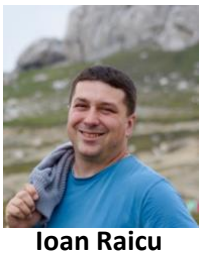
Syllabus

CS 553: Cloud Computing

http://www.cs.iit.edu/~iraicu/teaching/CS553-S18/



Ioan Raicu

Alex Orhean

Poornima Nookala

Semester: Spring 2018
Lecture Time: Monday/Wednesday, 11:25AM - 12:40PM
Location: Stuart Building 104

Professor:

- **Dr. Ioan Raicu** (iraicu@cs.iit.edu, 1-312-567-5704)
  - Office Hours: Wednesday 12:45PM-2:00PM (SB226B)

Teaching Assistants

- **Alex Orhean** (aorhean@hawk.iit.edu)
  - Office Hours: Mondays 12:45PM-1:45PM / Wednesdays 10:15AM-11:15AM (SB007)
- **Poornima Nookala (**pnookala@hawk.iit.edu**)**
  - Office Hours: Mondays 10:15AM-11:15AM / Thursday 12:45PM-1:45PM (SB007)

Course Description

Cloud Computing is "A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet." It has become a driving force for information technology over the past several years, and it is hinting at a future in which we won't compute on local computers, but on centralized facilities operated by third-party compute and storage utilities. Governments, research institutes, and industry leaders are rushing to adopt Cloud Computing to solve their ever-increasing computing and storage problems arising in the Internet Age. There are three main factors contributing to the surge and interests in Cloud Computing: 1) rapid decrease in hardware cost and increase in computing power and storage capacity, and the advent of multi-core architecture and modern supercomputers consisting of hundreds of thousands of cores; 2) the exponentially growing data size in scientific instrumentation/simulation and Internet publishing and archiving; and 3) the wide-spread adoption of Services Computing and Web 2.0 applications. This course is a tour through various topics and technologies related to Cloud Computing. Topics include distributed system models and enabling technologies, computer clusters for scalable Computing, virtual machines and virtualization of clusters and datacenters, design of cloud computing platforms, cloud programming and software environments (Workflow Systems, MapReduce, Google App Engine, Amazon AWS, Microsoft Azure, and emerging cloud software stacks), grid computing and resource management, P2P computing with overlay networks, ubiquitous computing with clouds and the Internet of things, and data-intensive distributed computing. The course involves lectures, projects, programming assignments, and exams. Prerequsites: CS450 or CS455.

Required Texts

We will be using the textbook Distributed and Cloud Computing: Clusters, Grids, Clouds, and the Future Internet by Kai Hwang, Jack Dongarra & Geoffrey C. Fox. (Required)

Prerequisites

CS450 (OS) or CS455 (Data Communications). Other courses that might contribute to having a better in depth understanding of this course are CS451, CS542, CS546, CS550, CS551, CS552, CS554, CS570, and CS595 (VMs). Many of these graduate courses are part of the Master of Computer Science Specialization in Distributed and Cloud Computing.

Detailed Course Topics

- Distributed System Models
- Parallel Computing
- Cloud Platform Architectures
- Cloud Programming
- Grid Computing
- Supercomputing Computing

Programming Assignments

There will be 3 programming assignments throughout the semester, each worth 15% of the total grade, and each taking about 3~4 weeks to complete. These assignments must be completed individually. The projects will require knowledge of Java, C and/or C++. It is expected that students know the basics of these languages, but students are free to generally choose the languages they want to implement their assignments in. These assignments must all work in a Linux environment (in which they will be graded in).

Computer Usage

Computer systems that will be used for development of projects (more information about access to these will be passed in the first several lectures):

- **Amazon AWS (**https://aws.amazon.com**)**
- **Chameleon** (https://www.chameleoncloud.org)
- **IIT/CS Teaching Cluster**

Project

There will be 1 project which will be worth 10% of the grade. This assignment must be completed individually, and will require about 4 weeks to be completed. It will involve students to read online material, compute and generate graphs, and write a report.

Late Policy

Assignments will be due at 11:59PM on the day of the due date. There will be a 15-minute grace period. There will also be a 4-day late pass, where students can submit late assignments without penalty; the late pass can be used in 1-day increments spread out over multiple assignments. Any late submissions beyond the grace period and beyond the 4-day late pass, will be penalized 20% every day it is late.

Plagiarism Policy

Cheating will not be tolerated. We will use the MOSS: Measure Of Software Similarity system from Stanford (https://theory.stanford.edu/~aiken/moss/). It is used to automatically determine the similarity of programs (even if they are written in different programming languages). The supported languages are: C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, a8086 assembly, MIPS assembly, HCL2.

You will receive a 0 on the assignment; extremely serious offences will fail the course and be reported to the university.

Some example screen shots from the MOSS system:

Moss Results

Tue Sep 8 23:29:31 PDT 2015

Options -l python -d -m 10

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (99%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (99%)	86
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (76%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (66%)	91
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (81%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (82%)	69
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (70%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]3/ (61%)	70
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (69%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]2/ (40%)	71
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (56%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]4/ (50%)	43
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (62%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]9/ (55%)	67
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (55%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (48%)	40
/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]/ (54%)	/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/6/raw/[redacted]9/ (55%)	40

/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted] (68%)		/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted] (73%)	
4-71		2-66	
95-111		90-106	
74-91		69-86	
115-132		110-127	

/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]

>>>> file: LongJump.py

```
print("***** Long Jump Information System *****")
print("Please enter the names of competitors. (Press return when done.)")
print("Competitor no. 1:")
competitor = input()
b,c,g,h,d,k = 1,0,0,0,[],0
maxi,competitors = [],[competitor]
while True:
    b += 1
    print("Competitor no. "+str(b)+":")
    competitor = input()
    if competitor == "":break
    else:
        competitors.append(competitor)
print("Please enter the distances for each competitor.")
for each in competitors:
    at1 = input("Attempt 1:\n")
    at2 = input("Attempt 2:\n")
    at3 = input("Attempt 3:\n")
    x = (at1+at2+at3).lower()
    if (at1+at2+at3).find("oul") != -1:
    d.append(at1)
    d.append(at2)
    d.append(at3)
    maxi.append(max(eval(at1),eval(at2),eval(at3)))
```

/home/ubuntu/Projects/work/2015/uct-csc1010h/tutorials/4/raw/[redacted]

>>>> file: LongJump.py

```
print("***** Long Jump Information System *****")
print("Please enter the names of competitors. (Press return when done.)")
print("Competitor no. 1:")
competitor = input()
b,c,g,h,d,k = 1,0,0,0,[],0
maximums,competitors = [],[competitor]
while True:
    b += 1
    print("Competitor no. "+str(b)+":")
    competitor = input()
    if competitor == "":break
    else:
        competitors.append(competitor)
print("Please enter the distances for each competitor.")
for each in competitors:
    attempt1 = input("Attempt 1:\n")
    attempt2 = input("Attempt 2:\n")
    attempt3 = input("Attempt 3:\n")
    g = (attempt1+attempt2+attempt3).lower()
    if (attempt1+attempt2+attempt3).find("oul") != -1:
    d.append(attempt1)
    d.append(attempt2)
    d.append(attempt3)
    maximums.append(max(eval(attempt1),eval(attempt2),eval(attempt3)))
    d.remove("foul")
```

Exams

There will be one exam worth 45% of the overall grade. If you are an in-class student, you must take the exams in class; for remote students, you can either come take it in class, or you can take the exam at an official testing center in a proctored environment. The exams will be individual. Students **WILL NOT BE ALLOWED** to use any material on the exam, such as textbooks, slides, notes, electronic devices (e.g. phones, eReaders, tablets, or laptops). The exam date, time, and location will be announced once the university has scheduled the final exams during the official final exam week between April 30th and May 4th 2018. **THERE WILL BE NO MAKEUP EXAMS.**

Grades

Grading Policies:

- **Programming Assignments (3):** 45% -- can use late day passes (PA1=15%, PA2=15%, PA3=15%)
- **Project (1):** 10% -- can use late day passes
- **Exam (1):** 45% -- NO MAKEUPS

The following grading scale will be used. The scale will be adjusted downwards based on the overall performance of the entire class. Traditionally, in my classes, the class average score typically falls around 80% (a solid B-grade). There are two separate (but similar) grading scales, one for undergraduate students, and one for graduate students.

Undergraduate Students:

- **A: 85% ~ 100%**
- **B: 70% ~ 84%**
- **C: 60% ~ 69%**
- **D: 50%~59%**
- **E: 0% ~ 49%**

Graduate Students:

- **A: 85% ~ 100%**
- **B: 70% ~ 84%**
- **C: 50% ~ 69%**
- **E: 0% ~ 49%**

Discussion Forum

This course will use Piazza to facilitate discussions for assignments, at http://piazza.com/iit/spring2018/cs553/home (it has not been setup yet, more instructions will follow). Piazza should be the primary mechanism of communication between the students and the professor and the TAs. Direct email communication can be used, but make sure to clearly mark your emails in the subject with "CS553".

CS553 Programming Assignment #1

Benchmarking

Instructions:

- *Assigned date: Monday February 19th, 2018*
- *Due date: 11:59PM on Friday March 9th, 2018*
- *Maximum Points: 100%*
- *This programming assignment must be done individually*
- *Please post your questions to the Piazza forum*
- *Only a softcopy submission is required; it must be submitted through GIT*
- *Late submission will be penalized at 20% per day (beyond the 4-day late pass).*

1 Your Assignment

This project aims to teach you how to benchmark different parts of a computer system, from the CPU, memory, disk, and network. You can be creative with this project. You are free to use any of the following programming languages (C, C++, Java, Python) and abstractions (PThreads, Sockets) that might be needed. Other programming languages will not be allowed due to the increased complexity in grading; do not write code that relies on complex libraries (e.g. boost), as those will simplify certain parts of your assignments, and will increase complexity in grading.

You can use any Linux system for your development, but you must use the virtual cluster found at 129.114.33.105 for the formal testing and evaluation. This virtual cluster is running on the Chameleon testbed [https://www.chameleoncloud.org]; more information about the hardware in this testbed can be found at https://www.chameleoncloud.org/about/hardware-description/, under Standard Cloud Units. Even more details can be found at https://www.chameleoncloud.org/user/discovery/, choose "Compute", then Click the "View" button.

You have been created accounts on this virtual cluster for this assignment; if you have not received your accounts information, reach out to the TAs for this information. The computer system found at 129.114.33.105 is the cluster login node. You can connect to this login node with ssh. Do not use the login nodes for anything other than ssh from your system to the cluster, and other light commands (e.g. file system operations, compilation, code editing, etc). Once on the login node, you must use the Slurm job management system to submit jobs to run your benchmarks. You must use GIT for source control throughout your assignment; we will use GIT to also collect assignment submissions.

The cluster has a NFS setup and configured at /exports/home/userid with a 20GB quota per user. This directory can be used to store data that you want to access across the virtual cluster. There are two additional storage resources that are considered node local storage: SSD (/tmp with 20GB quota, it is cleaned up after every job) and RAMDisk (/dev/shm with variable limited space, it is cleaned up after every job). Please pay attention to where you must run your benchmarks from (e.g. NFS, SSD, or RAMDisk).

In this project, you need to design a benchmarking program that covers the four system components: processor, memory, disk, and network. You will perform strong scaling studies, unless otherwise noted; this means you will set the amount of work (e.g. the number of instructions or the amount of data to evaluate in your benchmark), and reduce the amount of work per thread as you increase the number of threads. The TAs will compile (with the help of make, ant, or maven) and test your code on this virtual

cluster. If your code does not compile and the TAs cannot run your project, you will get 0 for the assignment.

1. **Processor:**
    a. Implement: MyCPUBench benchmark
    b. Workload: 1 trillion arithmetic (quarter precision, half precision, single precision, double precision) operations
        i. QP: quarter precision operations compute on 1-byte char data types
        ii. HP: half precision operations compute on 2-byte short data types
        iii. SP: single precision operations compute on 4-byte integers data types
        iv. DP: double precision operations compute on 8-byte double data types
    c. Concurrency: 1 thread, 2 threads, 4 threads
    d. Measure: processor speed, in terms of operations per second; report data in GigaOPS, giga operations (10^9) per second
    e. Run the HPL benchmark from the Linpack suite (http://en.wikipedia.org/wiki/LINPACK) and report the best performance achieved using double precision floating point; make sure to run Linpack across all cores and to use a problem size that is large enough to consume ¾ of the available memory of your testing node (for this benchmark, you do not have to compute 1 trillion arithmetic computations, but the problem size you set to fill ¾ of the available memory will dictate the workload size). You can download the Linpack benchmark suite from https://software.intel.com/en-us/articles/intel-mkl-benchmarks-suite (binaries of the HPL benchmarks are available at this link, there is no need to compile the benchmark); run the HPL without MPI benchmark from the suite. Make sure to tune the HPL benchmark in HPL.dat (see http://www.netlib.org/benchmark/hpl/tuning.html for more information on how to interpret the HPL.dat configuration)
    f. Compute the theoretical peak performance of your processor. What efficiency do you achieve compared to the theoretical performance? Compare and contrast your performance to that achieved by your benchmark, HPL, and the theoretical peak performance.
    g. Fill in the table 1 below for Processor Performance:

Workload	Concurrency	MyCPUBench Measured Ops/Sec (GigaOPS)	HPL Measured Ops/Sec (GigaOPS)	Theoretical Ops/Sec (GigaOPS)	MyCPUBench Efficiency (%)	HPL Efficiency (%)
QP	1		N/A			N/A
QP	2		N/A			N/A
QP	4		N/A			N/A
HP	1		N/A			N/A
HP	2		N/A			N/A
HP	4		N/A			N/A
SP	1		N/A			N/A
SP	2		N/A			N/A
SP	4		N/A			N/A
DP	1					
DP	2					
DP	4					

2. **Memory:**
    a. Implement: MyRAMBench benchmark; ***hint:*** *you are unlikely going to be able to do this benchmark in Java or other high level languages, while C/C++ is a natural language to implement this benchmark; make sure to measure the performance of your memory and not your processor caches*
    b. Workload: 1GB data; operate over it 100X times with various access patterns (RWS, RWR) and various block sizes (1KB, 1MB, 10MB)
        i. RWS: read+write (e.g. memcpy) with sequential access pattern
        ii. RWR: read+write (e.g. memcpy) with random access pattern
    c. Concurrency: 1 thread, 2 threads, 4 threads
    d. Measure:
        i. throughput, in terms of bytes per second; report data in GB/sec, gigabytes (10^9) per second; these experiments should be conducted over 100GB of data
        ii. latency (read+write 1 byte of data), in terms of time per access; report data in us, microseconds; limit experiment to 100 million operations
    e. Run the Memory benchmark pmbw (https://panthema.net/2013/pmbw/) with 1, 2, and 4 threads, and measure the memory sub-system performance
    f. Compute the theoretical bandwidth and latency of your memory. What efficiency do you achieve compared to the theoretical performance? Compare and contrast your performance to that achieved by pmbw, and to the theoretical performance.
    g. Fill in the table 2 below for Memory Throughput:

Work-load	Con-currency	Block Size	MyRAMBench Measured Throughput (GB/sec)	pmbw Measured Throughput (GB/sec)	Theoretical Throughput (GB/sec)	MyRAMBench Efficiency (%)	pmbw Efficiency (%)
RWS	1	1KB					
RWS	1	1MB					
RWS	1	10MB					
RWS	2	1KB					
RWS	2	1MB					
RWS	2	10MB					
RWS	4	1KB					
RWS	4	1MB					
RWS	4	10MB					
RWR	1	1KB					
RWR	1	1MB					
RWR	1	10MB					
RWR	2	1KB					
RWR	2	1MB					
RWR	2	10MB					
RWR	4	1KB					
RWR	4	1MB					
RWR	4	10MB					

    h. Fill in the table 3 below for Memory Latency:

Work-load	Con-currency	Block Size	MyRAMBench Measured Latency (us)	pmbw Measured Latency (us)	Theoretical Latency (us)	MyRAMBench Efficiency (%)	pmbw Efficiency (%)
RWS	1	1B					
RWS	2	1B					
RWS	4	1B					
RWR	1	1B					
RWR	2	1B					
RWR	4	1B					

3. **Disk:**
    a. Implement: MyDiskBench benchmark; ***Hint:*** *there are multiple ways to read and write to disk, explore the different APIs, and pick the fastest one out of all them; also make sure you are measuring the speed of your disk and not your memory (you may need to flush your disk cache managed by the OS)*
    b. Workload: 10GB data; operate over it 1X times with various access patterns (RWS, RWR) and various block sizes (1MB, 10MB, 100MB)
        i. RS: read with sequential access pattern
        ii. WS: write with sequential access pattern
        iii. RR: read with random access pattern
        iv. WS: write with random access pattern
    c. Concurrency: 1 thread, 2 threads, 4 threads
    d. Measure:
        i. throughput, in terms of bytes per second; report data in MB/sec, megabytes (10^6) per second; these experiments should be conducted over 10GB of data
        ii. latency (read or write 1KB of data), in terms of time per access; report data in ms, milliseconds and in IOPS (I/O operations per second); limit experiment to 1 million operations (1GB data)
    e. Run the Disk benchmark IOZone benchmark (http://www.iozone.org/) with 1, 2, and 4 threads, and measure the disk performance
    f. Compute the theoretical bandwidth and latency of your disk. What efficiency do you achieve compared to the theoretical performance? Compare and contrast your performance to that achieved by IOZone, and to the theoretical performance.
    g. Fill in the table 4 below for Disk Throughput:

Work-load	Con-currency	Block Size	MyDiskBench Measured Throughput (MB/sec)	IOZone Measured Throughput (MB/sec)	Theoretical Throughput (MB/sec)	MyDiskBench Efficiency (%)	IOZone Efficiency (%)
RS	1	1MB					
RS	1	10MB					
RS	1	100MB					
RS	2	1MB					
RS	2	10MB					
RS	2	100MB					
RS	4	1MB					

RS	4	10MB					
RS	4	100MB					
WS	1	1MB					
WS	1	10MB					
WS	1	100MB					
WS	2	1MB					
WS	2	10MB					
WS	2	100MB					
WS	4	1MB					
WS	4	10MB					
WS	4	100MB					
RR	1	1MB					
RR	1	10MB					
RR	1	100MB					
RR	2	1MB					
RR	2	10MB					
RR	2	100MB					
RR	4	1MB					
RR	4	10MB					
RR	4	100MB					
WR	1	1MB					
WR	1	10MB					
WR	1	100MB					
WR	2	1MB					
WR	2	10MB					
WR	2	100MB					
WR	4	1MB					
WR	4	10MB					
WR	4	100MB					

h. Fill in the table 5 below for Disk Latency (measured in ms):

Work-load	**Con-currency**	**Block Size**	**MyDiskBench Measured Latency (ms)**	**IOZone Measured Latency (ms)**	**Theoretical Latency (ms)**	**MyDiskBench Efficiency (%)**	**IOZone Efficiency (%)**
RR	1	1KB					
RR	2	1KB					
RR	4	1KB					
RR	8	1KB					
RR	16	1KB					
RR	32	1KB					
RR	64	1KB					
RR	128	1KB					
WR	1	1KB					
WR	2	1KB					
WR	4	1KB					

WR	8	1KB					
WR	16	1KB					
WR	32	1KB					
WR	64	1KB					
WR	128	1KB					

i. Fill in the table 6 below for Disk Latency (measured in IOPS); this data can be collected at the same time as the data collected from table 5:

Work-load	**Con-currency**	**Block Size**	**MyDiskBench Measured IOPS**	**IOZone Measured IOPS**	**Theoretical IOPS**	**MyDiskBench Efficiency (%)**	**IOZone Efficiency (%)**
RR	1	1KB					
RR	2	1KB					
RR	4	1KB					
RR	8	1KB					
RR	16	1KB					
RR	32	1KB					
RR	64	1KB					
RR	128	1KB					
WR	1	1KB					
WR	2	1KB					
WR	4	1KB					
WR	8	1KB					
WR	16	1KB					
WR	32	1KB					
WR	64	1KB					
WR	128	1KB					

4. **Network:**
   a. Implement: MyNETBench benchmark (must support both client and server functionality in the same program; multi-threaded support must exist at both client and server; ***hint:*** *you are going to need two nodes for these experiments, one for the server (receiver) and one for the client (sender)*
   b. Workload: 1GB data; operate over it 100X times with various block sizes (1KB, 32KB)
   c. Protocols: evaluate both the TCP and UDP protocol
   d. Concurrency: 1 thread, 2 threads, 4 threads, 8 threads
   e. Measure:
      i. throughput, in terms of bytes per second; report data in Mb/sec, megabits (10^6) per second; these experiments should be conducted over 100GB of data to send from a client and be received at a server; the data should be kept in memory in a 1GB piece of memory on the client, and it should be stored in a 1GB piece of memory on the server that it overwrites as more data is received
      ii. latency (ping-pong 1 byte of data), in terms of time per RTT (round-trip-time); report data in ms, milliseconds; limit experiment to 1 million operations

f. Run the Network benchmark iperf (http://en.wikipedia.org/wiki/Iperf) with 1, 2, 4, and 8 threads, and measure the network throughput performance between two nodes; network latency can be measured with the ping utility

g. Compute the theoretical bandwidth and latency of your network. What efficiency do you achieve compared to the theoretical performance? Compare and contrast your performance to that achieved by iperf, and to the theoretical performance.

h. Fill in the table 7 below for Network Throughput:

Proto-col	Con-currency	Block Size	MyNETBench Measured Throughput (Mb/sec)	iperf Measured Throughput (Mb/sec)	Theoretical Throughput (Mb/sec)	MyNETBench Efficiency (%)	iperf Efficiency (%)
TCP	1	1KB		564 Mbits/sec			
TCP	1	32KB		3626 Mbits/sec			
TCP	2	1KB		566 Mbits/sec			
TCP	2	32KB		3666 Mbits/sec			
TCP	4	1KB		562 Mbits/sec			
TCP	4	32KB		3,524.66 Mbits/sec			
TCP	8	1KB		563 Mbits/sec			
TCP	8	32KB		3585 Mbits/sec			
UDP	1	1KB		559 Mbits/sec			
UDP	1	32KB		1693 Mbits/sec			
UDP	2	1KB		892 Mbits/sec			
UDP	2	32KB		1761 Mbits/sec			
UDP	4	1KB		987 Mbits/sec			
UDP	4	32KB		1198 Mbits/sec			
UDP	8	1KB		959 Mbits/sec			
UDP	8	32KB		1399 Mbits/sec			

i. Fill in the table 8 below for Network Latency:

Proto-col	Con-currency	Message Size	MyNETBench Measured Latency (ms)	ping Measured Latency (ms)	Theoretical Latency (ms)	MyNETBench Efficiency (%)	iperf Efficiency (%)
TCP	1	1B					
TCP	2	1B					
TCP	4	1B					
TCP	8	1B					
UDP	1	1B					
UDP	2	1B					
UDP	4	1B					
UDP	8	1B					

Other requirements:

- You must write all benchmarks from scratch. You can use well known benchmarking software to verify your results, but you must implement your own benchmarks. Do not use code you find online, as you will get 0 credit for this assignment. If you have taken other courses where you

wrote similar benchmarks, you are welcome to start with your codebase as long as you wrote the code in your prior class.

- All of the benchmarks will have to evaluate concurrency performance; concurrency can be achieved using threads. Use strong scaling in all experiments, unless it is not possible, in which case you need to explain why a strong scaling experiment was not done. Be aware of the thread synchronizing issues to avoid inconsistency or deadlock in your system.
- Most benchmarks could be run on a single machine, but some benchmarks (e.g. network) will require 2 machines.
- Not all timing functions have the same accuracy; you must find one that has at least 1ms accuracy or better, assuming you are running the benchmarks for at least seconds at a time.
- Since there are many experiments to run, find ways (e.g. scripts) to automate the performance evaluation.
- For the best reliability in your results, repeat each experiment 3 times and report the average and standard deviation. This will help you get more stable results that are easier to understand and justify.
- No GUIs are required. Simple command line interfaces are expected.

2 Skeleton Code

- Skeleton code written in C for this assignment can be downloaded from a public git repo using this command.
  `git clone ssh://<userid>@129.114.33.105/exports/git/public/cs553-pa1.git`
- If you prefer to use JAVA or Python, please make sure to follow the same naming conventions and input/output file formats as in the C code.
- Skeleton code has 4 folders – CPU, Disk, Memory and Network. Each folder has a C file with blank main function, a Makefile for compiling the code, input files and a sample output file format.
- ReadMe file in each folder has more details on the input format.
- Output should be space separated columns of values.
- Your main function should read input from the files provided and generate output in the format provided.

3 Where you will submit

You will have to submit your solution to a private git repository created for you. You will have to firstly clone the repository. Then you will have to add or update your source code, documentation and report. Your solution will be collected automatically after the deadline grace period. If you want to submit your homework later, you will have to push your final version of the solution and you will have let the TAs know of it through email cs553-s18@datasys.cs.iit.edu. There is no need to submit anything on BB for this assignment. Here are some examples on how to clone your private repository and how to add files to it (replace **userid** with your username):

```
git clone ssh://userid@129.114.33.105/exports/git/userid/cs553-pa1.git
cd cs553-pa1/
touch readme.txt
cat "Username A20*" > readme.txt
```

```
git add readme.txt
git commit -m "Added the readme file"
git push
```

If you cannot access your repository contact the TAs. You can find a git cheat sheet here: https://www.git-tower.com/blog/git-cheat-sheet/

~~4~~3 What you will submit

When you have finished implementing the complete assignment as described above, you should submit your solution to your private git repository. Each program must work correctly and be detailed in-line documented. You should hand in:

1. **Source code (30%):** All of the source code; in order to get full credit for the source code, your code must have in-line documents, must compile, and must be able to run the sample benchmarks from #2 above. You must have a makefile for easy compilation.
2. **Readme (10%):** A detailed manual describing how the program works. The manual should be able to instruct users other than the developer to run the program step by step. The manual should contain example commands to invoke each of the five benchmarks. This should be included as readme.txt in the source code folder.
3. **Report / Performance (60%):** Must have working code that compiles and runs on the specified cluster to receive credit for report/performance; furthermore, the code must match the performance presented in the report. A separate (typed) design document (named pa1-report.pdf) of approximately 1-3 pages describing the overall program design, and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made). Since this is an assignment aimed at teaching you about benchmarking, this is one of the most important part; you must evaluate the four benchmarks with the entire parameters space mentioned in Section 1, and put as a sub-section to your design document mentioned in (1) above. You must produce 8 tables to showcase the results; we encourage you to create additional figures to highlight your performance evaluation. Please combine data and plot on the same graph wherever possible, for either compactness reasons, or comparison reasons. Don't forget to plot the average and standard deviation (if you do multiple runs of an experiment). Also, you need to explain each graph's results in words. Hint: graphs with no axis labels, legends, well defined units, and lines that all look the same, are likely very hard to read and understand graphs. You will be penalized if your graphs are not clear to understand. Please specify which student contributed to what benchmark experiments.

Submit code through github.

Grades for late programs will be lowered 20% per day late beyond the 4-day late pass.

Introduction/Tutorial on the Linux Ecosystem

Bash scripting, SSH, POSIX Threads, BSD Sockets (part 1)

Alexandru Iulian Orhean
aorhean@hawk.iit.edu

Illinois Institute of Technology
Computer Science Department
Data-Intensive Distributed Systems Laboratory

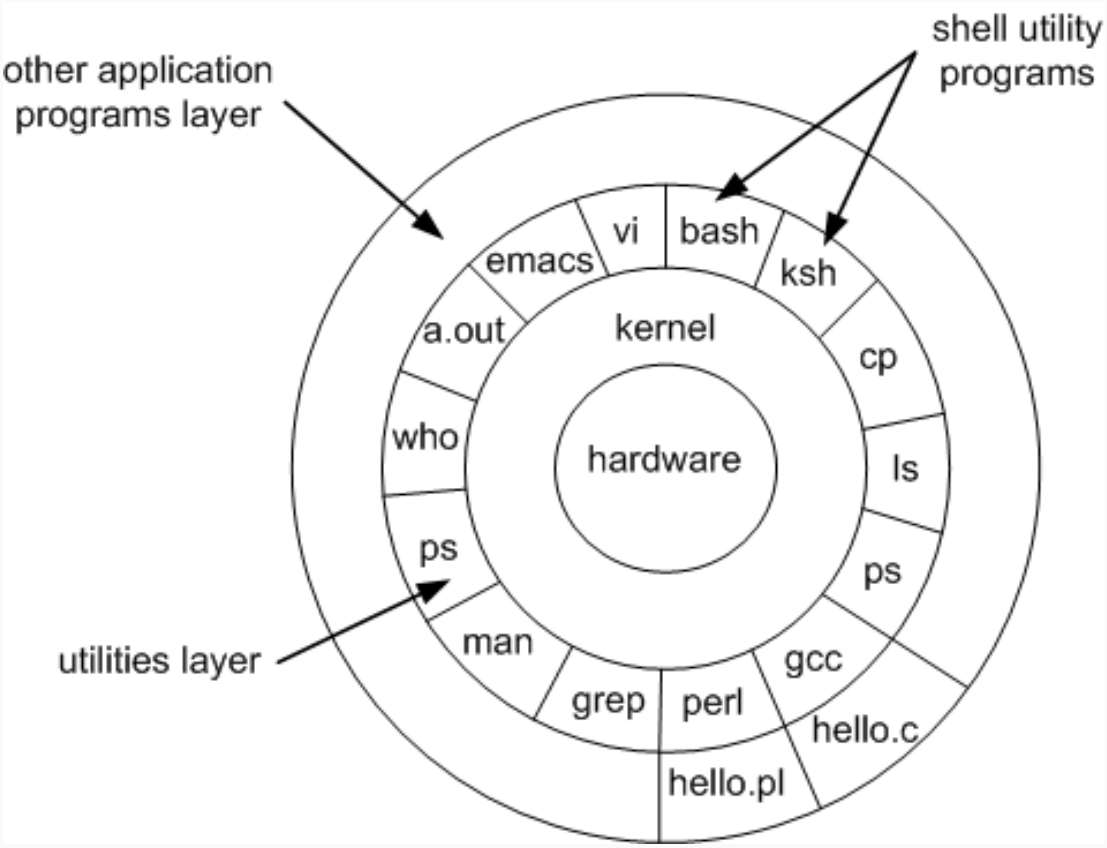
Table of contents

Introduction

Operating System (Overview)

- acts as and intermediary between a user and the hardware;
- manages computer hardware and software resources;
- large and complex software, implemented in layers;
- variety in purpose, design and implementation;

Operating System (Overview)



http://homepages.uc.edu/~thomam/Intro_Unix_Text/OS_Organization.html

Linux Distributions

The Shell

The Shell

- collection of utilities, text-based interface;
- allows users to communicate with the computer;
- allows programs to be started and tasks to be run;

- **sh** (Bourne shell) - basic shell, all UNIX systems;
- **ksh** (Korn shell) - Bourne shell enhanced;
- **csh** (C shell) - syntax similar to the C programming language;
- **bash** (Bourne Again Shell) - combines Korn Shell and C Shell, default in most Linux distributions;

Unix/Linux commands

- file commands (ls, cd, pwd, rm, mv, cat, head);
- process management (ps, top, kill, bg, fg, jobs);
- file permissions (chmod, chown);
- system information (whoami, uname, man, du, df, free, which);
- compression (tar, gzip);
- network (ping, dig, wget, ssh);
- build and install (make, cmake, automake, dpkg, rpm, gcc);
- shortcuts (Ctrl+C, Ctrl+Z);

cheat sheet - `https://files.fosswire.com/2007/08/fwunixref.pdf`

Unix/Linux commands

```
<cmd> --help
ps --help
du -help

man <cmd>
man tar
man uname

apropos <cmd>
apropos network
apropos disk
```

search the Internet - `https://www.google.com`
cheat sheet -
`https://files.fosswire.com/2007/08/fwunixref.pdf`

Bash Scripting

Haddop configuration - snippet

```
#!/bin/bash

export HADOOP_PREFIX=$(pwd)/download/hadoop-2.7.4
master="null"

while read line
do
  if [ "$master" == "null" ]
  then
    master=$line
    echo "$line" > $HADOOP_PREFIX/etc/hadoop/slaves
  else
    echo "$line" >> $HADOOP_PREFIX/etc/hadoop/slaves
  fi
done < hadoop-config.cfg
```

Hadoop configuration - snippet

```
#!/bin/bash

export HADOOP_PREFIX=$(pwd)/download/hadoop-2.7.4
master="null"

head -n 24 $HADOOP_PREFIX/etc/hadoop/hadoop-env.sh > temp.txt
echo "export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64" \
  >> temp.txt
echo "export HADOOP_PREFIX=$(pwd)/download/hadoop-2.7.4" >> temp.txt
head -n -25 $HADOOP_PREFIX/etc/hadoop/hadoop-env.sh >> temp.txt
cat temp.txt > $HADOOP_PREFIX/etc/hadoop/hadoop-env.sh

rm temp.txt
```

Iterate program - snippet

```
#!/bin/bash

function clear_cache {
  sync
  sudo sh -c 'echo 3 > /proc/sys/vm/drop_caches'
}

export CLASSPATH="lib/lucene-7.1.0/build/core/classes/java/:bin"
path_hdd="/home/loki/data/"
path_ssd="/storage/data/"
log="iteration.log"

echo -n "" > $log
for i in {1..10}
do
  file="dataset$(($i * 200))MB.txt"

  clear_cache
  java XSearchData $path_hdd$file $path_hdd$terms &>> $log
done
```

Bash Scripting

- ”gluing together” system calls, tools, utilities, and binary programs;
- automate configuration steps and file editing procedures;
- automate program iteration under variant arguments;
- especially suited for system administration tasks;

more resource - `http://tldp.org/LDP/abs/html/`

Secure Shell

POSIX Threads

BSD Sockets

Last remarks

Where there is a shell, there is a way.

Introduction/Tutorial on the Linux Ecosystem

Hyperion cluster, SLURM scheduler, Measuring execution time (part 3)

Alexandru Iulian Orhean
aorhean@hawk.iit.edu

Illinois Institute of Technology
Computer Science Department
Data-Intensive Distributed Systems Laboratory

Table of contents

Hyperion cluster

Hyperion cluster

- The platform where you will run PA1;
- The platform where we will grade PA1;

```
localhost$ ssh <username>@129.114.33.105
hyperion$ passwd
```

- SLURM - free & open-source job scheduler for Linux and UNIX*;
- allocates computer resources to users in an **exclusive** and/or non-exclusive mode, for a limited amount of time;
- provides a framework for starting, executing and monitoring work on a set of allocated nodes;
- arbitrates contention for resources by managing a job queue;

SLURM scheduler

SLURM commands

```
$ sinfo
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
interactive    up    1:00:00     10   idle bluecompute-[1-10]
compute*       up      15:00     50   idle redcompute-[1-50]

$ squeue
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST
   46   compute     bash  aorhean  R       0:40      1 redcompute-1

$ scancel 46

$ srun -n 1 -p interactive --pty /bin/bash

$ sbatch run.slurm
```

SLURM job script

```
#!/bin/bash

#SBATCH --nodes=2
#SBATCH --output=main.out
#SBATCH --wait-all-nodes=1

echo $SLURM_JOB_NODELIST

./myprogram /tmp/input.txt /tmp/output.txt
```

Measuring execution time

User time vs System time vs Wall time

```
$ time du -sh /home
8.4G    /home/

real    0m17.233s
user    0m0.350s
sys     0m1.850s
```

- Wall (Real) time -> total time from start to finish, including wait time (end of process quanta or waiting for I/O to complete);
- User time -> total time spent on the CPU in user space (other processes and time the processes spends blocked do not count);
- Sys time -> total time spent on the CPU in kernel space (other processes and time the processes spends blocked do not count);

C example - CPU time

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    clock_t start, end;
    start = clock();
    sleep(10);
    end = clock();

    printf("elapsed: %fs\n", (((float) end - start)
        / CLOCKS_PER_SEC));
    return 0;
}

$ gcc -Wall main.c
$ ./a.out
elapsed: 0.000034s
```

C example - high precision Wall time

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    struct timeval  start, end;
    gettimeofday(&start, NULL);
    sleep(10);
    gettimeofday(&end, NULL);

    printf("elapsed: %fs\n",
        (float) (end.tv_usec - start.tv_usec) / 1000000 +
        (float) (end.tv_sec - start.tv_sec));
    return 0;
}

$ gcc -Wall main.c
$ ./a.out
elapsed: 10.000120s
```

Hyperion upgrade

Hyperion upgrade

Friday 2nd of Feb scheduled upgrade!

- not accessible between 6pm-12am;
- upgrade the login node
  (more ram & more cores)
- add more compute nodes
  (aiming for a total of 90 compute nodes)