Nuts and Bolts

Matthew Gentzkow
Jesse M. Shapiro

Chicago Booth and NBER
Introduction

- We have focused on the statistical / econometric issues that arise with big data
- In the time that remains, we want to spend a little time on the *practical* issues...
Introduction

- We have focused on the statistical / econometric issues that arise with big data
- In the time that remains, we want to spend a little time on the practical issues...
  - E.g., where do you actually put a 2 TB dataset?
Introduction

- We have focused on the statistical / econometric issues that arise with big data
- In the time that remains, we want to spend a little time on the practical issues...
  - E.g., where do you actually put a 2 TB dataset?
- Goal: Sketch some basic computing ideas relevant to working with large datasets.
Introduction

- We have focused on the statistical / econometric issues that arise with big data.
- In the time that remains, we want to spend a little time on the practical issues...
  - E.g., where do you actually put a 2 TB dataset?
- Goal: Sketch some basic computing ideas relevant to working with large datasets.
- Caveat: We are all amateurs.
The Good News

- Much of what we’ve talked about here you can do on your laptop
  - Your OS knows how to do parallel computing (multiple processors, multiple cores)
  - Many “big” datasets are < 5 GB
  - Save the data to local disk, fire up Stata or R, and off you go...
### How Big is Big?

<table>
<thead>
<tr>
<th>Dataset Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Congressional record text (1870-2010)</td>
<td>≈50 GB</td>
</tr>
<tr>
<td>Congressional record pdfs (1870-2010)</td>
<td>≈500 GB</td>
</tr>
<tr>
<td>Nielsen scanner data (34k stores, 2004-2010)</td>
<td>≈5 TB</td>
</tr>
<tr>
<td>Wikipedia (2013)</td>
<td>≈6 TB</td>
</tr>
<tr>
<td>20% Medicare claims data (1997-2009)</td>
<td>≈10 TB</td>
</tr>
<tr>
<td>Facebook (2013)</td>
<td>≈100,000 TB</td>
</tr>
<tr>
<td>All data in the world</td>
<td>≈2.7 billion TB</td>
</tr>
</tbody>
</table>
Outline

- Software engineering for economists
- Databases
- Cluster computing
- Scenarios
Software Engineering for Economists
Motivation

- A lot of the time spent in empirical research is writing, reading, and debugging code.
- Common situations...
Broken Code

do C:\demo\newspaper.do

use "C:/demo/newspapers.dta"
file C:/demo/newspapers.dta not found
r(601);

end of do-file

r(601);
Incoherent Data

```
use "C:/demo/statepop.dta"

merge 1:1 state using "C:/demo/statename"
variable state does not uniquely identify observations in the master data
r(459);
```
Rampant Duplication

```
use "C:/demo/data.dta", clear
regress probgen coll pharm hlthworker hlthmajor scimajor i.market
regress probgen coll pharm hlthworker hlthmajor scimajor i.year
regress probgen coll pharm hlthworker hlthmajor scimajor i.market i.year
regress probgen coll pharm hlthworker hlthmajor scimajor i.market i.year age
regress probgen coll pharm hlthworker hlthmajor scimajor i.market income
regress probgen coll pharm hlthworker hlthmajor scimajor i.market i.product
regress probgen coll pharm hlthworker hlthmajor scimajor i.year age income
regress probgen coll pharm hlthworker hlthmajor scimajor i.market age income
regress probgen coll pharm hlthworker hlthmajor scimajor i.market i.product i.year age income
regress probgen coll pharm hlthworker hlthmajor scimajor i.product i.year age income
regress probgen coll pharm hlthworker hlthmajor scimajor i.market i.year income
```
Replication Impossible
Tons of Versions
This Talk

- We are not software engineers or computer scientists.
- But we have learned that most common problems in social sciences have analogues in these fields and there are standard solutions.
- Goal is to highlight a few of these that we think are especially valuable to researchers.
- Focus on incremental changes: one step away from common practice.
Automation
Raw Data

Data from original source...
<table>
<thead>
<tr>
<th>county</th>
<th>state</th>
<th>year</th>
<th>chip_sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1940</td>
<td>1012</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1941</td>
<td>1020</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1942</td>
<td>1034</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1943</td>
<td>1058</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1944</td>
<td>1085</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1945</td>
<td>1148</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1946</td>
<td>1205</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1947</td>
<td>1287</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1948</td>
<td>1299</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1949</td>
<td>1344</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1950</td>
<td>1365</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1951</td>
<td>1397</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1952</td>
<td>1455</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1953</td>
<td>1501</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1954</td>
<td>1582</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1955</td>
<td>1656</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1956</td>
<td>1723</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1957</td>
<td>1795</td>
</tr>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1958</td>
<td>1878</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>county</th>
<th>state</th>
<th>year_tv_introduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autauga</td>
<td>AL</td>
<td>1940</td>
</tr>
<tr>
<td>Baldwin</td>
<td>AL</td>
<td>1935</td>
</tr>
<tr>
<td>Barbour</td>
<td>AL</td>
<td>1942</td>
</tr>
<tr>
<td>Bibb</td>
<td>AL</td>
<td>1942</td>
</tr>
<tr>
<td>Blount</td>
<td>AL</td>
<td>1939</td>
</tr>
<tr>
<td>Bullock</td>
<td>AL</td>
<td>1945</td>
</tr>
<tr>
<td>Butler</td>
<td>AL</td>
<td>1942</td>
</tr>
<tr>
<td>Calhoun</td>
<td>AL</td>
<td>1936</td>
</tr>
<tr>
<td>Chambers</td>
<td>AL</td>
<td>1940</td>
</tr>
<tr>
<td>Cherokee</td>
<td>AL</td>
<td>1939</td>
</tr>
<tr>
<td>Chilton</td>
<td>AL</td>
<td>1941</td>
</tr>
<tr>
<td>Choctaw</td>
<td>AL</td>
<td>1942</td>
</tr>
<tr>
<td>Clarke</td>
<td>AL</td>
<td>1940</td>
</tr>
<tr>
<td>Clay</td>
<td>AL</td>
<td>1941</td>
</tr>
<tr>
<td>Cleburne</td>
<td>AL</td>
<td>1943</td>
</tr>
<tr>
<td>Coffee</td>
<td>AL</td>
<td>1936</td>
</tr>
<tr>
<td>Colbert</td>
<td>AL</td>
<td>1937</td>
</tr>
<tr>
<td>Conecuh</td>
<td>AL</td>
<td>1940</td>
</tr>
<tr>
<td>Coosa</td>
<td>AL</td>
<td>1943</td>
</tr>
</tbody>
</table>
Manual Approach

- Open spreadsheet
- Output to text files
- Open Stata
- Load data, merge files
- Compute log(chip sales)
- Run regression
- Copy results to MS Word and save
Manual Approach

Two main problems with this approach

- Replication: how can we be sure we’ll find our way back to the exact same numbers?
- Efficiency: what happens if we change our mind about the right specification?
Semi-automated Approach

Problems

- Which file does what?
- In what order?
Fully Automated Approach

File: rundirectory.bat

- stattransfer export_to_csv.stc
- statase -b mergefiles.do
- statase -b cleandata.do
- statase -b regressions.do
- statase -b figures.do
- pdflatex tv_potato.tex

- All steps controlled by a shell script
- Order of steps unambiguous
- Easy to call commands from different packages
Make

- Framework to go from source to target
- Tracks dependencies and revisions
- Avoids rebuilding components that are up to date
- Used to build executable files
Dates demarcate versions, initials demarcate authors

Why do this?

- Facilitates comparison
- Facilitates “undo”
What’s Wrong with the Approach?

- Why not do this?
  - It’s a pain: always have to remember to “tag” every new file
  - It’s confusing:
    - Which log file came from `regressions_022713_mg.do`?
    - Which version of `cleandata.do` makes the data used by `regressions_022413.do`?
  - It fails the market test: No software firm does it this way
Version Control

- Software that sits “on top” of your filesystem
  - Keeps track of multiple versions of the same file
  - Records date, authorship
  - Manages conflicts

- Benefits
  - Single authoritative version of the directory
  - Edit without fear: an undo command for everything
Documents library

- chips
- cleandata
- regressions
- regressions
- rundirectory
- tvdata
Life After Version Control
### Life After Version Control

![Screenshot of TortoiseSVN Log Messages](image)

#### Revision Log

<table>
<thead>
<tr>
<th>Revision</th>
<th>Actions</th>
<th>Author</th>
<th>Date</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20647</td>
<td></td>
<td>mwong4</td>
<td>Monday, July 01, 2013 1:54:40 PM</td>
<td>[CODA-15] Add example directories</td>
</tr>
</tbody>
</table>

**Path:**

```
/trunk/slides/Code and Data/examples/Project Directory/cleandata.do
```

**Action:** Modified

Showing 2 revision(s), from revision 20647 to revision 20657 - 1 revision(s) selected, showing 1 changed paths

- Show only affected paths
- Stop on copy/rename
- Include merged revisions

[Statistics] [Help] [Show All] [Next 100] [Refresh] [OK]
Life After Version Control
Life After Version Control
Life After Version Control
Aside: If you always run rundirectory.bat before you commit, you guarantee replicability.
Directories
One Directory Does Everything

Pros: Self-contained, simple

Cons:
  
  - Have to rerun everything for every change
  - Hard to figure out dependencies
Functional Directories

Documents library

- build
  - code
    - cleandata
    - mergefiles
    - rundirectory
  - input
    - extract0B
  - output
    - tvdata
    - chips
    - tv
- analysis
  - code
    - figures
    - regressions
    - regressions_alt
    - rundirectory
    - getinput
  - input
    - tvdata
  - output
    - fig1
    - fig2
    - tables
  - temp
    - regressions
    - regressions_alt
Dependencies Obvious
One Resource, Many Projects
Keys
<table>
<thead>
<tr>
<th>county</th>
<th>state</th>
<th>cnty_pop</th>
<th>state_pop</th>
<th>region</th>
</tr>
</thead>
<tbody>
<tr>
<td>36037</td>
<td>NY</td>
<td>3817735</td>
<td>43320903</td>
<td>1</td>
</tr>
<tr>
<td>36038</td>
<td>NY</td>
<td>422999</td>
<td>43320903</td>
<td>1</td>
</tr>
<tr>
<td>36039</td>
<td>NY</td>
<td>324920</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>36040</td>
<td>NY</td>
<td>143432</td>
<td>43320903</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>NY</td>
<td>.</td>
<td>43320903</td>
<td>1</td>
</tr>
<tr>
<td>37001</td>
<td>VA</td>
<td>3228290</td>
<td>7173000</td>
<td>3</td>
</tr>
<tr>
<td>37002</td>
<td>VA</td>
<td>449499</td>
<td>7173000</td>
<td>3</td>
</tr>
<tr>
<td>37003</td>
<td>VA</td>
<td>383888</td>
<td>7173000</td>
<td>4</td>
</tr>
<tr>
<td>37004</td>
<td>VA</td>
<td>483829</td>
<td>7173000</td>
<td>3</td>
</tr>
</tbody>
</table>
### Causes for Concern

<table>
<thead>
<tr>
<th>county</th>
<th>state</th>
<th>cnty_pop</th>
<th>state_pop</th>
<th>region</th>
</tr>
</thead>
<tbody>
<tr>
<td>36037</td>
<td>NY</td>
<td>3817735</td>
<td>43320903</td>
<td>1</td>
</tr>
<tr>
<td>36038</td>
<td>NY</td>
<td>422999</td>
<td>43320903</td>
<td>1</td>
</tr>
<tr>
<td>36039</td>
<td>NY</td>
<td>324920</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>36040</td>
<td>NY</td>
<td>143432</td>
<td>43320903</td>
<td>1</td>
</tr>
<tr>
<td>37001</td>
<td>VA</td>
<td>3228290</td>
<td>7173000</td>
<td>3</td>
</tr>
<tr>
<td>37002</td>
<td>VA</td>
<td>449499</td>
<td>7173000</td>
<td>3</td>
</tr>
<tr>
<td>37003</td>
<td>VA</td>
<td>383888</td>
<td>7173000</td>
<td>4</td>
</tr>
<tr>
<td>37004</td>
<td>VA</td>
<td>483829</td>
<td>7173000</td>
<td>3</td>
</tr>
</tbody>
</table>
Relational Databases

<table>
<thead>
<tr>
<th>county</th>
<th>state</th>
<th>population</th>
</tr>
</thead>
<tbody>
<tr>
<td>36037</td>
<td>NY</td>
<td>3817735</td>
</tr>
<tr>
<td>36038</td>
<td>NY</td>
<td>422999</td>
</tr>
<tr>
<td>36039</td>
<td>NY</td>
<td>324920</td>
</tr>
<tr>
<td>36040</td>
<td>NY</td>
<td>143432</td>
</tr>
<tr>
<td>37001</td>
<td>VA</td>
<td>3228290</td>
</tr>
<tr>
<td>37002</td>
<td>VA</td>
<td>449499</td>
</tr>
<tr>
<td>37003</td>
<td>VA</td>
<td>383888</td>
</tr>
<tr>
<td>37004</td>
<td>VA</td>
<td>483829</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>state</th>
<th>population</th>
<th>region</th>
</tr>
</thead>
<tbody>
<tr>
<td>NY</td>
<td>43320903</td>
<td>1</td>
</tr>
<tr>
<td>VA</td>
<td>7173000</td>
<td>3</td>
</tr>
</tbody>
</table>

- Each *variable* is an attribute of an *element* of the table
- Each table has a *key*
- Tables are connected by *foreign keys* (state field in the county table)
Steps

- Store data in normalized format as above
  - Can use flat files, doesn’t have to be fancy relational database software
- Construct a second set of files with key transformations
  - e.g., log population
- Merge data together and run analysis
What to do with enormous databases?
Abstraction
Rampant Duplication

```
use "C:/demo/data.dta", clear
regress probgen coll pharm hlthworker hlthmajor scimajor i.market
regress probgen coll pharm hlthworker hlthmajor scimajor i.year
regress probgen coll pharm hlthworker hlthmajor scimajor i.market i.year
regress probgen coll pharm hlthworker hlthmajor scimajor i.market i.year i.age
regress probgen coll pharm hlthworker hlthmajor scimajor i.market income
regress probgen coll pharm hlthworker hlthmajor scimajor i.market i.product
regress probgen coll pharm hlthworker hlthmajor scimajor i.year age income
regress probgen coll pharm hlthworker hlthmajor scimajor i.market age income
regress probgen coll pharm hlthworker hlthmajor scimajor i.market i.product i.year age income
regress probgen coll pharm hlthworker hlthmajor scimajor i.product i.year age income
regress probgen coll pharm hlthworker hlthmajor scimajor i.market i.year income
```
use "C:/demo/data.dta", clear

local dep_var "probgen"
local info_proxies "coll pharm hlthworker hlthmajor scimajor"

regress `dep_var' `info_proxies' i.market
regress `dep_var' `info_proxies' i.year
regress `dep_var' `info_proxies' i.market i.year
regress `dep_var' `info_proxies' i.market i.product
regress `dep_var' `info_proxies' i.year age income
regress `dep_var' `info_proxies' i.market age income
regress `dep_var' `info_proxies' i.market i.product i.year age income
regress `dep_var' `info_proxies' i.product i.year age income
regress `dep_var' `info_proxies' i.market i.year income
Three Leave-Out Means

* Per capita consumption within state
  egen total_pc_potato = total(pc_potato), by(state)
  egen total_obs = count(pc_potato), by(state)
  gen leaveout_state_pc_potato = (total_pc_potato - pc_potato)/(total_obs - 1)

* Per capita consumption within metro area
  egen total_pc_potato = total(pc_potato), by(metroarea)
  egen total_obs = count(pc_potato), by(state)
  gen leaveoutMetro_pc_potato = (total_pc_potato - pc_potato)/(total_obs - 1)

* Per household consumption within metro area
  egen total_hh_potato = total(hh_potato), by(metroarea)
  egen total_obs = count(hh_potato), by(state)
  gen leaveoutMetro_hh_potato = (total_hh_potato - pc_potato)
Copy and Paste Errors

```stata
* Per capita consumption within state
egen total_pc_potato = total(pc_potato), by(state)
egen total_obs = count(pc_potato), by(state)
gen leaveout_state_pc_potato = (total_pc_potato - pc_potato)/(total_obs - 1)

* Per capita consumption within metro area
gen leaveout_metro_pc_potato = (total_pc_potato - pc_potato)/(total_obs - 1)

* Per household consumption within metro area
gen leaveout_metro_hh_potato = (total_hh_potato - pc_potato)
```

program leaveout_mean
    syntax, invar(varname) outvar(name) byvar(varname)
    tempvar tot_invar count_invar
    egen `tot_invar' = total(`invar'), by(`byvar')
    egen `count_invar' = count(`invar'), by(`byvar')
    gen `outvar' = (`tot_invar' - `invar') / (`count_invar' - 1)
end

leaveout_mean, invar(pc_potato) outvar(leaveout_state_pc_potato) byvar(state)
leaveout_mean, invar(pc_potato) outvar(leaveoutMetro_pc_potato) byvar(metro)
leaveout_mean, invar(hh_potato) outvar(leaveoutMetro_hh_potato) byvar(metro)
Documentation
Too Much Documentation
Here we run a county fixed effects regression. The dependent variable is the log of chip sales. The television variable, tv_linear, grows linearly over time, beginning in the year when television was first introduced. Standard errors are clustered by county.

<table>
<thead>
<tr>
<th>log_chip_sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>tv_linear</td>
</tr>
<tr>
<td>constant</td>
</tr>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

tv_linear  \(0.138^*\)
constant   58.83***
            (0.734)
N           6643

<table>
<thead>
<tr>
<th>log_chip_sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>tv_linear</td>
</tr>
<tr>
<td>cons</td>
</tr>
<tr>
<td>sigma_u</td>
</tr>
<tr>
<td>sigma_e</td>
</tr>
<tr>
<td>rho</td>
</tr>
</tbody>
</table>

| tv_linear   |
| cons        |
| sigma_u     |
| sigma_e     |
| rho         |

\(0.1347008\) \(0.06504\) 2.07 0.041 0.0054875 0.2639142
58.89938 7389931 79.66 0.000 57.40124 60.33752
8.640774 70.281104 0.01489061 (fraction of variance due to u_i)
/** run_regressions.do fits our county fixed effects model with and
without controls for ranch dip sales and salsa consumption
***************************************************************************/

use "C:\demo\data\chips_tv_1940.dta", clear

forval i=1941/2012 {
    append using "C:\demo\data\chips_tv_`i'.dta"
}

save "C:\demo\data\chips_tv_allyears.dta", replace
Too Much Documentation

```plaintext
run_regressions.do fits our county fixed effects model with and
without controls for ranch dip sales and sales consumption.

use "C:\demo\data\chips_tv_1940.dta", clear

forval i=1941/2012 {
    append using "C:\demo\data\chips_tv_`i'.dta"
}

save "C:\demo\data\chips_tv_allyears.dta", replace
```
local el = 0.4 / 0.2
compwlf, input('el')
Self-Documenting Code

```plaintext
local el = 0.4 / 0.2
compwlf, input(`el')

local percent_change_in_quantity = -0.4
local percent_change_in_price = 0.2
local elasticity = `percent_change_in_quantity'/`percent_change_in_price'
compute_welfare_loss, elasticity(`elasticity')
```
Management
Hey Matt,

Do you have that robustness check where we control for the amount of ranch dip sold in each county? I am writing the section on dipping sauces and wanted to mention it.

Jesse
A Friendly Chat

Sorry, I thought you were doing that because it’s similar to that other thing you were doing with controlling for salsa sales. Let me know if you want to do it or if you want me to take over.

MG
I thought Matt was doing ranch dip and Mike was doing salsa?

Jesse
I did the salsa robustness check two weeks ago. See my e-mail from 8/14, 9:36am.

Mike
Right, but in that e-mail you were controlling for the log of salsa consumption. I thought we agreed we wanted the level of consumption?

Jesse
A Friendly Chat

Shapiro, Jesse; Gentzkow, Matthew

On it!

Mike

See more about: Shapiro, Jesse.
Task Management

Salsa Robustness Check

Run main specifications adding a control for per capita salsa consumption. Add a line to our robustness table reflecting the results.

Hide earlier activity

Jesse Shapiro created task. Jun 27
Jesse Shapiro assigned to Michael Sinkinson. Jun 27

Michael Sinkinson On it!
Jun 28 at 2:00pm

Michael Sinkinson See the new version of the paper posted in /drafts/Potato Chips and the supporting code in /analysis/Potato Chips. Is this what you had in mind?
Jun 28 at 2:08pm

Jesse Shapiro Almost. Our econometric model implies that salsa consumption should enter in levels not logs. Can you revise?
Jun 28 at 2:10pm

Michael Sinkinson Ok, how about now?
Jun 28 at 2:12pm

Jesse Shapiro Yup, looks good.
Jun 28 at 2:13pm

Michael Sinkinson ✔ completed this task
Jun 28 at 2:15pm
Parting Thoughts
Code and Data

- Data are getting larger
- Research is getting more collaborative
- Need to manage code and data responsibly for collaboration and replicability
- Learn from the pros, not from us
Databases
What is a Database?

- **Database Theory**
  - Principles for how to store / organize / retrieve data efficiently (normalization, indexing, optimization, etc.)

- **Database Software**
  - Manages storage / organization / retrieval of data (SQL, Oracle, Access, etc.)
  - Economists rarely use this software because we typically store data in flat files & interact with them using statistical programs
  - When we receive extracts from large datasets (the census, Medicare claims, etc.) someone else often interacts with the database on the back end
Normalisation

“Database Normalisation is the process of organizing the fields and tables of a relational database to minimize redundancy and dependency. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them.”
Benefits of Normalization

- Efficient storage
- Efficient modification
- Guarantees coherence
- Makes logical structure of data clear
Medicare claims data for 1997-2010 are roughly 10 TB
These data are stored at NBER in thousands of zipped SAS files
Medicare claims data for 1997-2010 are roughly 10 TB

These data are stored at NBER in thousands of zipped SAS files

To extract, say, all claims for heart disease patients aged 55-65, you would need to read every line of every one of those files

THIS IS SLOW!!!
Indexing

- The obvious solution, long understood for book, libraries, economics journals, and so forth, is to build an index.
- Database software handles this automatically.
  - Allows you to specify fields that will be often used for lookups, subsetting, etc. to be indexed.
  - For the Medicare data, we could index age, gender, type of treatment, etc. to allow much faster extraction.
Indexing

- **Benefits**
  - Fast lookups
  - Easy to police data constraints

- **Costs**
  - Storage
  - Time

- **Database optimization** is the art of tuning database structure and indexing for a specific set of needs
Data Warehouses

- Traditional databases are optimized for *operational* environments
  - Bank transactions
  - Airline reservations
  - etc.
Data Warehouses

- Traditional databases are optimized for *operational* environments
  - Bank transactions
  - Airline reservations
  - etc.

- Characteristics
  - Many small reads and writes
  - Many users accessing simultaneously
  - Premium on low latency
  - Only care about current state
Data Warehouses

- In analytic / research environments, however, the requirements are different
  - Frequent large reads, infrequent writes
  - Relatively little simultaneous access
  - Value throughput relative to latency
  - May care about history as well as current state
  - Need to create and re-use many custom extracts

Database systems tuned to these requirements are commonly called data warehouses
Data Warehouses

- In analytic / research environments, however, the requirements are different
  - Frequent large reads, infrequent writes
  - Relatively little simultaneous access
  - Value throughput relative to latency
  - May care about history as well as current state
  - Need to create and re-use many custom extracts

- Database systems tuned to these requirements are commonly called “data warehouses”
Distributed Computing
Distributed Computing

- Definition: Computation shared among many independent processors

Terminology
- Distributed vs. Parallel (latter usually refers to systems with shared memory)
- Cluster vs. Grid (latter usually more decentralized & heterogeneous)
Distributed Computing

- Definition: Computation shared among many independent processors

- Terminology
  - Distributed vs. Parallel (latter usually refers to systems with shared memory)
  - Cluster vs. Grid (latter usually more decentralized & heterogeneous)
On Your Local Machine

- Your OS can run multiple processors each with multiple cores
- Your video card has hundreds of cores
- Stata, R, Matlab, etc. can all exploit these resources to do parallel computing
On Your Local Machine

- Your OS can run multiple processors each with multiple cores
- Your video card has hundreds of cores
- Stata, R, Matlab, etc. can all exploit these resources to do parallel computing

Stata

- Buy appropriate “MP” version of Stata
- Software does the rest
On Your Local Machine

- Your OS can run multiple processors each with multiple cores
- Your video card has hundreds of cores
- Stata, R, Matlab, etc. can all exploit these resources to do parallel computing

Stata
- Buy appropriate “MP” version of Stata
- Software does the rest

R / Matlab
- Install appropriate add-ins (\textit{parallel} package in R, “parallel computing toolbox” in Matlab)
- Include parallel commands in code (e.g., \textit{parfor} in place of \textit{for} in Matlab)
Resources abound

- University / department computing clusters
- Non-commercial scientific computing grids (e.g., XSEDE)
- Commercial grids (e.g., Amazon EC2)
On Cluster / Grid

- Resources abound
  - University / department computing clusters
  - Non-commercial scientific computing grids (e.g., XSEDE)
  - Commercial grids (e.g., Amazon EC2)
- Most of these run Linux w/ distribution handled by a “batch scheduler”
- Write code using your favorite application, then send it to scheduler with a bash script
MapReduce

- MapReduce is a programming model that facilitates distributed computing
  - Developed by Google around 2004, though ideas predate that
MapReduce

- MapReduce is a programming model that facilitates distributed computing
  - Developed by Google around 2004, though ideas predate that
- Most algorithms for distributed data processing can be represented in two steps
  - **Map**: Process individual “chunk” of data to generate an intermediate “summary”
  - **Reduce**: Combine “summaries” from different chunks to produce a single output file
MapReduce

- MapReduce is a programming model that facilitates distributed computing
  - Developed by Google around 2004, though ideas predate that
- Most algorithms for distributed data processing can be represented in two steps
  - **Map**: Process individual “chunk” of data to generate an intermediate “summary”
  - **Reduce**: Combine “summaries” from different chunks to produce a single output file
- If you structure your code this way, MapReduce software will handle all the details of distribution:
  - Partitioning data
  - Scheduling execution across nodes
  - Managing communication between machines
  - Handling errors / machine failures
MapReduce: Examples

- Count words in a large collection of documents
  - Map: Document $i \rightarrow$ Set of $(word, count)$ pairs $C_i$
  - Reduce: Collapse $\{C_i\}$, summing $count$ within $word$
MapReduce: Examples

- Count words in a large collection of documents
  - Map: Document $i \rightarrow$ Set of \((word, count)\) pairs $C_i$
  - Reduce: Collapse \(\{C_i\}\), summing \(count\) within \(word\)

- Extract medical claims for 65-year old males
  - Map: Record set $i \rightarrow$ Subset of $i$ that are 65-year old males $H_i$
  - Reduce: Append elements of \(\{H_i\}\)
MapReduce: Examples

- Count words in a large collection of documents
  - Map: Document $i \rightarrow$ Set of $(\text{word}, \text{count})$ pairs $C_i$
  - Reduce: Collapse $\{C_i\}$, summing $\text{count}$ within $\text{word}$

- Extract medical claims for 65-year old males
  - Map: Record set $i \rightarrow$ Subset of $i$ that are 65-year old males $H_i$
  - Reduce: Append elements of $\{H_i\}$

- Compute marginal regression for text analysis (e.g., Gentzkow & Shapiro 2010)
  - Map: Counts $x_{ij}$ of phrase $j \rightarrow$ Parameters $\left(\hat{\alpha}_j, \hat{\beta}_j\right)$ from $E(x_{ij}|y_i) = \alpha_j + \beta_j x_{ij}$
  - Reduce: Append $\left\{\hat{\alpha}_j, \hat{\beta}_j\right\}$
Inverted Index:
The map function parses each document, and emits a sequence of \(\langle\text{word}, \text{document ID}\rangle\) pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a \(\langle\text{word}, \text{list (document ID)}\rangle\) pair. The set of all output pairs forms a simple inverted index. It is easy to augment this computation to keep track of word positions.

Distributed Sort:
The map function extracts the key from each record, and emits a \(\langle\text{key}, \text{record}\rangle\) pair. The reduce function emits all pairs unchanged. This computation depends on the partitioning facilities described in Section 4.1 and the ordering properties described in Section 4.2.

3 Implementation

Many different implementations of the MapReduce interface are possible. The right choice depends on the environment. For example, one implementation may be suitable for a small shared-memory machine, another for a large NUMA multi-processor, and yet another for an even larger collection of networked machines.

This section describes an implementation targeted to the computing environment in wide use at Google: large clusters of commodity PCs connected together with switched Ethernet [4]. In our environment:

1. Machines are typically dual-processor x86 processors running Linux, with 2-4 GB of memory per machine.
2. Commodity networking hardware is used – typically either 100 megabits/second or 1 gigabit/second at the machine level, but averaging considerably less in overall bisection bandwidth.
3. A cluster consists of hundreds or thousands of machines, and therefore machine failures are common.
4. Storage is provided by inexpensive IDE disks attached directly to individual machines. A distributed file system [8] developed in-house is used to manage the data stored on these disks. The file system uses replication to provide availability and reliability on top of unreliable hardware.
5. Users submit jobs to a scheduling system. Each job consists of a set of tasks, and is mapped by the scheduler to a set of available machines within a cluster.

3.1 Execution Overview

The Map invocations are distributed across multiple machines by automatically partitioning the input data.
MapReduce: Implementation

- MapReduce is the original software developed by Google
- Hadoop is the open-source version most people use (developed by Apache)
- Amazon has a hosted implementation (Amazon EMR)
MapReduce: Implementation

- MapReduce is the original software developed by Google
- Hadoop is the open-source version most people use (developed by Apache)
- Amazon has a hosted implementation (Amazon EMR)
- How does it work?
  - Write your code as two functions called map and reduce
  - Send code & data to scheduler using bash script
Distributed File Systems

- Data transfer is the main bottleneck in distributed systems
- For big data, it makes sense to distribute data as well as computation
  - Data broken up into chunks, each of which lives on a separate node
  - File system keeps track of where the pieces are and allocates jobs so computation happens “close” to data whenever possible

Tight coupling between MapReduce software and associated systems:
- MapReduce → Google File System (GFS)
- Hadoop → Hadoop Distributed File System (HDFS)
- Amazon EMR → Amazon S3
Distributed File Systems

- Data transfer is the main bottleneck in distributed systems
- For big data, it makes sense to distribute data as well as computation
  - Data broken up into chunks, each of which lives on a separate node
  - File system keeps track of where the pieces are and allocates jobs so computation happens “close” to data whenever possible
- Tight coupling between MapReduce software and associated file systems
  - MapReduce → Google File System (GFS)
  - Hadoop → Hadoop Distributed File System (HDFS)
  - Amazon EMR → Amazon S3
Distributed File Systems

Legend:
- Data messages
- Control messages
- Application (file name, chunk index)
- chunk handle, chunk locations
- GFS master
- File namespace
  - /foo/bar
  - chunk 2ef0
- Instructions to chunkserver
- Chunkserver state
- GFS chunkserver
- Linux file system
- chunk data
- GFS chunkserver
- Linux file system
- Application (file name, chunk index)
- chunk handle, chunk locations
-(chunk handle, byte range)

Figure 1: GFS Architecture
Scenarios
Scenario 1: Not-So-Big Data

- *My data is 100 gb or less*
Scenario 1: Not-So-Big Data

- *My data is 100 gb or less*

- **Advice**
  - Store data locally in flat files (csv, Stata, R, etc.)
  - Organize data in normalized tables for robustness and clarity
  - Run code serially or (if computation is slow) in parallel
Scenario 2: Big Data, Small Analysis

- My raw data is > 100 gb, but the extracts I actually use for analysis are << 100 gb
Scenario 2: Big Data, Small Analysis

- My raw data is > 100 gb, but the extracts I actually use for analysis are << 100 gb

- Example
  - Medicare claims data → analyze heart attack spending by patient by year
  - Nielsen scanner data → analyze average price by store by month

Advice:
- Store data in relational database optimized to produce analysis extracts efficiently
- Store extracts locally in archives (csv, Stata, R, etc.)
- Organize extracts in normalized tables for robustness and clarity
- Run code serially or (if computation is slow) in parallel

Note: Gains to database increase for more structured data. For completely unstructured data, you may be better off using distributed file system + Map Reduce to create extracts.
Scenario 2: Big Data, Small Analysis

- My raw data is > 100 gb, but the extracts I actually use for analysis are << 100 gb

Example

- Medicare claims data → analyze heart attack spending by patient by year
- Nielsen scanner data → analyze average price by store by month

Advice

- Store data in relational database optimized to produce analysis extracts efficiently
- Store extracts locally in flat files (csv, Stata, R, etc.)
- Organize extracts in normalized tables for robustness and clarity
- Run code serially or (if computation is slow) in parallel
My raw data is > 100 gb, but the extracts I actually use for analysis are << 100 gb

Example

- Medicare claims data $\rightarrow$ analyze heart attack spending by patient by year
- Nielsen scanner data $\rightarrow$ analyze average price by store by month

Advice

- Store data in relational database optimized to produce analysis extracts efficiently
- Store extracts locally in flat files (csv, Stata, R, etc.)
- Organize extracts in normalized tables for robustness and clarity
- Run code serially or (if computation is slow) in parallel

Note: Gains to database increase for more structured data. For completely unstructured data, you may be better off using distributed file system + map reduce to create extracts.
Scenario 3: Big Data, Big Analysis

- My data is > 100 GB and my analysis code needs to touch all of the data

  Example: 2 TB of SECling text → run variable selection using all data

  Advice: Store data in distributed file system; Use MapReduce or other distributed algorithms for analysis
Scenario 3: Big Data, Big Analysis

- My data is > 100 GB and my analysis code needs to touch all of the data

Example

- 2 TB of SEC filing text → run variable selection using all data
Scenario 3: Big Data, Big Analysis

- *My data is > 100 GB and my analysis code needs to touch all of the data*

- Example
  - 2 TB of SEC filing text $\rightarrow$ run variable selection using all data

- Advice
  - Store data in distributed file system
  - Use MapReduce or other distributed algorithms for analysis