Project 2 – Distributed Shell

Project 2 – Distributed Shell

Due: Thursday, April 5th

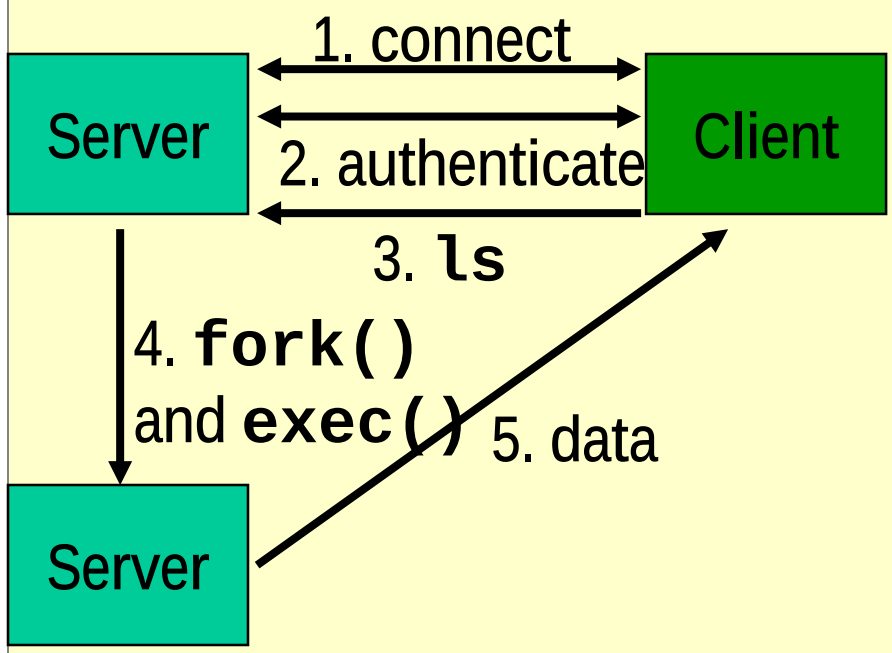
Motivation

- **Shell is core operating systems concept**
- **Executing commands on remote machine is core distributed systems concept**
  - Computing in the “cloud” (except in this case, server address is known, not a service) with results returned to user

→ ***Distributed Shell***

Distributed Shell

- **Client-Server**
  - Server on "known" port for client
- **Non-interactive**
  - Command line args
  - **`get-opt.c`**
  - Authentication built-in
- **Uses TCP sockets**
  - **`listen.c`** and **`talk.c`**
- **Security**
  - Authentication (next slide)
- **Can handle multiple requests**
  - **`fork()`**
  - Concurrent server



Simple Authentication

- **Client connects to server, sending in user-name (not password)**
  - Note, typically name used to lookup unique data (salt) stored with each name that further randomizes hash salt – here, just a check that valid
- **Server responds by sending back unique random number**
  - Use **`rand()`**
  - Changes each invocation (provide different random seed each run, using **`srand()`**)
- **Client encrypts using password and number as key**
  - Here, password can be "hard coded" into client
  - Use **`crypt()`**
- **…**

Simple Authentication

- **…**
- **Client sends hashed value to server**
- **Server encrypts with user's password and same number as key**
  - Server “knows” password (e.g., externally exchanged)
  - Here, password can be “hard coded” into server
- **Server compares two hashed/encrypted values, if same then ok**
  - Otherwise, return error message and exit
- **Note, can do authentication either before or after `fork()`**
  - Question: pluses and minuses for each?

Sample

Server

```
YourVM% ./server
./server activating.
port: 4513
dir: /home/claypool/dsh Socket created!
Accepting connections.

Connection request received.
forked child
received: john
password ok
command: ls
executing command...
```

Client

```
OtherVM% ./dsh -c "ls" -s cccwork3
Makefile
client.c
dsh
dsh.c
index.html
server
server.c
server.h
sock.c
sock.h
```

Hints

- **Build shell independent of server**
- **Get basic connection working without shell**
  - Socket help: **`listen-tcp.c`** and **`talk-tcp.c`**
  - Socket slides (next deck)
- **Shell help**
  - **`fork.c`**
  - **`execl.c`**
  - **`fork()`, `execve()`, `getopt()`, `strtok()`, `dup2()`**
- **Use a `Makefile`**
  - Web page's is simple
- **Use `man` pages for functionality, return codes**
- **Beware of zombies!**

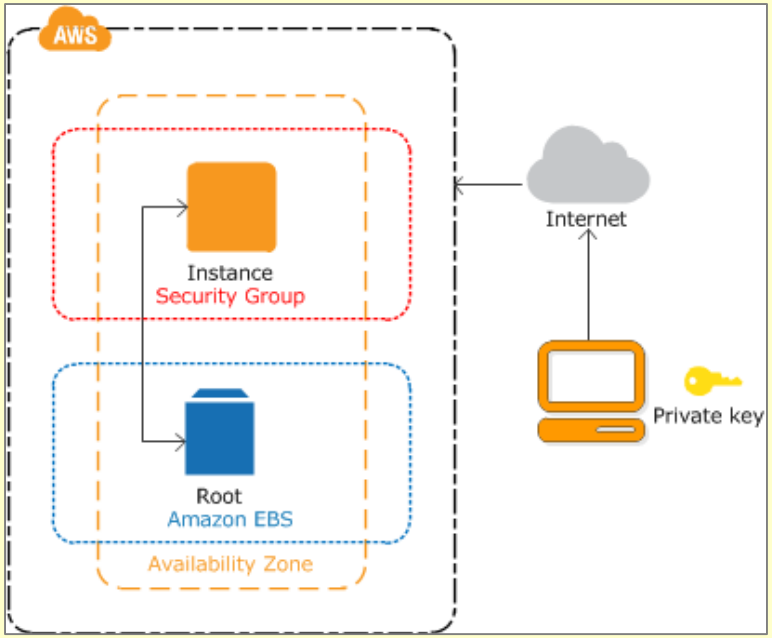
Debug locally

- **Do all of this on a pair of course virtual machines**
- **Show that you can write a command locally that will execute remotely**
  - On your teammate's VM
- **When that works …**

Cloud Computing – Amazon

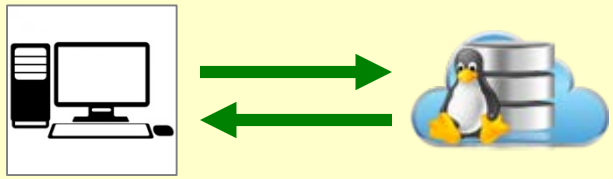
Setup Amazon Elastic Compute Cloud (EC2)

1. **Sign up (free, 1 year)**
2. **Launch Linux VM**
   - Copy security keys
3. **Connect (e.g., putty)**
4. **Clean up (when done)**



Cloud Computing – Amazon

- **Will have "cloud" Linux host to admin**
  - Note IP of instance when starting
- **Install software**
  - **gcc** (or **g++**)
  - **make**
  - **sysbench** (for performance)

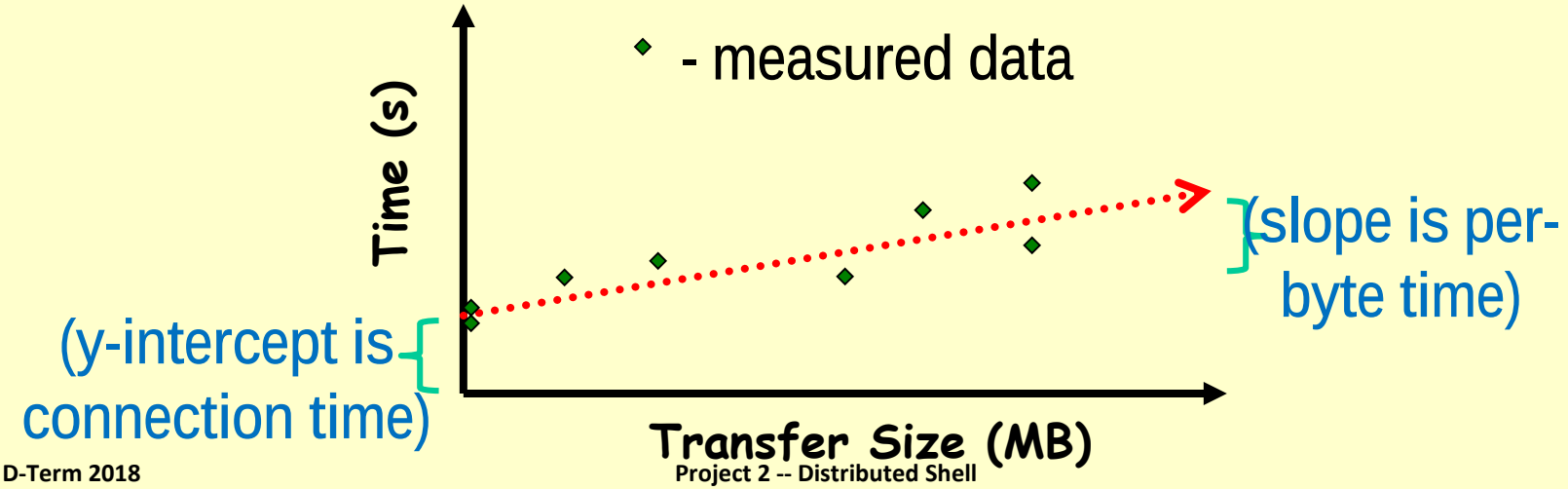


- **Typically: `sudo apt-get install {name}`**
- **Copy server source (e.g., `server.c` and `Makefile`) to cloud host**
- **Rebuild**
- **Re-test**
- **Debug (if necessary)**

(See Project Description for links to documentation)

Experiments – Network Performance

- **Latency of connection (milliseconds)**
  - Includes authentication
  - Force call to server so that server does not do `exec( )`
- **Maximum throughput (b/s)**
  - Transfer large file
    - redirect > to file else `stdout` may be bottleneck
- **Multiple runs**



Experiments – CPU and File I/O Performance

- **Sysbench for CPU**

```
sysbench --test=cpu --cpu-max-prime=20000 run
```

  - *total time: 23.8724s*

- **Sysbench for File I/O**

```
sysbench --test=fileio --file-total-size=16G prepare
sysbench --test=fileio --file-total-size=16G --file-test-mode=rndrw --init-rng=on --max-time=30 --max-requests=0 run
```

  - *Read 9.375Mb …(**53.316Kb/sec**)*

- **Clean up!**

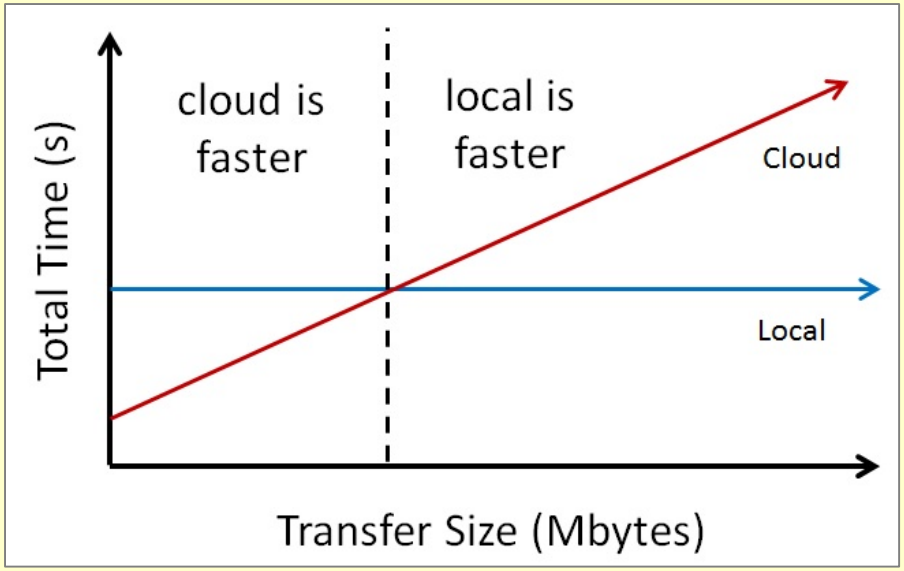
```
sysbench --test=fileio --file-total-size=16G cleanup
```

- **Local (own virtual machine) *vs* remote (cloud)**

Model

```
Local_CPU + Local_File_I/O = n * Network + Remote_CPU + Remote_File_I/O
```

- Simple, algebraic model
- Compute when (for data transfer) cloud computing gives better performance (reduced time for task)
- `Solve for n`



Writeup

- ***Design* - describe your experiments**
  - Programs/scripts (pseudo-code)
  - Number of runs
  - Data recording method
  - System conditions
    - **`lscpu`** CPU information (also **`/proc/cpuinfo`**)
    - **`lsblk`** block device information (also **`/df/mount | column -t`**)
    - **`free -h`** available memory
    - **`uname -a`** OS version (also **`/proc/version`**)
  - Other …
- ***Results* - depict your results *clearly***
  - Graph (see previous slide)
  - Table (at least *mean* and *standard deviation)*
- ***Analysis* - interpret results**
  - Describe what results mean

Hand In

- **Source code package**
  - **`server.c`, `client.c`**
  - Support files (**`*.h`**)
  - **`Makefile`**
  - **`README.txt`** detailing building and using
- **Experimental writeup**
- **Clean and `zip`**
- **Submit via Instruct Assist**

Grading

Basic shell program	(15 points)
Basic Client-Server communication program	(15 points)
Proper re-direction of standard output	(10 points)
Proper error checking of system/socket calls	(10 points)
Proper authentication	(10 points)
Proper closing of sockets in relation to fork	(5 points)
Proper handling of zombies	(5 points)
Amazon EC2 setup	(15 points)
Experiment Design	(6 points)
Experiment Results	(5 points)
Experiment Analysis	(4 points)

See rubric on Web page, too