 32 lines in the my.conf file if you want to use this partition. The my.conf file has four components per line:

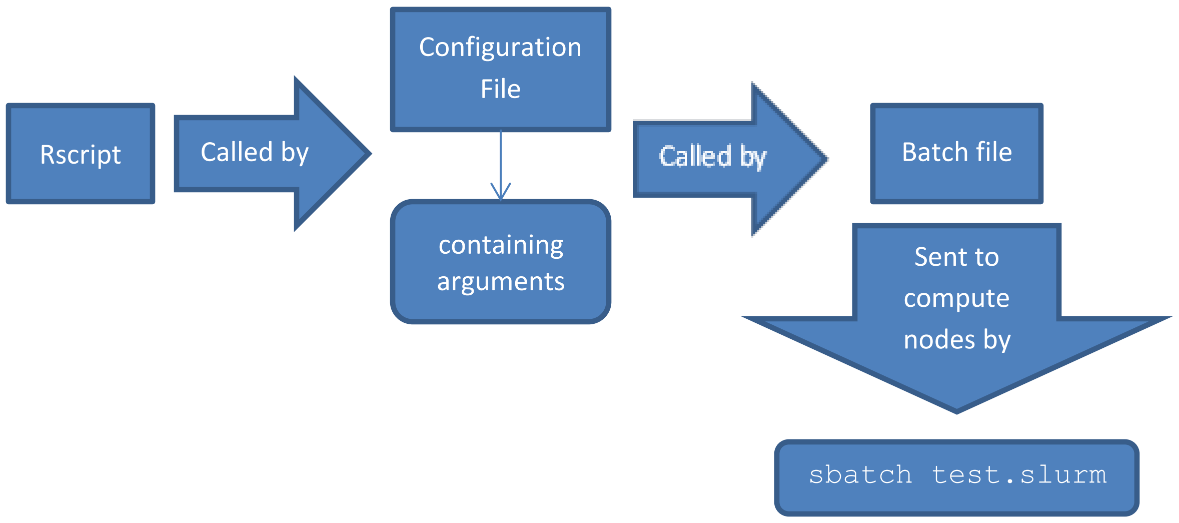
- Number of the line - This is a number from 0-31; start with 0 and progressively work your way up
- Rscript command - This tells it to run Rscript on the file specified in the path that follows
- Directory path - This must be the full path, not just the path from your home directory
- Arguments - These are the arguments you would type into the command line after Rscript if you were working on the command line in your home directory. It includes any arguments that you may need to pass to the script. In this example I am passing two arguments per line. The script reads these in to a list object called args

The following is an example batch file, test.slurm

```
#!/bin/bash
#SBATCH --job-name=myR-test
#SBATCH --time=00:30:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2

#SBATCH -o  out.%j
#SBATCH -e  err.%j

#SBATCH --mail-type=FAIL,BEGIN,END
#SBATCH --mail-user=willis.barton@aggiemail.usu.edu

#SBATCH --account=usu-em
#SBATCH --partition=usu-em

module load R/3.2.3.omp
srun --multi-prog my.conf
```

This batch file (test.slurm) holds the commands that tell Slurm what to do. The majority of the file tells Slurm how to set up the run. All of the lines with a # in front are commands that are needed. If you want to change these contact CHPC to make sure you get it right. You can email them your file and they can proof-read it for you.

On line 2 is the name of the job. Line 3 contains the upper limit approximation of the time the job is expect to take, there is a hard limit of 72 hours on general cluster nodes. Identify the number of tasks in your configuration file on line 5. Lines 7 and 8 will generate output and error files with the names out.[jobID] and err.[jobID], the output file will contain anything in the script that is written to the console and the error file will contain any warning messages and error messages. If you wish to receive an email when your job has started, completed, or failed include lines 10 and 11 with your email address. You will get an email with the subject line identifying what action has been taken and relevant time stamps. For using the USU partition include lines 13-14. Line 16 identifies the application you will be using. Finally the command `srun --multi-prog my.conf` on line 17 tells Slurm to run the command lines in the my.conf file. The `--multi-prog` command tells it that there will be multiple calls running at the same time. Go to https://www.chpc.utah.edu/documentation/software/slurm.php for more documentation on Slurm. Once your files are ready to go send them to SLURM with the following command in the terminal: `sbatch test.slurm`



After submitting the batch you can exit the terminal session and/or close your virtual desktop and the job will queue until it begins. If you indicated you want emails, you'll receive notification of when it started and the time it spent in the job queue. When it ends, you'll receive an additional notification with the exit status and elapsed time.

5.4.2. Actual Random Forest example using Slurm

The code used in my second experiment (see section 4.2) is contained in A.1R Code. It expects one command line argument passed from the configuration file that is dynamically written to take advantage of the total number of cores on a node (see A.2). This is helpful if I weren't using the USU partition as there are nodes with 8, 16, 24, or other numbers of cores. And finally, the batch file in A.3 creates a working directory with copies of the script and data, and then creates and runs the configuration file. For each job of the 10 jobs, I incremented the chunk object in line 54 in A.1 by one to cycle through each group of 32 models.

6. BIBLIOGRAPHY

L. Breiman. Random forests. Machine Learning, 45(1): 5–32, 2001.

L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone. Classification and Regression Trees. Wadsworth, Behrnont, CA (1984)

A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18--22.

APPENDIX

A.1. R Code

```
#!/usr/bin/env Rscript
########## Airline script with timing ##########
# Load utils package
if(require("utils")){
  print("utils is loaded correctly")
} else {
  print("trying to install utils")
  install.packages("utils")
  if(require(utils)){
    print("utils installed and loaded")
  } else {
    stop("could not install utils")
  }
}
# Load stats package
if(require("stats")){
  print("stats is loaded correctly")
} else {
  print("trying to install stats")
  install.packages("stats")
  if(require(doRNG)){
    print("stats installed and loaded")
  } else {
    stop("could not install stats")
  }
}
# Load randomForest package
if(require("randomForest")){
  print("randomForest is loaded correctly")
} else {
  print("trying to install randomForest")
  install.packages("randomForest")
  if(require(randomForest)){
    print("randomForest installed and loaded")
  } else {
    stop("could not install randomForest")
  }
}
# Load R.utils package
if(require("R.utils")){
  print("R.utils is loaded correctly")
} else {
  print("trying to install R.utils")
  install.packages("R.utils")
  if(require(R.utils)){
    print("R.utils installed and loaded")
  } else {
    stop("could not install R.utils")
  }
}
# Retrieve command line argument in the configuration file and
# increment as needed. Count chunks starting at 0, each node in the
# usu-em partition has 32 cores.
chunk = 0
arg <- as.numeric(cmdArgs()) + 1 + 32*chunk
arg
if(arg>312)stop("All done with loops")

# Read 2008.csv in and clean up
fileIn = "/uufs/chpc.utah.edu/common/home/u6009697/Research/2008.csv"
fileInColumns = c("NULL", rep("integer", 7), rep("NULL", 3),
                  rep("integer", 5), "NULL", "NULL", rep("integer",
                  3), rep("NULL", 8))
data <- read.csv(fileIn, colClasses = fileInColumns)
# Ignore flights that got completely canceled or diverted.
data <- data[complete.cases(data),]
# Create response variable, call it a factor variable to force it to
# do classification
data.late15 <- as.factor(data$ArrDelay >= 15)
data <- data[,-11]


class.sum=function(truth,predicted){
  xt=table(truth,round(predicted+0.000001))
  pcc=sum(diag(xt))/sum(xt)
  spec=xt[1,1]/sum(xt[1,])
  sens=xt[2,2]/sum(xt[2,])
  return(c(PCC = pcc, Specificity = spec, Sensitivity = sens))
}

# my_rf() is a wrapper for randomForest() and returns a data.frame of
# accuracy of the model generated and
my_rf <- function(input=data, output=data.late15, maxnodes, mtry){
  timer <- proc.time()
  temp <- randomForest(x=input, y=output, ntree=500, replace=FALSE,
                       sampsize=.05*nrow(input), keep.forest=FALSE,
                       maxnodes=maxnodes, mtry=mtry)
  timer <- proc.time() - timer
  accuracy <- class.sum(as.numeric(output),
                        as.numeric(temp$predicted))
  return(data.frame(mtry=temp$mtry,
                    maxnodes=maxnodes,
                    sampsizePercent=5,
                    ntree=temp$ntree,
                    error.rate=1-accuracy[1],
                    false.negative=1-accuracy[2],
                    false.positive=1-accuracy[3],
                    time = timer[3],
                    row.names=NULL))
```

```
}

# Generate the list of model arguments
mtry <- c(3:14)
maxnodes <- c(2,3,4,5,6,8,10,14,18,22,26,30,60,100,200,300,400,500,
              1000,2000,3000,4000,5000,10000,15000,20000)
loops <- expand.grid(mtry=mtry, maxnodes=maxnodes)

loops[arg,]

set.seed(925)
result <- my_rf(  input    = data
                , output   = data.late15
                , maxnodes = loops$maxnodes[arg]
                , mtry     = loops$mtry[arg]
                )

fileOut=paste0("/uufs/chpc.utah.edu/common/home/u6009697/Research/Resu
                lts/test_result", arg, ".csv")
write.csv(result, file=fileOut, row.names=FALSE)
```

A.2. Configuration File

```
#!/bin/bash

# NOTE:
#   EXE : rwapper.sh
#   TASK_ID     : Id of the task
#   SCRATCH_DIR : EACH task has its own scratch directory
#   SCRIPT_DIR  : Script is identical for each task => Same directory
for ALL tasks
#   OUT_DIR     : EACH task has its own output directory

# Retrieve variable from the command line
START_DIR=$PWD
EXE=$0
TASK_ID=$1
SCRATCH_DIR=$2
SCRIPT_DIR=$3
OUT_DIR=$4

if [ "$#" -ne 4 ] ; then
     echo "  ERROR: Command line needs 4 parameters"
     echo "  Current arg list: $@"
else
     echo "  TaskID:$TASK_ID started at `date`"
     mkdir -p $SCRATCH_DIR
     cd $SCRATCH_DIR
     # Copy content SCRIPT_DIR to SCRATCH_DIR
     cp -pR $SCRIPT_DIR/* .
     Rscript airlinescript.R $TASK_ID > $TASK_ID.out 2>&1

     # Copy results back to OUT_DIR
     mkdir -p $OUT_DIR
     cp -pR * $OUT_DIR
     cd $START_DIR
     rm -rf $SCRATCH_DIR
     echo "  TaskID:$TASK_ID ended at `date`"
fi
```

A.3. SLURM Batch File

```
#!/bin/bash
#SBATCH --time=08:00:00
#SBATCH --nodes=1

#SBATCH --mail-type=FAIL,BEGIN,END
#SBATCH --mail-user=willis.barton@aggiemail.usu.edu
#SBATCH -o out.%j
#SBATCH -e err.%j

#SBATCH --account=usu-em
#SBATCH --partition=usu-em

#SBATCH --job-name=airline

# Job Parameters
export EXE=./rwrapper.sh
export WORK_DIR=~/Research
export SCRATCH_DIR=/scratch/local/$SLURM_JOBID
export SCRIPT_DIR=$WORK_DIR/RFiles
export OUT_DIR=$WORK_DIR/`echo $UUFSCELL | cut -b1-4`/$SLURM_JOBID

# Load R (version 3.2.3)
module load R/3.2.3.omp

# Run an array of serial jobs
export OMP_NUM_THREADS=1

echo " Calculation started at:`date`"
echo " #$SLURM_TASKS_PER_NODE cores detected on `hostname`"

# Create the my.config.$SLURM_JOBID file on the fly
for (( i=0; i < $SLURM_TASKS_PER_NODE ; i++ )); \
   do echo $i $EXE $i $SCRATCH_DIR/$i $SCRIPT_DIR $OUT_DIR/$i ; \
done > my.config.$UUFSCELL.$SLURM_JOBID

# Running a task on each core
cd $WORK_DIR
srun --multi-prog my.config.$UUFSCELL.$SLURM_JOBID

# Clean-up the root scratch dir
rm -rf $SCRATCH_DIR

echo "  Calculation ended at:`date`"
```