STAT 215A Fall 2017
Week 8

Rebecca Barter
10/13/2017

Speeding up computation

Easy ways to speed up computation:

1. Don’t repeatedly re-compute objects (e.g. a similarity matrix) that really only need to be computed once
2. Don’t define objects unnecessarily
3. In R: use apply functions (base R) and map functions (purr) instead of base for-loops

Less easy ways to speed up computation

4. Parallelize: using the SCF cluster or using multiple cores (or threads) in your laptop
5. Write functions in faster languages (C++) and read those into R

Doing things simultaneously: Parallelization

Parallelization

Parallelization has a few different flavors:

- Multicore processors (the typical MacBook has ? cores)
- Computer clusters
- GPUs
- Other fancy things I know nothing about

Parallelization

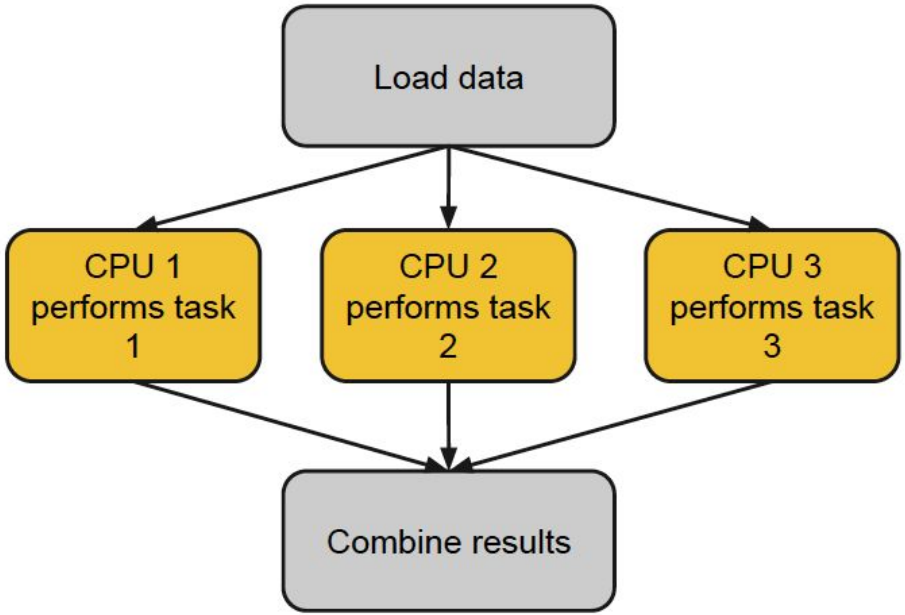
Chris Paciorek (of STAT 243) is a local expert. The material today is mostly his.

Useful resources prepared (mostly) by Chris:

http://statistics.berkeley.edu/computing/training

Specifically: https://github.com/berkeley-scf/tutorial-parallel-basics

Local parallelization



Parallel tasks cannot talk to one another.

We usually parallelize to speed up computation, e.g. by

- doing loops simultaneously, or
- computing on multiple subsets of a large dataset simultaneously

Local parallelization

How would you parallelize the following tasks:

- Obtaining a bootstrapped estimate of the mean
- OLS regression
- The K-means algorithm

Local parallelization

How would you parallelize the following tasks:

- Obtaining a bootstrapped estimate of the mean
- OLS regression
- The K-means algorithm

We will only focus on **embarrassingly parallel** tasks

Parallelization in R: foreach

There are a few key functions in R for parallelization:

The foreach package

```
library(foreach)

library(doParallel)

nCores <- 4 # to set manually

registerDoParallel(nCores)

result <- foreach(i = 1:nSub) %dopar% {
          # stuff to run in each iteration
  }
```

Parallelization in R: parLapply

There are a few key functions in R for parallelization:

parallel package (parLapply, parSapply)

```
library(parallel)

nCores <- 4   # to set manually

cl <- makeCluster(nCores)

result <- parLapply(cl, X = data, FUN = fun)
```

Parallelization in R

See example in `parallel.R`

To check how many cores your machine has:

```
library(parallel)

detectCores(all.tests = FALSE, logical = TRUE)
```

Using the SCF cluster

Using the SCF cluster

If you haven't already, sign up for an SCF account at

https://scf.berkeley.edu/account

Information on submitting jobs to the cluster can be found here:

http://statistics.berkeley.edu/computing/servers/cluster

Using the SCF cluster

1. SSH into an SCF machine
2. Copy your files to that computer
3. Set up a shell script that runs your job (e.g. `shell_example.sh`)
4. Submit your job using SLURM something like:

```
sbatch shell_example.sh
```

SSH: enter a machine on the SCF cluster

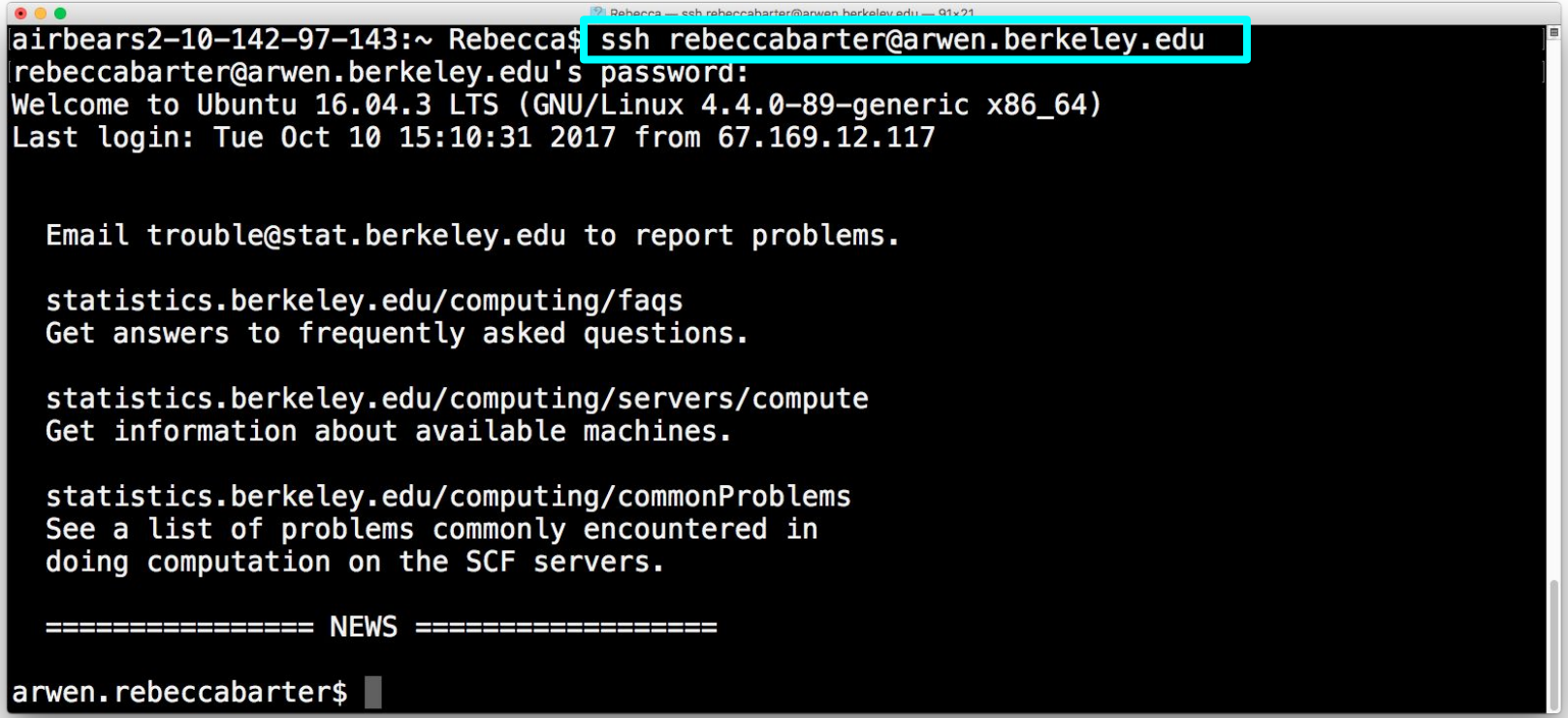
The SCF cluster contains the following LOTR-inspired machines that you can ssh into:

```
arwen, beren, bilbo, gandalf, gimli, legolas, pooh, radagast, roo,
shelob, springer, treebeard
```

To SSH into a machine, in your terminal type:

`ssh rebeccabarter@arwen.berkeley.edu`
(obviously using your SCF username and password instead of mine!)

SSH: enter a machine on the SCF cluster



Shell scripts: toy example

Example shell_example.sh:

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --nodes=1

R CMD BATCH --no-save job.R job.out
```

Make sure the shell script is executable (change permissions)

```
chmod 755 shell_example.sh
```

To run a shell script in the terminal write

```
./shell_example.sh
```

Copying files from your local machine to the remote machine

There are several ways to do this. The easiest is to **clone a github repo** on the remote machine.

Other options can be found here:

http://statistics.berkeley.edu/computing/copying-files

Submitting your job to the SCF cluster (from a LOTR machine)

```
sbatch shell_example.sh
```

To cancel your job if you made a mistake:

```
scancel {job_id}
```

Check that your jobs are running as expected on the SCF cluster

```
squeue
```

To see only my jobs:

```
squeue -u rebeccabarter
```

To see how many CPUs I am using

```
squeue -u rebeccabarter -o"%.7i %.9P %.8j %.8u %.2t %.10M %.6D %C"
```

Writing faster functions: Rcpp and C++

Rcpp

Rcpp_demo.R:

```r
library('Rcpp')
sourceCpp('Rcpp_demo.cpp')

x = rnorm(1e7)
y = rnorm(1e7)
z <- cbind(x, y)

DistanceCPP(x, y)
```

Rcpp_demo.cpp:

```cpp
#include <Rcpp.h>

Rcpp::NumericVector DistanceCPP(Rcpp::NumericVector x, Rcpp::NumericVector
  // Calculate the euclidian distance between <x> and <y>.

  // C++ requires initialization of variables.
  double result = 0.0;
  // This is the length of the x vector.
  int n = x.size();
  // Check that the size is the same and return NA if it is not.
  if (y.size() != n) {
    Rcpp::Rcout << "Error: the size of x and y must be the same.\n";
    return(Rcpp::NumericVector::create(NA_REAL));
  }
  for (int i = 0; i < n; i++) {
    result += pow(x[i] - y[i], 2.0);
  }
  // We need to convert between the double type and the R numeric vector ty
  return Rcpp::NumericVector::create(sqrt(result));
}
```

Exercises

Exercises

1. Use foreach to parallelize k-means with random different starting points.
2. Use Rcpp to make the binary matrix from lab2 for a particular question.
3. Run something on the SCF.