TCSS 562: SOFTWARE ENGINEERING FOR CLOUD COMPUTING

Introduction

Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma

1

FEEDBACK FROM 9/25

- Perspective on material: 6.625 (→ *mostly new to me*)
- Pace: 5.625 (→ *slightly fast*)
- 24 respondents
- What are the prerequisites for TCSS 562?
  - Admission into the MSCSS program
- What will the midterm exam be like?
  - To be an example, a practice midterm is given in class the class before the midterm
  - Questions have a variety of formats (multiple choice, fill in the blank, problems, etc.)

2

FEEDBACK - 2

- Tutorials grading (20%)
  - Objective is to complete 7 tutorials
  - Tutorials typically have a list of questions to answer, or require a final result to be provided to verify completion
  - Additional tutorials (>7) can be completed. Top 7 scores used
- Presentation grading (15%)
  - Cloud technology or research paper presentation
  - Rubric to be provided with the assignments when posted

3

FEEDBACK - 3

- Level of abstraction in the cloud: IaaS, PaaS, FaaS
  - Low to high abstraction: IaaS → PaaS → SaaS
  - These are “cloud computing delivery models”
  - A good question might be, given two delivery models, which provides more hardware (HW) abstraction?
  - IaaS and PaaS ?
  - PaaS and SaaS ?
  - FaaS and PaaS ?
  - Cloud computing delivery models are covered in depth, coming up
- What Linux commands can be used to gauge CPU utilization?
  - top –d 1
  - htop

4

FEEDBACK - 4

- Does TCSS 562 cover distributed system protocols?
  - These fall into TCSS 558 Applied Distributed Computing
  - TCSS 562: load balancing…

5

DEMOGRAPHICS SURVEY

- Please complete the ONLINE demographics survey:
- https://forms.gle/dE4Q7Lt13rAXtahJ9
- Linked from course webpage in Canvas:
- http://faculty.washington.edu/wlloyd/courses/tcss562/announcements.html

6

OBJECTIVES

- **Cloud Computing: How did we get here?**
  - *Parallel and distributed systems (Marinescu Ch. 2: 1st edition, or Ch. 4: 2nd edition)*
  - Data, thread-level, task-level parallelism
  - Parallel architectures
  - SIMD architectures, vector processing, multimedia extensions
  - Graphics processing units
  - Speed-up, Amdahl's Law, Scaled Speedup
  - Properties of distributed systems
  - Modularity
  - Functional vs. Non-functional requirements

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.7

7

CLOUD COMPUTING: HOW DID WE GET HERE?

- Compute clouds are large-scale distributed systems
- Infrastructure-as-a-Service (IaaS)
  - Provide VMs on demand to users
  - ***ec2instances.info*** (AWS EC2)
- Clouds can consist of
  - **Homogeneous** hardware (servers, etc.)
  - **Heterogeneous** hardware (servers, etc.)
- Which is preferable?

September 25, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L1.8

8

HARDWARE HETEROGENEITY

- **If providing IaaS, what are advantages/ disadvantages of using homogeneous hardware?**
  - Easier to provide same quality of service to end users
    - Less performance variance
    - Components with variable performance: CPUs, memory (speed differences), disks (SSDs, HDDs), network interfaces (caches?)
  - Homogeneous hardware (servers): components are interchangeable
    - As components fail, identical backups are immediately available
    - Example: **blade servers**
  - As clouds grow, why is HW homogeneity difficult to maintain?
- **What are some advantages of using heterogeneous HW?**

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.9

9

TYPES OF PARALLELISM

- Parallelism:
  - Goal: Perform multiple operations at the same time to achieve a speed-up
- Thread-level parallelism (TLP)
  - Control flow architecture
- Data-level parallelism
  - Data flow architecture
- Bit-level parallelism
- Instruction-level parallelism (ILP)

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.10

10

THREAD LEVEL PARALLELISM (TLP)

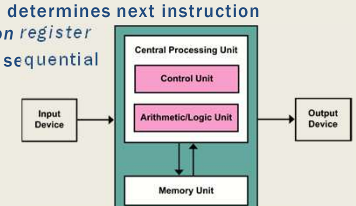
- Number of threads an application runs at any one time
- Varies throughout program execution
- As a metric:
- **Minimum**: 1 thread
- Can measure **average**, **maximum (peak)**
- **QUESTION: What are the consequences of average (TLP) for scheduling an application to run on a computer with a fixed number of CPU cores and hyperthreads?**
- Let's say there are 4 cores, or 8 hyper-threads…
- ***Key to avoiding waste of computing resources is knowing your application's TLP…***

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.11

11

CONTROL-FLOW ARCHITECTURE

- By John von Neumann (1945)
- Also called the Von Neumann architecture
- Dominant computer system architecture
- Program counter (PC) determines next instruction to load into *instruction register*
- Program execution is sequential



September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.12

12

DATA-LEVEL PARALLELISM

- Partition data into big chunks, run separate copies of the program on them with little or no communication
- Problems are considered to be ***embarrassingly parallel***
- Also perfectly parallel or pleasingly parallel…
- Little or no effort needed to separate problem into a number of parallel tasks
- MapReduce programming model is an example

13

DATA FLOW ARCHITECTURE

- ***Alternate architecture*** used by network routers, digital signal processors, special purpose systems
- Operations performed when input (data) becomes available
- Envisioned to provide much higher parallelism
- Multiple problems has prevented wide-scale adoption
  - Efficiently broadcasting data tokens in a massively parallel system
  - Efficiently dispatching instruction tokens in a massively parallel system
  - Building content addressable memory large enough to hold all of the dependencies of a real program

14

DATA FLOW ARCHITECTURE - 2

- Architecture not as popular as control-flow
- Modern CPUs emulate data flow architecture for dynamic instruction scheduling since the 1990s
  - Out-of-order execution – reduces CPU idle time by not blocking for instructions requiring data by defining execution windows
  - Execution windows: identify instructions that can be run by data dependency
  - Instructions are completed in data dependency order within execution window
    - Execution window size typically 32 to 200 instructions

Utility of data flow architectures has been much less than envisioned

15

BIT-LEVEL PARALLELISM

- Computations on large words (e.g. 64-bit integer) are performed as a single instruction
- Fewer instructions are required on 64-bit CPUs to process larger operands (A+B) providing dramatic performance improvements
- Processors have evolved: 4-bit, 8-bit, 16-bit, 32-bit, 64-bit

***QUESTION:** How many instructions are required to add two 64-bit numbers on a 16-bit CPU? (Intel 8088)*

- 64-bit MAX int = 9,223,372,036,854,775,807 (signed)
- 16-bit MAX int = 32,767 (signed)
- Intel 8088 – limited to 16-bit registers

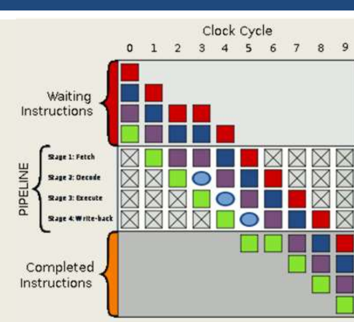
16

INSTRUCTION-LEVEL PARALLELISM (ILP)

- CPU pipelining architectures enable ILP
- CPUs have multi-stage processing pipelines
- Pipelining: split instructions into sequence of steps that can execute concurrently on different CPU circuitry
- Basic RISC CPU - Each instruction has 5 pipeline stages:
- **IF** – *instruction fetch*
- **ID**- *instruction decode*
- **EX** – *instruction execution*
- **MEM** – *memory access*
- **WB** – *write back*

17

CPU PIPELINING



18

INSTRUCTION LEVEL PARALLELISM - 2

- RISC CPU:
- After 5 clock cycles, all 5 stages of an instruction are loaded
- Starting with 6th clock cycle, one full instruction completes each cycle
- The CPU performs 5 tasks per clock cycle!
  Fetch, decode, execute, memory read, memory write back
- Pentium 4 (CISC CPU) – processing pipeline w/ 35 stages!

19

MICHAEL FLYNN'S COMPUTER ARCHITECTURE TAXONOMY

- Michael Flynn's proposed taxonomy of computer architectures based on concurrent instructions and number of data streams (1966)
- **SISD (Single Instruction Single Data)**
- **SIMD (Single Instruction, Multiple Data)**
- **MIMD (Multiple Instructions, Multiple Data)**
- *LESS COMMON*: MISD (Multiple Instructions, Single Data)
- Pipeline architectures: functional units perform different operations on the same data
- For fault tolerance, may want to execute same instructions redundantly to detect and mask errors – for task replication

20

FLYNN'S TAXONOMY

- **SISD (Single Instruction Single Data)**
  Scalar architecture with one processor/core.
  - Individual cores of modern multicore processors are "SISD"
- **SIMD (Single Instruction, Multiple Data)**
  Supports vector processing
  - When SIMD instructions are issued, operations on individual vector components are carried out concurrently
  - Two 64-element vectors can be added in parallel
  - Vector processing instructions added to modern CPUs
  - Example: Intel MMX (multimedia) instructions

21

(SIMD): VECTOR PROCESSING ADVANTAGES

- Exploit data-parallelism: vector operations enable speedups
- Vectors architecture provide vector registers that can store entire matrices into a CPU register
- SIMD CPU extension (e.g. MMX) add support for vector operations on traditional CPUs
- Vector operations reduce total number of instructions for large vector operations
- Provides higher potential speedup vs. MIMD architecture
- Developers can think sequentially; not worry about parallelism

22

FLYNN'S TAXONOMY - 2

- **MIMD (Multiple Instructions, Multiple Data)** - system with several processors and/or cores that function asynchronously and independently
- At any time, different processors/cores may execute different instructions on different data
- Multi-core CPUs are MIMD
- Processors share memory via interconnection networks
  - Hypercube, 2D torus, 3D torus, omega network, other topologies
- MIMD systems have different methods of sharing memory
  - Uniform Memory Access (UMA)
  - Cache Only Memory Access (COMA)
  - Non-Uniform Memory Access (NUMA)

23

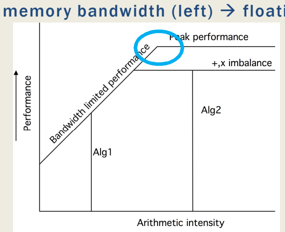
ARITHMETIC INTENSITY

- **Arithmetic intensity**: Ratio of work (W) to memory traffic r/w (Q) $I = \frac{W}{Q}$
  Example: # of floating point ops per byte of data read
- Characterizes application scalability with SIMD support
  - *SIMD can perform many fast matrix operations in parallel*
- ***High arithmetic Intensity:***
  Programs with dense matrix operations scale up nicely (many calcs vs memory RW, supports lots of parallelism)
- ***Low arithmetic intensity:***
  Programs with sparse matrix operations do not scale well with problem size
  (memory RW becomes bottleneck, not enough ops!)

24

ROOFLINE MODEL

- When program reaches a given arithmetic intensity performance of code running on CPU hits a "roof"
- CPU performance bottleneck changes from: memory bandwidth (left) → floating point performance (right)



Key take-aways:
When a program's has **low** Arithmetic Intensity, memory bandwidth limits performance..

With **high** Arithmetic intensity, the system has peak parallel performance…
→ *performance is limited by??*

25

GRAPHICAL PROCESSING UNITS (GPUs)

- GPU provides multiple SIMD processors
- Typically 7 to 15 SIMD processors each
- 32,768 total registers, divided into 16 lanes (2048 registers each)
- GPU programming model: single instruction, multiple thread
- Programmed using CUDA- C like programming language by NVIDIA for GPUs
- CUDA threads – single thread associated with each data element (e.g. vector or matrix)
- Thousands of threads run concurrently

26

PARALLEL COMPUTING

- Parallel hardware and software systems allow:
  - Solve problems demanding resources not available on single system.
  - Reduce time required to obtain solution
- The *speed-up* (S) measures effectiveness of parallelization:

$$S(N) = T(1) / T(N)$$

T(1) → execution time of total sequential computation
T(N) → execution time for performing N parallel computations in parallel

27

SPEED-UP EXAMPLE

- Consider embarrassingly parallel image processing
- Eight images (multiple data)
- Apply image transformation (greyscale) in parallel
- 8-core CPU, 16 hyperthreads
- Sequential processing: perform transformations one at a time using a single program thread
  - 8 images, 3 seconds each: `T(1) = 24 seconds`
- Parallel processing
  - 8 images, 3 seconds each: `T(N) = 3 seconds`
- Speedup: `S(N) = 24 / 3 = 8x speedup`
- Called "**perfect scaling**"
- Must consider data transfer and computation setup time

28

AMDAHL'S LAW

- Portion of computation which cannot be parallelized determines the overall speedup
- For an embarrassingly parallel job of fixed size
- Assuming no overhead for distributing the work, and a perfectly even work distribution

  α: fraction of program run time which can't be parallelized (e.g. must run sequentially)

- Maximum speedup is:

$$S = 1/ \alpha$$

- **Example:**
  Consider a program where 25% cannot be parallelized
  Q: What is the maximum possible speedup of the program?

29

GUSTAFSON'S LAW

- Calculates the ***scaled speed-up*** using "N" processors

$$S(N) = N + (1 - N)\, \alpha$$

N: Number of processors
α: fraction of program run time which can't be parallelized (e.g. must run sequentially)

- **Example:**
  Consider a program that is embarrassingly parallel, but 25% cannot be parallelized. α=.25
  QUESTION: If deploying the job on a 2-core CPU, what scaled speedup is possible assuming the use of two processes that run in parallel?

30

GUSTAFSON'S EXAMPLE

- **QUESTION:**
  What is the maximum theoretical speed-up on a **2-core CPU** ?
- S(N) = N + (1 - N) α
- N=2, α=.25
- S(N) = 2 + (1 - 2) .25
- S(N) = ?

- What is the maximum theoretical speed-up on a **4-core CPU**?
- S(N) = N + (1 - N) α
- N=4, α=.25
- S(N) = 4 + (1 - 4) .25
- S(N) = ?

31

MOORE'S LAW

- Transistors on a chip doubles approximately every 1.5 years
- CPUs now have billions of transistors
- Power dissipation issues at faster clock rates leads to heat removal challenges
  - Transition from: increasing clock rates → to adding CPU cores
- ***Symmetric core processor*** –multi-core CPU, all cores have the same computational resources and speed
- ***Asymmetric core processor*** – on a multi-core CPU, some cores have more resources and speed
- ***Dynamic core processor*** – processing resources and speed can be dynamically configured among cores
- ***Observation: asymmetric processors offer a higher speedup***

32

DISTRIBUTED SYSTEMS

- Collection of autonomous computers, connected through a network with distribution software called "middleware" that enables coordination of activities and sharing of resources
- **Key characteristics:**
- Users perceive system as a single, integrated computing facility.
- Compute nodes are autonomous
- Scheduling, resource management, and security implemented by every node
- Multiple points of control and failure
- Nodes may not be accessible at all times
- System can be scaled by adding additional nodes
- Availability at low levels of HW/software/network reliability

33

DISTRIBUTED SYSTEMS - 2

- Key non-functional attributes
  - Known as "ilities" in software engineering
- Availability – 24/7 access?
- Reliability - Fault tolerance
- Accessibility – reachable?
- Usability – user friendly
- Understandability – can under
- Scalability – responds to variable demand
- Extensibility – can be easily modified, extended
- Maintainability – can be easily fixed
- Consistency – data is replicated correctly in timely manner

34

TRANSPARENCY PROPERTIES OF DISTRIBUTED SYSTEMS

- **Access transparency**: local and remote objects accessed using identical operations
- **Location transparency**: objects accessed w/o knowledge of their location.
- **Concurrency transparency**: several processes run concurrently using shared objects w/o interference among them
- **Replication transparency**: multiple instances of objects are used to increase reliability
  - users are unaware if and how the system is replicated
- **Failure transparency**: concealment of faults
- **Migration transparency**: objects are moved w/o affecting operations performed on them
- **Performance transparency**: system can be reconfigured based on load and quality of service requirements
- **Scaling transparency**: system and applications can scale w/o change in system structure and w/o affecting applications

35

TYPES OF MODULARITY

- ***Soft modularity:*** TRADITIONAL
- Divide a program into modules (classes) that call each other and communicate with shared-memory
- A procedure calling convention is used (or method invocation)
- Object-oriented programming classic best practices:
- **Minimize coupling** between classes (OO) and modules
- **Maximize cohesion** between functions in classes (OO) and modules
  - Best practices lead to improved software reusability, maintainability, portability

36

TYPES OF MODULARITY - 2

- ***Enforced modularity:*** CLOUD COMPUTING
- Program is divided into modules that communicate only through message passing
- The ubiquitous ***client-server*** paradigm
- Clients and servers are independent decoupled modules
- System is more robust if servers are stateless
- May be scaled and deployed separately
- May also FAIL separately!

COUPLING AND COHESION

- **Object-oriented coupling**
- Degree of interdependence between software modules
- A measure of how connected two classes or modules are
- Captures the degree of the relationships between modules
- Coupling is usually contrasted with cohesion
- Low coupling often correlates with high cohesion
- High coupling often correlates with low cohesion

- **Object-oriented cohesion**
- Degree to which elements inside a class or module belong together
- Do the methods and data inside of a class interoperate with each other (***High cohesion***)? Or is the class a catch all bin of random functions (***Low cohesion***)?
  - E.g. "Util" class where random helper routines land… (***low cohesion***)

FUNCTIONAL VS. NON-FUNCTIONAL ATTRIBUTES OF SYSTEMS

- **Functional requirement:**
- Pertains to a system supporting a specific function
- What a system is supposed to do
- Testable with unit tests, integration tests, etc.

- **Non-functional requirement:**
- Specifies criteria used to judge how a system operates
- How a system should be (or behave)
- Considered as "quality" attributes of systems
- Testable by applying metrics to characterize degree of possessing a given quality

NON-FUNCTIONAL REQUIREMENT: HIGH AVAILABILITY

- ***The system should be highly available.***
- The system should be **99.9%** available per month
  - Maximum downtime: **43m 49.7s** monthly, **8hr 45min 36s** yearly
  - Functional attribute: system should notify users if there is an issue affects the availability or may cause downtime.
- Availability equation:

$$\text{AVAILABILITY} = \frac{\text{MTBF}}{\text{MTBF}+\text{MTTR}}$$

- MTBF: Mean time between failures
- MTTR: Mean time to Repair
- MTBF = ~1 month = 43,757 min; MTTR = 43 min
- AVAILABILITY = 43757 / 43800 = 99.9018265%

STRATEGIES FOR HIGH AVAILABILITY

- Replicate system resources in multiple data centers or cloud computing regions
- Use redundant infrastructure components
  - For load balancing, fault tolerance
- Report availability status via portal
- Allow users to immediately report outages
- Notification systems to alert system admins when system experiences an outage

- **Tradeoffs:**
- Highly available cloud resources are more expensive
- Replicating app components (e.g. database) adds cost

QUANTIFYING NON-FUNCTIONAL QUALITY ATTRIBUTES

- What are the "best" metrics to quantify non-functional quality attributes?
  - Consider ease/effort/time/cost of assessment
  - Relationship to expert opinion (e.g. correlation)
  - Relationship to other measures

CLOUD COMPUTING – HOW DID WE GET HERE? SUMMARY OF KEY POINTS

- Multi-core CPU technology and hyper-threading
- What is a
  - Heterogeneous system?
  - Homogeneous system?
  - Autonomous or self-organizing system?
- **Fine grained vs. coarse grained parallelism**
- Parallel message passing code is easier to debug than shared memory (e.g. p-threads)
- Know your application's max/avg **Thread Level Parallelism** (***TLP***)
- **Data-level parallelism**: Map-Reduce, (SIMD) Single Instruction Multiple Data, Vector processing & GPUs

43

CLOUD COMPUTING – HOW DID WE GET HERE? SUMMARY OF KEY POINTS - 2

- **Bit-level parallelism**
- **Instruction-level parallelism** (CPU pipelining)
- **Flynn's taxonomy**: computer system architecture classification
  - **SISD** – Single Instruction, Single Data (modern core of a CPU)
  - **SIMD** – Single Instruction, Multiple Data (Data parallelism)
  - **MIMD** – Multiple Instruction, Multiple Data
  - MISD is RARE; application for fault tolerance…
- **Arithmetic intensity**: ratio of calculations vs memory RW
- **Roofline model:**
  Memory bottleneck with low arithmetic intensity
- **GPUs**: ideal for programs with high arithmetic intensity
  - SIMD and Vector processing supported by many large registers

44

CLOUD COMPUTING – HOW DID WE GET HERE? SUMMARY OF KEY POINTS - 3

- **Speed-up (S)**
  $S(N) = T(1) / T(N)$
- **Amdahl's law:**
  $S = 1/ \alpha$
  α = percent of program that must be sequential
- **Scaled speedup with N processes:**
  $S(N) = N – \alpha(N-1)$
- Moore's Law
- Symmetric core, Asymmetric core, Dynamic core CPU
- Distributed Systems Non-function quality attributes
- Distributed Systems – Types of Transparency
- Types of modularity- Soft, Enforced
- Functional vs. Non-functional requirements

45

INTRODUCTION TO CLOUD COMPUTING

46

OBJECTIVES - 2

- **Introduction to Cloud Computing**
  - Why study cloud computing?
  - History of cloud computing
  - Business drivers
  - Cloud enabling technologies
  - Terminology
  - Benefits of cloud adoption
  - Risks of cloud adoption

47

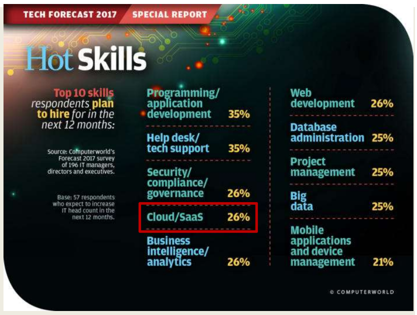
WHY STUDY CLOUD COMPUTING?

- LINKEDIN - TOP IT Skills from job app data
  - #1 Cloud and Distributed Computing
  - https://learning.linkedin.com/week-of-learning/top-skills
  - #2 Statistical Analysis and Data Mining
- FORBES Survey – 6 Tech Skills That'll Help You Earn More
  - #1 Data Science
  - #2 Cloud and Distributed Computing
  - http://www.forbes.com/sites/laurencebradford/2016/12/19/6-tech-skills-thatll-help-you-earn-more-in-2017/

48

WHY STUDY CLOUD COMPUTING? - 2

- Computerworld Magazine



September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.49

49

A BRIEF HISTORY OF CLOUD COMPUTING

- John McCarthy, 1961
  - Turing award winner for contributions to AI
- "If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility… The computer utility could become the basis of a new and important industry…"

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.50

50

CLOUD HISTORY - 2

- Internet based computer utilities
- Since the mid-1990s
- Search engines: Yahoo!, Google, Bing
- Email: Hotmail, Gmail

- 2000s
- Social networking platforms: MySpace, Facebook, LinkedIn
- Social media: Twitter, YouTube

- Popularized core concepts
- Formed basis of cloud computing

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.51

51

CLOUD HISTORY: SERVICES - 1

- Late 1990s – Early Software-as-a-Service (SaaS)
  - Salesforce: Remotely provisioned services for the enterprise

- 2002 -
  - Amazon Web Services (AWS) platform: Enterprise oriented services for remotely provisioned storage, computing resources, and business functionality

- 2006 – **Infrastructure-as-a-Service (IaaS)**
  - Amazon launches Elastic Compute Cloud (EC2) service
  - Organization can "lease" computing capacity and processing power to host enterprise applications
  - Infrastructure

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.52

52

CLOUD HISTORY: SERVICES - 2

- 2006 – **Software-as-a-Service (SaaS)**
  - Google: Offers Google DOCS, "MS Office" like fully-web based application for online documentation creation and collaboration

- 2009 – **Platform-as-a-Service (PaaS)**
  - Google: Offers Google App Engine, publicly hosted platform for hosting scalable web applications on google-hosted datacenters

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.53

53

CLOUD COMPUTING NIST GENERAL DEFINITION

"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (networks, servers, storage, applications and services) that can be rapidly provisioned and reused with minimal management effort or service provider interaction"…

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.54

54

MORE CONCISE DEFINITION

"Cloud computing is a specialized form of distributed computing that introduces utilization models for remotely provisioning scalable and measured resources."

From Cloud Computing Concepts, Technology, and Architecture
Z. Mahmood, R. Puttini, Prentice Hall, 5th printing, 2015

55

BUSINESS DRIVERS FOR CLOUD COMPUTING

- Capacity planning
- Cost reduction
- Operational overhead
- Organizational agility

56

BUSINESS DRIVERS FOR CLOUD COMPUTING

- Capacity planning
  - Process of determining and fulfilling future demand for IT resources
  - Capacity vs. demand
  - Discrepancy between capacity of IT resources and actual demand
  - Over-provisioning: resource capacity exceeds demand
  - Under-provisioning: demand exceeds resource capacity
  - Capacity planning aims to minimize the discrepancy of available resources vs. demand

57

Dwight, The Office TV sitcom

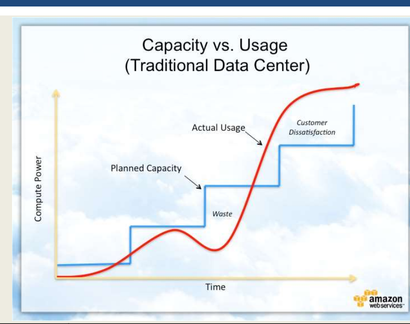
58

BUSINESS DRIVERS FOR CLOUD - 2

- Capacity planning
  - Over-provisioning: is costly due to too much infrastructure
  - Under-provisioning: is costly due to potential for business loss from poor quality of service
- Capacity planning strategies
  - Lead strategy: add capacity in anticipation of demand (pre-provisioning)
  - Lag strategy: add capacity when capacity is fully leveraged
  - Match strategy: add capacity in small increments as demand increases
- Load prediction
  - Capacity planning helps anticipate demand flucations

59

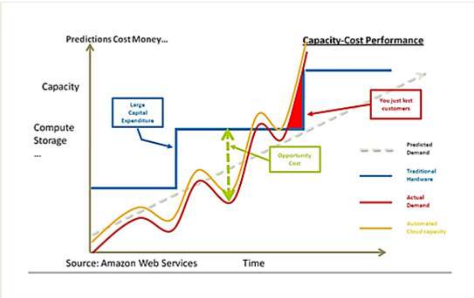
CAPACITY PLANNING



60

CAPACITY PLANNING - 2

- Ca



September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.61

61

BUSINESS DRIVERS FOR CLOUD - 3

- Cost reduction
  - IT Infrastructure acquisition
  - IT Infrastructure maintenance
- Operational overhead
  - Technical personnel to maintain physical IT infrastructure
  - System upgrades, patches that add testing to deployment cycles
  - Utility bills, capital investments for power and cooling
  - Security and access control measures for server rooms
  - Admin and accounting staff to track licenses, support agreements, purchases

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.62

62

BUSINESS DRIVERS FOR CLOUD - 4

- Organizational agility
  - Ability to adapt and evolve infrastructure to face change from internal and external business factors
  - Funding constraints can lead to insufficient on premise IT
  - Cloud computing enables IT resources to scale with a lower financial commitment

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.63

63

TECHNOLOGY INNOVATIONS LEADING TO CLOUD

- Cluster computing
- Grid computing
- Virtualization
- Others

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.64

64

CLUSTER COMPUTING

- Cluster computing (clustering)
  - Cluster is a group of independent IT resources interconnected as a single system
  - Servers configured with homogeneous hardware and software
    - Identical or similar RAM, CPU, HDDs
  - Design emphasizes redundancy as server components are easily interchanged to keep overall system running
    - Example: if a RAID card fails on a key server, the card can be swapped from another redundant server
  - Enables warm replica servers
    - Duplication of key infrastructure servers to provide HW failover to ensure high availability (HA)

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.65

65

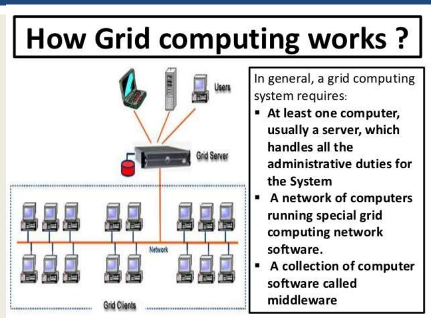
GRID COMPUTING

- On going research area since early 1990s
- Distributed heterogeneous computing resources organized into logical pools of loosely coupled resources
- For example: heterogeneous servers connected by the internet
- Resources are heterogeneous and geographically dispersed
- Grids use middleware software layer to support workload distribution and coordination functions
- Aspects: load balancing, failover control, autonomic configuration management
- Grids have influenced clouds contributing common features: networked access to machines, resource pooling, scalability, and resiliency

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.66

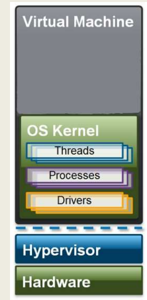
66

GRID COMPUTING - 2



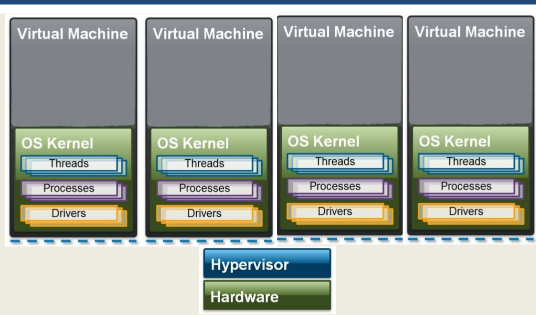
67

VIRTUALIZATION



68

VIRTUALIZATION



69

VIRTUALIZATION

- Simulate physical hardware resources via software
  - The virtual machine (virtual computer)
  - Virtual local area network (VLAN)
  - Virtual hard disk
  - Virtual network attached storage array (NAS)
- Early incarnations featured significant performance, reliability, and scalability challenges
- CPU and other HW enhancements have minimized performance GAPs

70

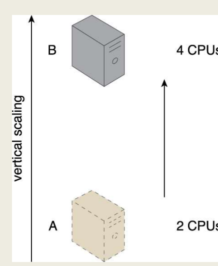
KEY TERMINOLOGY

- **On-Premise Infrastructure**
  - Local server infrastructure not configured as a cloud
- **Cloud Provider**
  - Corporation or private organization responsible for maintaining cloud
- **Cloud Consumer**
  - User of cloud services
- **Scaling**
  - **Vertical scaling**
    - Scale up: increase resources of a single virtual server
    - Scale down: decrease resources of a single virtual server
  - **Horizontal scaling**
    - Scale out: increase number of virtual servers
    - Scale in: decrease number of virtual servers

71

VERTICAL SCALING

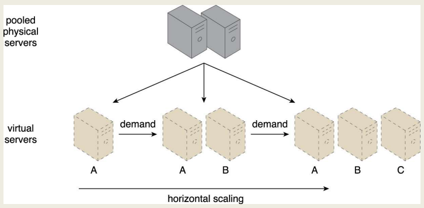
- Reconfigure virtual machine to have different resources:
  - CPU cores
  - RAM
  - HDD/SDD capacity
- May require VM migration if physical host machine resources are exceeded



72

HORIZONTAL SCALING

- Increase (scale-out) or decrease (scale-in) number of virtual servers based on demand



September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.73

73

HORIZONTAL VS VERTICAL SCALING

Horizontal Scaling	Vertical Scaling
Less expensive using commodity HW	Requires expensive high capacity servers

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.74

74

HORIZONTAL VS VERTICAL SCALING

Horizontal Scaling	Vertical Scaling
Less expensive using commodity HW	Requires expensive high capacity servers
IT resources instantly available	IT resources typically instantly available

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.75

75

HORIZONTAL VS VERTICAL SCALING

Horizontal Scaling	Vertical Scaling
Less expensive using commodity HW	Requires expensive high capacity servers
IT resources instantly available	IT resources typically instantly available
Resource replication and automated scaling	Additional setup is normally needed

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.76

76

HORIZONTAL VS VERTICAL SCALING

Horizontal Scaling	Vertical Scaling
Less expensive using commodity HW	Requires expensive high capacity servers
IT resources instantly available	IT resources typically instantly available
Resource replication and automated scaling	Additional setup is normally needed
Additional servers required	No additional servers required

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.77

77

HORIZONTAL VS VERTICAL SCALING

Horizontal Scaling	Vertical Scaling
Less expensive using commodity HW	Requires expensive high capacity servers
IT resources instantly available	IT resources typically instantly available
Resource replication and automated scaling	Additional setup is normally needed
Additional servers required	No additional servers required
Not limited by individual server capacity	Limited by individual server capacity

September 30, 2019 | TCSS562: Software Engineering for Cloud Computing [Fall 2019] School of Engineering and Technology, University of Washington - Tacoma | L2.78

78

KEY TERMINOLOGY - 2

- Cloud services
  - Broad array of resources accessible “as-a-service”
  - Categorized as Infrastructure (IaaS), Platform (PaaS), Software (SaaS)
- Service-level-agreements (SLAs):
  - Establish expectations for: uptime, security, availability, reliability, and performance

79

GOALS AND BENEFITS

- **Cloud providers**
  - Leverage economies of scale through mass-acquisition and management of large-scale IT resources
  - Locate datacenters to optimize costs where electricity is low
- **Cloud consumers**
  - Key business/accounting difference:
  - **Cloud computing enables anticipated capital expenditures to be replaced with operational expenditures**
  - Operational expenditures always scale with the business
  - Eliminates need to invest in server infrastructure based on anticipated business needs
  - Businesses become more agile and lower their financial risks by eliminating large capital investments in physical infrastructure

80

CLOUD BENEFITS - 2

- On demand access to pay-as-you-go resources on a short-term basis (less commitment)
- Ability to acquire “unlimited” computing resources on demand when required for business needs
- Ability to add/remove IT resources at a fine-grained level
- Abstraction of server infrastructure so applications deployments are not dependent on specific locations, hardware, etc.
  - The cloud has made our software deployments more agile…

81

CLOUD BENEFITS - 3

- Example: Using 100 servers for 1 hour costs the same as using 1 server for 100 hours
- Rosetta Protein Folding: Working with a UW-Tacoma graduate student, we recently deployed this science model across 5,900 compute cores on Amazon for 2-days…
- ***What is the cost to purchase 5,900 compute cores?***
- Recent Dell Server purchase example: 20 cores on 2 servers for $4,478…
- Using this ratio 5,900 cores costs $1.3 million (purchase only)

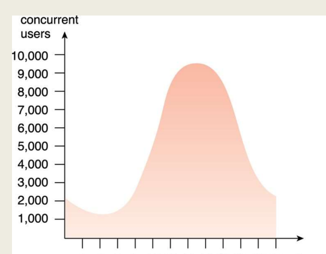
82

Gene Wilder, Charlie and the Chocolate Factory

83

CLOUD BENEFITS

- Increased scalability
  - Example demand over a 24-hour day →
- Increased availability
- Increased reliability



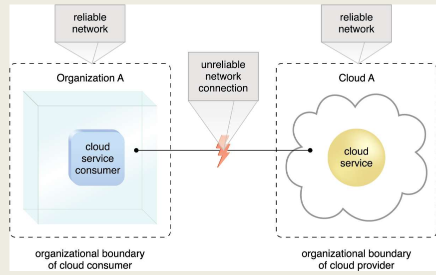
84

CLOUD ADOPTION RISKS

- **Increased security vulnerabilities**
  - Expansion of trust boundaries now include the external cloud
  - Security responsibility shared with cloud provider
- **Reduced operational governance / control**
  - Users have less control of physical hardware
  - Cloud user does not directly control resources to ensure quality-of-service
  - Infrastructure management is abstracted
  - Quality and stability of resources can vary
  - Network latency costs and variability

85

NETWORK LATENCY COSTS



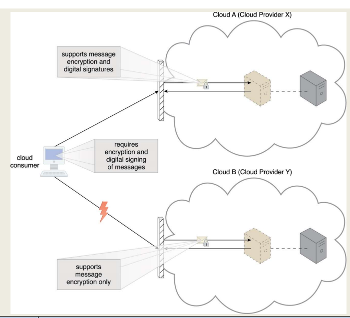
86

CLOUD RISKS - 2

- **Performance monitoring of cloud applications**
  - Cloud metrics (AWS cloudwatch) support monitoring cloud infrastructure (network load, CPU utilization, I/O)
  - Performance of cloud applications depends on the health of aggregated cloud resources working together
  - User must monitor this aggregate performance
- **Limited portability among clouds**
  - Early cloud systems have significant “vendor” lock-in
  - Common APIs and deployment models are slow to evolve
  - Operating system containers help make applications more portable, but containers still must be deployed
- **Geographical issues**
  - Abstraction of cloud location leads to legal challenges with respect to laws for data privacy and storage

87

CLOUD: VENDOR LOCK-IN



88

QUESTIONS

89