C² Lab Demo

Scaling your code for big data

Seth Frey

Scaling your code for big data

- Motivation
- Overall strategy
- Managing the types of bottleneck (CPU, RAM, cache, etc)
- Summary

Scaling your code for big data

- **Motivation**
- Overall strategy
- Managing the types of bottleneck (CPU, RAM, cache, etc)
- Summary

Motivation

- Why scalable code?
- Why not?

Motivation

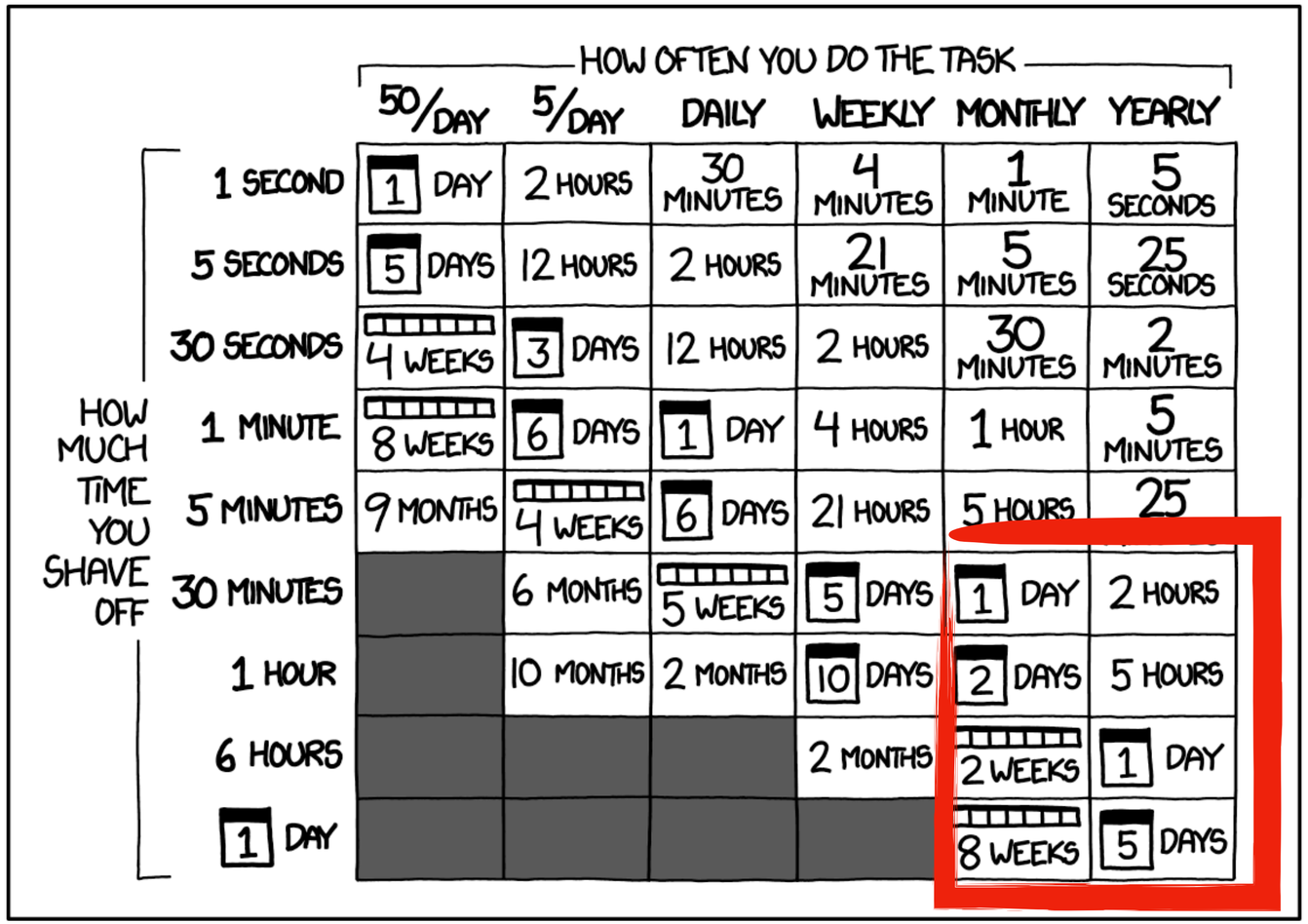
- **Why scalable code?**
  - Basic literacy of big data
  - Saves you time
  - Pays for itself
  - Makes you better at coding
  - Makes you better at computering

Motivation

- **Why not?**
  - There is such a thing as overkill
  - 80/20 rule

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?

(ACROSS FIVE YEARS)



Learn as you go

- The purpose of these slides is not to teach you all the jargon
- It's to tell you what jargon to learn for what problem

Scaling your code for big data

- Motivation
- **Overall strategy**
- Managing the types of bottleneck (CPU, RAM, cache, etc)
- Summary

Overall strategy

- "Big data"
- Linear vs exponential problem growth
- Start your solutions small
- Know when to stop
- Rough arc

"Big data"

- For our purposes, "big" means your laptop can't handle it.

Linear v. exponential

- **Linear scaling**
  - 100x more tweets takes 100x longer
- **Exponential**
  - 2x more tweets takes 100x longer

Linear v. exponential

- **Linear scaling**
  - Bottleneck is going to tend to be CPU
- **Exponential**
  - Bottleneck is going to tend to be RAM
  - These problems require more talent and experience to scale

Start small

- Get your whole pipeline working on just 10 data points
  - Then on 100, and fix things
  - Then back to (newly problematic) 10
  - Then 1000
  - (and back to 10)
  - Keep adding 0's

Start small

- Get your whole pipeline working on just 10 data points
- Vastly accelerates debugging
- Forces you to stay close to the data
- Good engineering

Stopping rule

- 80/20 rule when possible
- You are not a software engineer
  - don't get it beautiful, get it done
- *Prototypes not products*
  - Benefit of published code is 95% scientific/ethical, 5% useful

Scaling arc

Your data is in the …

—**10s–100s**: *analyze by hand or any program*
—**100s–10000s**: decent *R/Python, good Excel*
—**10000–low 10 millions**: *good R/Python*
—**millions–10 millions**: *SQL and/or better computer*
—**ten millions–high 100 millions**: *SQL column/vertical-store*
—**high 100 millions–trillions**: *cluster*

Your analysis is

—**Low complexity**: You're OK up to upper side of bound
—**High complexity**: Stay at lower side of bound

Scaling your code for big data

- Motivation
- Overall strategy
- **Managing the types of bottleneck** (CPU, RAM, cache, etc)
- Summary

Types of bottleneck

- Bottleneck at CPU
- Bottleneck at RAM
- Bottleneck at disk space
- also
  - Bottleneck at network
  - Bottleneck at file IO

Bottleneck at CPU

What does it look like?

- Your code takes days/hours and it would be nice if it took hours/minutes/seconds
- Your code runs at 100% CPU
  - If less than 100%, bottleneck is elsewhere

Bottleneck at CPU

What should you do?

Bottleneck at CPU

What should you do?

- **Can you wait?**

Bottleneck at CPU

What should you do?

- **Can you wait?**
  - **Yes.**

Bottleneck at CPU

What should you do?

- **Can you wait?**
  - **Yes.**
    1. Leave code notebook env
       - Learn shell
    2. Move to a PC you can defend
    3. Save intermediate files

Bottleneck at CPU

What should you do?

- **Can you wait?**
  - **No.**

Bottleneck at CPU

Can't wait

1. Learn "*benchmarking code*"
2. Then learn to replace for loops with "*vectorization*" (esp. in R)
   —a.k.a "*vector operations*"
3. Then learn "*code profiling*"
4. Then move to a better computer
   —and learn shell
5. Then learn simple "*parallelism"*
   —Start with parallelism on the whole dataset (run your script once)
6. Or learn "*database*"/"*SQL"* (trade CPU for disk)
7. Or learn enough "*C++"* to rewrite your slowest fn
8. Then learn to move to a cluster (at UCD: "Peloton")
   —parallelism on shattered dataset (run your script lots of times)
   —"*embarrassingly parallel*" problems

Bottleneck at RAM

What does it look like?

- Your computer crashes or grinds to a halt
- Other apps start to fail

Bottleneck at RAM

What should you do?

Bottleneck at RAM

What should you do?

- **Do you really need all that data?**

Bottleneck at RAM

- What should you do?
  - Do you really need all that data?
    - No.

Bottleneck at RAM

What should you do?

- **Do you really need all that data?**
  - **No.**
    1. Sample, random or not
    2. Use half to explore and half to confirm
       - Could be exponentially faster

Bottleneck at RAM

- What should you do?
  - Do you really need all that data?
    - Yes.

Bottleneck at RAM

Need all the data

1. First use "*iterators*" ("*streaming"* instead of loading)
2. Then learn to pull your script apart into several linked in a pipeline, each writing its output to file the input to the next
   — Saving "intermediate progress" files from several steps
   —("Trade disk for RAM")
3. Fiddling with swap on SSD ("*swap on SSD*")
4. Then move to a better (high RAM) computer
   —and learn shell
5. Then move to cluster
   —"real" parallelism

Bottleneck at disk space

What does it look like?

- You can't download all your data
- Your OS warns you there's no more space
- (If it happen while your code is running, it's likely RAM, not disk)

Bottleneck at disk space

What should you do?

1. Buy a bigger drive
2. Learn to load/stream files while still zipped
3. Learn to load/stream files over network one at a time
4. Learn to write pipelines that store "intermediate progress" files. Store originals remotely
   ——and learn shell
5. Break input up from one large files to lots of small files over lots of nested folders
   ——Write pipeline around single "observations"

Bottleneck at network

What does it look like?

- You are webscraping
- Code is slower than it should be
  - (*it's spending most of its time downloading*)
- CPU underutilized
- Your internet may be slow

Bottleneck at network

What should you do?

1. Get remote account close to data
   - Have to know/be data owner
2. Store raw data locally
   - On a scraper: one step for downloading HTML, one for scraping that local HTML
     - "Don't scrape the web, scrape your disk"
     - (Trade network for disk)
   - Download once, chug often
3. Download compressed versions, unzip locally

Bottleneck at file IO

What does it look like?

- CPU underutilized
  - at 10–40% instead of 90–100%

Bottleneck at file IO

What should you do?

1. Get all data local and onto fast drives
   —"*SSD*", high RPM, and "*RAID*" storage
2. Get all data local and onto fast drives
   —"scratch" on a cluster
3. Load once, write once
   —and/or move to high RAM machine
   —(trade IO for RAM)
4. Pat yourself on the back
   —This is a pro problem
   —Having it often means you've solved all the easier ones

Scaling your code for big data

- Motivation
  - The skill side of big data
- Overall strategy
  - Arc
- Managing the types of bottleneck
  - (CPU, RAM, disk, network, IO)

Scaling your code for big data

Seth Frey