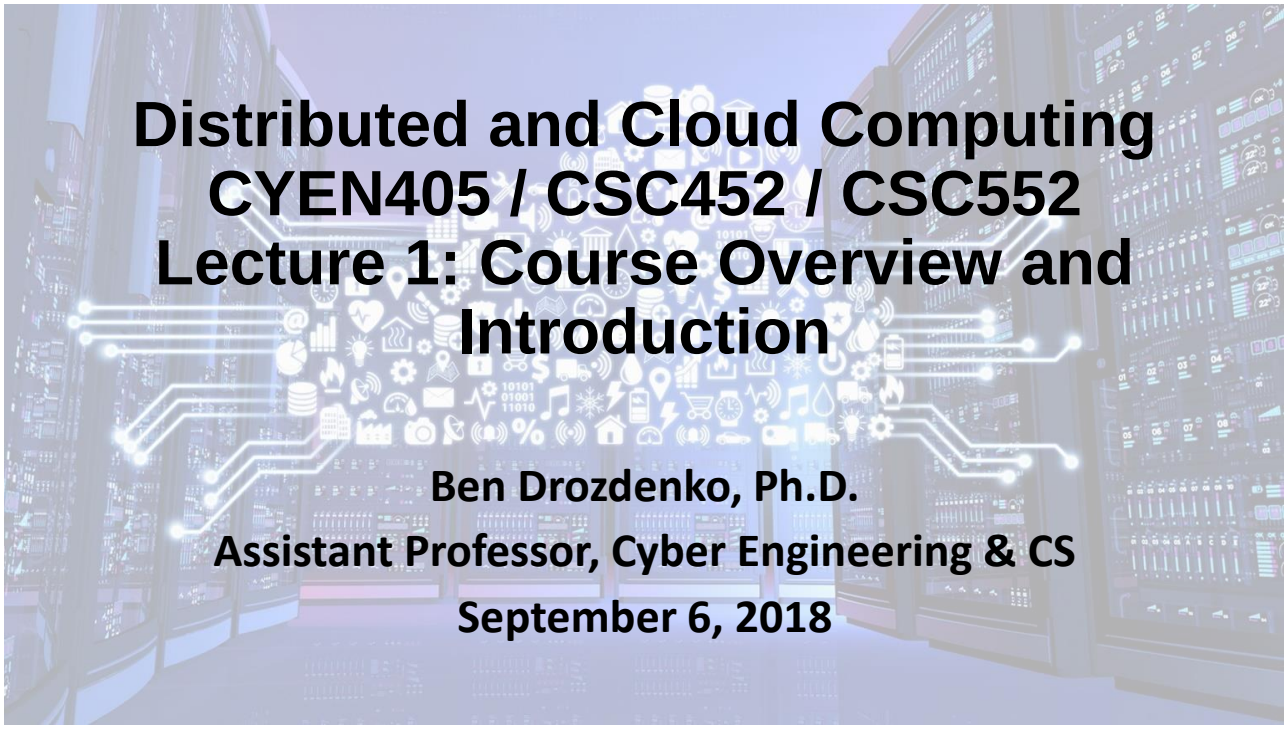


Distributed and Cloud Computing CYEN405 / CSC452 / CSC552 Lecture 1: Course Overview and Introduction

Ben Drozdenko, Ph.D.
Assistant Professor, Cyber Engineering & CS
September 6, 2018

Lecture 1: Outline
Reading: van Steen Ch. 1 (pp. 1-53)

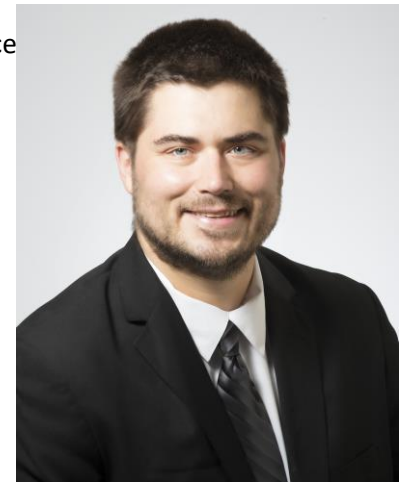
- 1.1 Class Logistics
  - 1.1.1 Instructor Information
  - 1.1.2 Classroom & Textbooks
  - 1.1.3 Course Description & Topics
  - 1.1.4 Student Responsibilities
  - 1.1.5 Class Participation
  - 1.1.6 Academic Integrity-Honesty
  - 1.1.7 Anticipated Schedule
  - 1.1.8 Assessments & Grading
  - 1.1.9 Moodle & Other Details
- 1.2 Introduction to DS
  - 1.2.1 What is a distributed system?
  - 1.2.2 Characteristics of DS's
  - 1.2.3 Middleware: the OS of DS's
  - 1.2.4 Application domains
  - 1.2.5 An example financial system
  - 1.2.6 Typical portion of the Internet
  - 1.2.7 Portable & handheld devices
  - 1.2.8 Growth of the Internet
  - 1.2.9 Web servers & web browsers
  - 1.2.10 Cloud computing

Lecture 1: Outline (cont.)
Opt. reading: Colouris ch. 1, Erl ch. 3

- 1.3 Introduction to Cloud
  - 1.3.1 Cloud Symbol
  - 1.3.2 Common IT Resources
  - 1.3.3 Example Cloud & IT resources
  - 1.3.4 IT resource scaling by adding
  - 1.3.5 IT resource scaling by replacing
  - 1.3.6 Remote access
  - 1.3.7 Cloud service consumers
  - 1.3.8 IT resource changing demand
  - 1.3.9 Overlap of trust boundaries
  - 1.3.10 Unreliable network effects
- 1.4 Distributed System Concepts
  - 1.4.1 DS Goals
  - 1.4.2 Sharing resources
  - 1.4.3 Distribution transparency
  - 1.4.4 Openness of DS's
  - 1.4.5 Scalability in DS's
  - 1.4.6 Pitfalls
- 1.5 Types of Distributed Systems
  - 1.5.1 High-performance computing
  - 1.5.2 Distributed information systems
  - 1.5.3 Pervasive systems

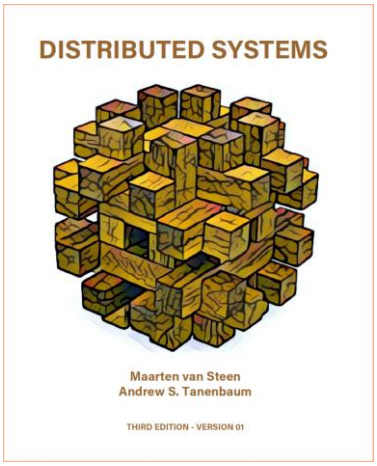
1.1.1: Instructor Information

- Instructor: "**Dr. Droz**". Benjamin Drozdenko, Ph.D.
  Assistant Professor of Cyber Engineering & Computer Science
  - Ph.D. in Computer Engineering, Northeastern U., 2017
  - M.S. in Electrical Engineering, Northeastern U., 2007
  - B.S. in Computer Science—Computer & Systems Engineering, Rensselaer Polytechnic Institute, 2004
  - Training Engineer—Signal Processing Content Specialist, MathWorks, Inc., 2008–2014
  - Systems Engineer, Raytheon Company, 2004–2008
- E-mail: droz@latech.edu
  - Put [ELEN333] into Subject of e-mail messages to me about class
- Office Hours for **Fall 2018**:
  Tuesday 9:00am–2:00pm, Thurs 9:00am–2:00pm
- Office Location: 117 Nethken Hall
- Office Phone: (318)257-2919
- Webpage: http://latech.edu/~droz



1.1.2: Classroom & Textbooks

- Lectures: Tues, Thurs 2:00–3:50 pm.
  - Nethken Hall 140.
- Labs: On your own time
- Textbook: [**Free**, Required]
  Maarten van Steen, Andrew S. Tannenbaum. 2017. **Distributed Systems, 3rd ed**.
  ISBN: 978-15-430573-8-6. Get PDF here: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017/ds3-sneak-preview/
- [Also recommended, since *I'll* be using them]:
  - George Coulouris, Jean Dollimore, Tim Kindberg. 2005. **Distributed Systems: Concepts and Design, 4th ed.** ISBN: 978-0-321-26354-4
  - Thomas Erl, Zaigham Mahmood, Ricardo Puttini. **Cloud Computing: Concepts, Technology & Architecture**. 2013. ISBN: 978-0133387520



1.1.3 Course Description & Topics

- Distributed Computing
- Cloud Computing
- Theoretical Aspects
- Applicable Aspects
- Modeling Aspects
  - Architecture
  - Performance
  - Reliability
  - Availability
  - Service Models
  - Security Characteristics
- Distributed Systems and Cloud Models
  - Characterization, System Models, Architectures
- Distributed Computation
  - Operating System Support, Processes and Threads, Virtualization
- Distributed Communication
  - Interprocess Communication
  - Networking and Internetworking
  - Distributed Objects and Remote Invocation
- Distributed Security, File Systems, and Naming
- Distributed Algorithms and Mechanisms
  - Timing, Coordination, and Agreement
  - Synchronization and Mutual Exclusion
- Distributed Shared Datasets
  - Transactions, Consistency, and Replication

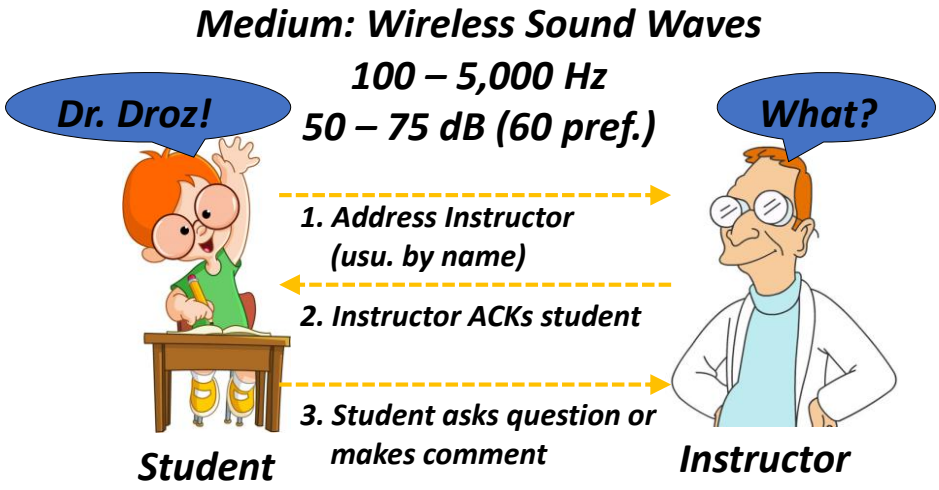
1.1.4 Student Responsibilities

- Attend every class session ***ON TIME***
  - Class attendance is governed by regulations published each year in the university bulletin
    (Ch. 3—Academic Policies, p. 12)
  - All missed assignments and class material is the responsibility of the student
  - Instructor will not provide missed lecture notes

1.1.5 Class Participation

- If you cannot attend (due to reasons medical, family, etc.), notify instructor (pref. both in person and by e-mail) 24 hours in advance
- Participation in class discussions will help you learn the course material and do well in the pop quizzes
- On a daily basis, all students are encouraged to participate in the class lecture
  - I'll ask questions and expect responses from students in front few rows
- This encourages you to think on your own
- Respond to my questions orally in class
  - If you did not read the textbook material, then please just leave
- Although this is not counted for credit, you'll be asked questions from this discussion on pop quizzes

Figure 1.1: ***Student-Initiated Class Participation Procedure***



This is a communications protocol!

1.1.6 Academic Integrity and Honesty

- Academic integrity and honesty are expected. Students should read carefully the Louisiana Tech Academic Honor Code
  - http://www.latech.edu/documents/honor-code.pdf
- Cheating, plagiarism, or other violations of the Academic Honor Code will result in a failing grade for the assignment or exam and may result in failing the course or expulsion from the university
- University policy allows for sanctions for violations of academic integrity ranging from zero credit for an assignment to expulsion (without expectation of readmission) from the University
- **Any student who is found to have violated the University's Academic Honor Code in this course, no matter how minor the violation, will at a minimum receive a 0 for the assignment or exam**
- **For serious violations, I will pursue sanctions of failure in the course or expulsion from the University.**

1.1.7 Anticipated Schedule (Fall 2018 1st Half)
Note: subject to change

Wk	Date	Topics	Reading Assignment
1	Sept 6	Overview, Distributed Systems, Cloud	Ch. 1, sec. 1.1-1.3
2	Sept 11	System Models, Architectures	Ch. 2, sec. 2.1-2.4, Sec. 8.1
2	Sept 13	Operating System Support, Threads	Ch. 3, sec. 3.1
3	Sept 18	Virtualization & VM Technology	Ch. 3, sec. 3.2
3	Sept 20	Cloud-Enabling Technology	Ch. 3, sec. 3.3-3.5, Sec. 8.2
4	Sept 25	Interprocess Communications, Networking	Ch. 4, sec. 4.1-4.4
4	Sept 27	Distributed Objects, Remote Invocation	Sec. 8.3-8.4
5	Oct 2	Security Fundamentals, Mechanisms	Ch. 9, sec. 9.1-9.3
5	Oct 4	File Systems, Name Services	Ch. 5, sec. 5.1-5.4, Sec. 9.4-9.5
6	Oct 9	Midterm Review	
6	Oct 11	Midterm Exam	

1.1.7 Anticipated Schedule (Fall 2018 2nd Half)
Note: subject to change

Wk	Date	Topics	Reading Assignment
7	Oct 16	Distributed Algorithms, Timing, Global States	Ch. 6
7	Oct 18	Coordination and Agreement	Ch. 6, Sec. 8.4
8	Oct 23	Cloud Computing Mechanisms	Ch. 6
8	Oct 25	Distributed Shared Datasets, Transactions	Sec. 8.5-8.6
9	Oct 30	Consistency and Replication	Ch. 7
9	Nov 1	Distributed Services, New Paradigms	Handouts
10	Nov 6	Working with Clouds, Case Studies	Handouts
10	Nov 8	Final Review	
11	Nov 13	Final Project Presentations	
11	Nov 15	Final Exam	

1.1.8 Assessments and Grading

- In-class Quizzes 8 % (est. 4 quizzes, 2 pts. each)
- Lab assignments 20 % (est. 4 labs, 5 pts. each)
- Homework 20 % (est. 4 homeworks, 5 pts. each)
- Mid-term Exam 20 % (exp. Thursday, Oct. 11, 2–3:50 pm)
- Final Exam 20 % (exp. Thursday, Nov. 15, 2–3:50 pm)
- Final Project 12 % (pres. Tues, Nov. 13; report Fri, Nov. 16)

Grades are rounded to the nearest tenth of an integer and determined from the final total points as follows:
A > 90.0%, B = 89.9-80.0%, C = 79.9-70.0%, D = 69.9-60.0%, F < 60.0%.

1.1.8.1 Assessments: Quizzes

- About once every other week, at the start of class I will hand out pop quizzes
- These quizzes assess whether you are completing the assigned textbook readings in time, paying attention during lectures, and taking notes (you may take notes on lecture print-outs)
- By making these quizzes sporadic, open notes and closed book, I ensure that you are self-motivated to read the material carefully and take detailed notes both at-home and in-class
- This only comprises 8% of the total grade (formative)

1.1.8.2 Assessments: Laboratory Assignments

- Once every other week, you will complete lab exercises
- The early lab exercises (#1 – 2) are meant to first introduce you to the tools you must use to design distributed systems
- In the latter lab exercises (#3 – 4), you must incorporate your own knowledge of distributed systems and cloud computing to complete the labs
- Since these are much more important to your ultimate success as an engineer or scientist, this comprises 20% of the total grade.

1.1.8.3 Assessments: Homework Assignments

- Once every other week, you will be assigned a take-home assignment
  - Starting Spring 2018 quarter, you may work in teams of 1-4 people
- You have one full week to complete each take-home assignment, and are expected to submit your responses via Moodle in PDF format by 11:55 p.m. on the day the HW is due
  - Make sure to put the names of all team members at top of submitted PDF
  - Show your work!
- Each homework requires both forming new knowledge from the prescribed reading materials and evaluating your knowledge against a set of correct answers
- This is 20% of the total grade.

1.1.8.4 Assessments: Examinations

- There will be an in-class midterm examination during a lab session, tentatively Wednesday, April 11, 2018 (4/11/18), 2 – 5 pm
- The midterm examination will cover all topics covered in the textbook and class up until this date (e.g. Ch. 1-2, Ch. 4 sec. 1-3 & 5)
- The final examination will be Wednesday, May 16, 2018, 2 – 5 pm
  - Covers all topics discussed in the course (e.g. Ch 1-5)
- Since memorization is important at this introductory stage, I will probably make both exams closed-notes (closed phone/computer)
- Each exam counts for 20% of the total grade (40% total)

1.1.9 Moodle & Other Items

- Moodle Class Link: http://moodle.latech.edu/course/view.php?id=58148
- Submit all assignments via Moodle in PDF format
  - Best HW file name: "CYEN405_HW***X***_LastnameFirstname.pdf"
- The issuance of incomplete grades will strictly follow the College of Engineering and Science guidelines
- An incomplete will only be given for missed work at the end of the term due to illness.

1.2.1 Distributed System Definition

- ***Distributed System (DS)*****:** A collection of **autonomous computing elements** that appears to its users as a **single coherent system**

- Characteristic features of DSs:
  1. Autonomous computing elements, a.k.a. **nodes**, be they hardware devices or software processes
  2. Single coherent system: users or applications perceive a single system nodes need to collaborate

1.2.2 Characteristic 1: Collection of Autonomous Nodes

- Independent behavior
  - Each node is autonomous and will therefore have its **own notion of time**: there is no global clock
  - Leads to fundamental synchronization and coordination problems

- Collection of nodes
  - How to manage **group membership**?
  - How to know that you are indeed communicating with an **authorized (non)member**?

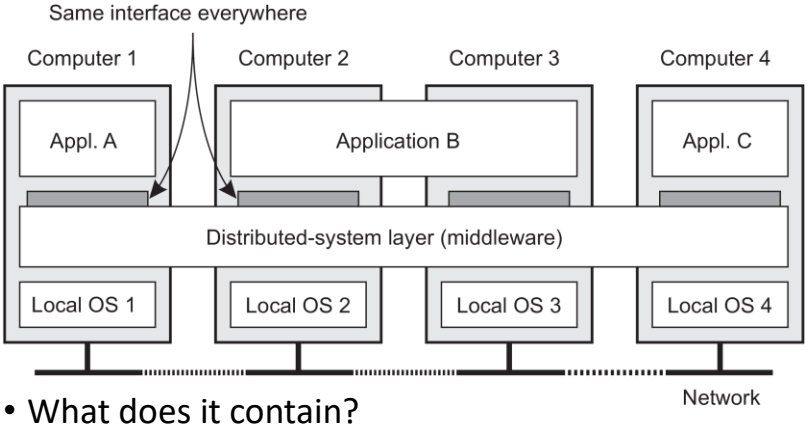
Organization

- Overlay network
  - Each node in the collection communicates only with other nodes in the system, its **neighbors**
  - The set of neighbors my be dynamic, or may be known only implicitly (i.e. requires a lookup)
- Overlay types
  - Well-known example of overlay networks: **peer-to-peer systems**
  - **Structured**: each node has a **well-defined set of neighbors** with whom it can communicate (e.g. tree, ring)
  - **Unstructured**: each node has references to randomly selected other nodes from the system

Characteristic 2: Coherent system

- In essence,
  - The collection of nodes as a whole operates the same, no matter where, when, and how interaction between a user and the system takes place
- Examples:
  - An end user cannot tell where a computation is taking place
  - Where data is exactly stored should be irrelevant to an application
  - If data has been replicated or not is completely hidden
- Keyword: **Distribution transparency**
  - Like approach in many Unix-like OS's, in which resources are accessed through a unifying file-system interface, effectively hiding the differences between files, storage devices, main memory, and networks
- Snag: Partial failures
  - It is inevitable that at any time only a part of the distributed system fails
  - Hiding partial failures and their recovery is often very difficult and in general impossible to hide

1.2.3 Middleware: the OS of distributed systems



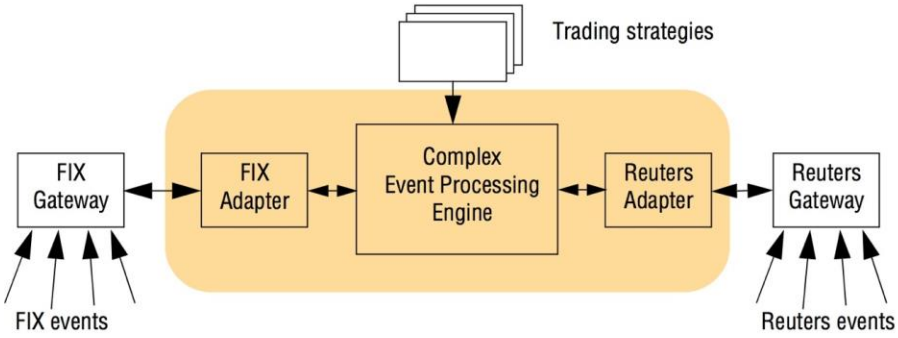
- DS's often organized to have a separate SW layer logically placed on top of the respective OSs of computers that are part of the system

- What does it contain?
  - Commonly used components and functions that need not be implemented by applications separately

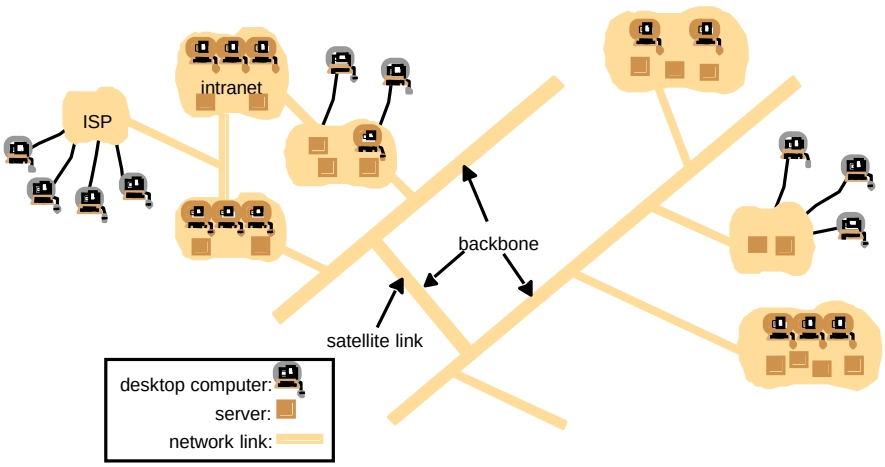
1.2.4 Application domains & associated networked applications

Finance and commerce	eCommerce e.g. Amazon and eBay, PayPal, online banking and trading
The information society	Web information and search engines, ebooks, Wikipedia; social networking: Facebook and MySpace.
Creative industries and entertainment	online gaming, music and film in the home, user-generated content, e.g. YouTube, Flickr
Healthcare	health informatics, on online patient records, monitoring patients
Education	e-learning, virtual learning environments; distance learning
Transport and logistics	GPS in route finding systems, map services: Google Maps, Google Earth
Science	The Grid as an enabling technology for collaboration between scientists
Environmental management	sensor technology to monitor earthquakes, floods or tsunamis

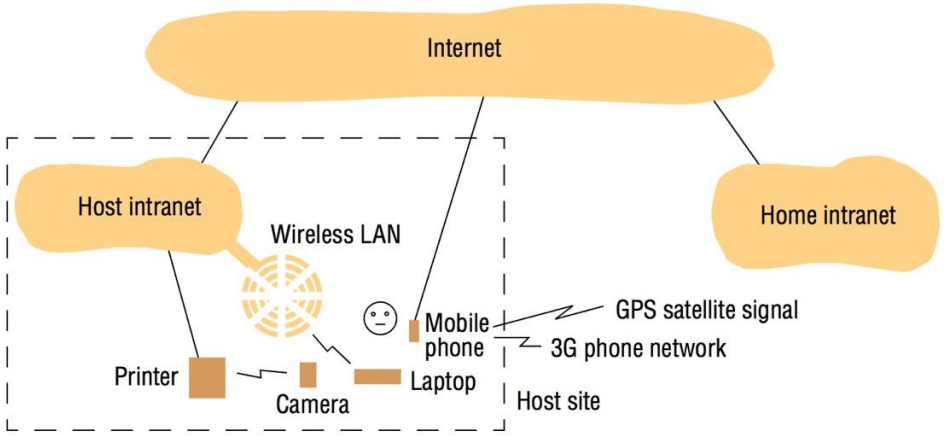
1.2.5 An example financial trading system



1.2.6 A typical portion of the Internet



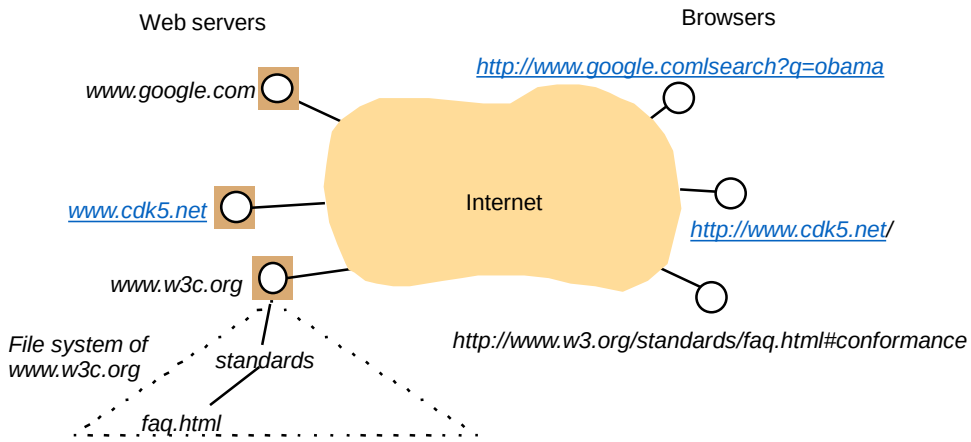
1.2.7 Portable and handheld devices in a DS



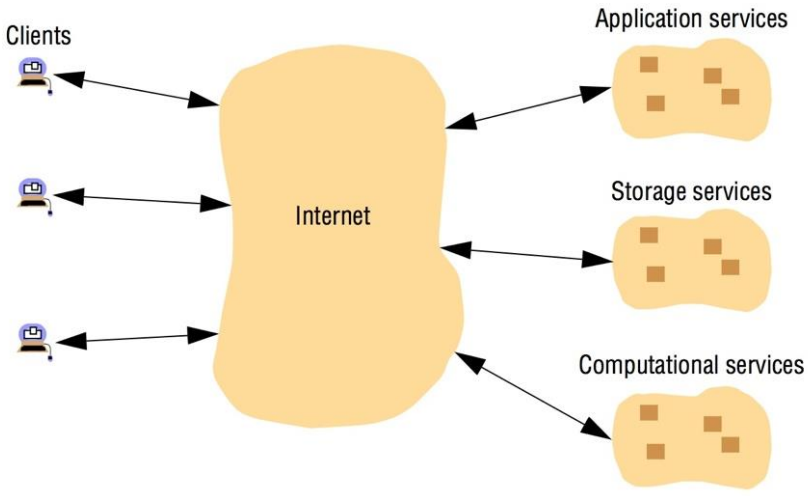
1.2.8 Growth of the Internet (computers & web servers)

Date	*Computers*	*Web servers*	*Percentage*
1993, July	1,776,000	130	0.008
1995, July	6,642,000	23,500	0.4
1997, July	19,540,000	1,203,096	6
1999, July	56,218,000	6,598,697	12
2001, July	125,888,197	31,299,592	25
2003, July	~200,000,000	42,298,371	21
2005, July	353,284,187	67,571,581	19

1.2.9 Web servers and web browsers



1.2.10 Cloud computing



1.2.10 Cloud computing

- Captures distributed computing as a **utility**
- **Cloud:** a set of Internet-based application, storage, and computational services sufficient to support most users' needs
  - Thus enabling them to largely or totally dispense with local data storage and application software
- Promotes a view of everything as a **service**, from physical or virtual infrastructure through to software
  - Often paid for on a per-usage basis, rather than purchased
- Reduces requirements on users' devices
  - Allowing very simple desktop or portable devices to access a potentially wide range of resources and services.

1.3.1 Cloud Symbol

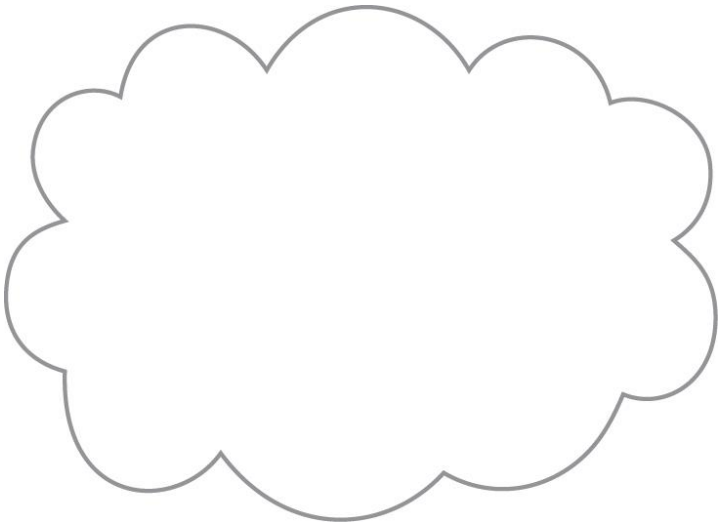


Figure 3.1 The symbol used to denote the boundary of a cloud environment.

1.3.2 Common IT Resources & Symbols

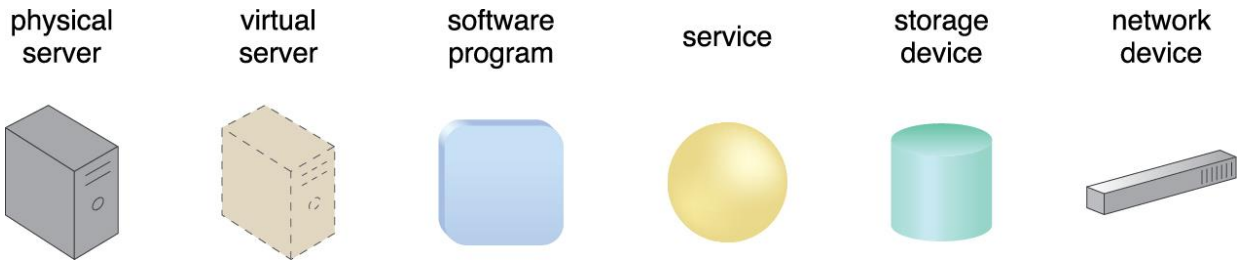


Figure 3.2 Examples of common IT resources and their corresponding symbols.

1.3.3 Example Cloud & IT resources

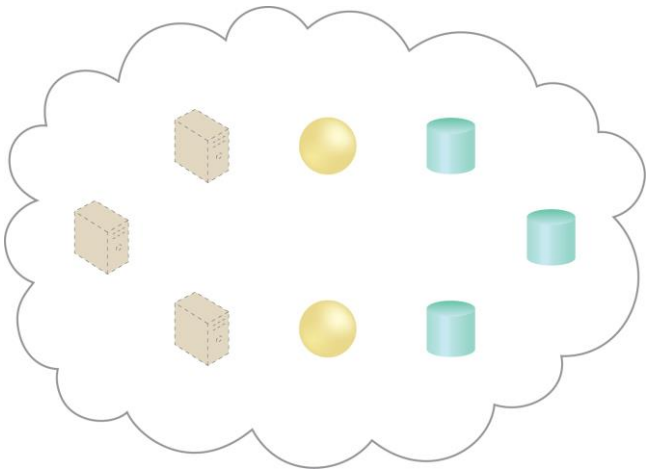


Figure 3.3 A cloud is hosting eight IT resources: three virtual servers, two cloud services, and three storage devices.

1.3.4 IT resource scaling by addition

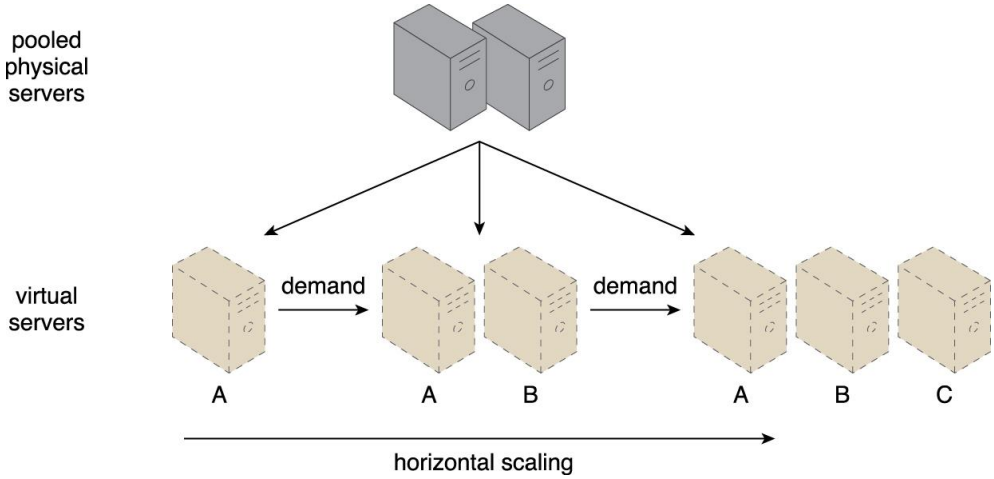


Figure 3.4 An IT resource (Virtual Server A) is scaled out by adding more of the same IT resources (Virtual Servers B and C).

1.3.5 IT resource scaling by replacement

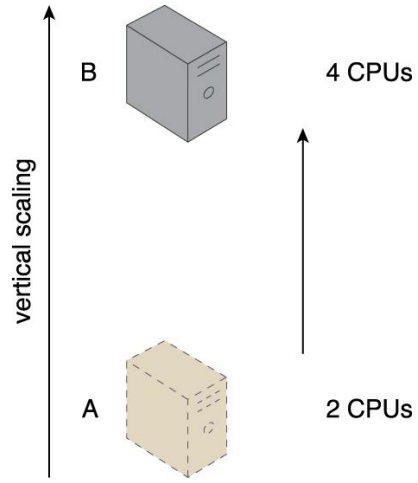


Figure 3.5 An IT resource (a virtual server with two CPUs) is scaled up by replacing it with a more powerful IT resource with increased capacity for data storage (a physical server with four CPUs).

1.3.6 IT resource scaling by replacement

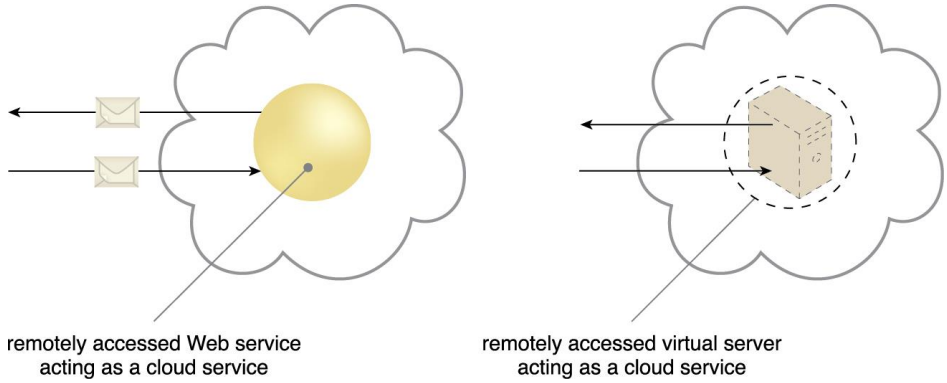


Figure 3.6 A cloud service with a published technical interface is being accessed by a consumer outside of the cloud (left). A cloud service that exists as a virtual server is also being accessed from outside of the cloud's boundary (right). The cloud service on the left is likely being invoked by a consumer program that was designed to access the cloud service's published technical interface. The cloud service on the right may be accessed by a human user that has remotely logged on to the virtual server.

1.3.7 Cloud Service Consumers

Figure 3.7 Examples of cloud service consumers. Depending on the nature of a given diagram, an artifact labeled as a cloud service consumer may be a software program or a hardware device (in which case it is implied that it is running a software program capable of acting as a cloud service consumer).

1.3.8 IT Resource Changing Demands

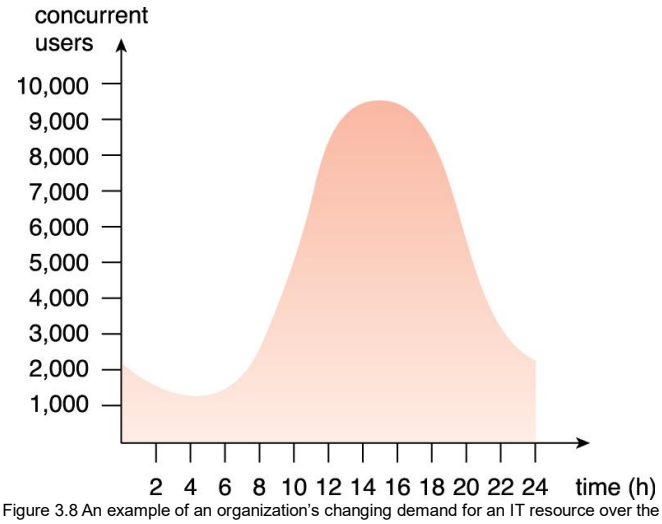


Figure 3.8 An example of an organization's changing demand for an IT resource over the course of a day.

1.3.9 Overlap of Trust Boundaries

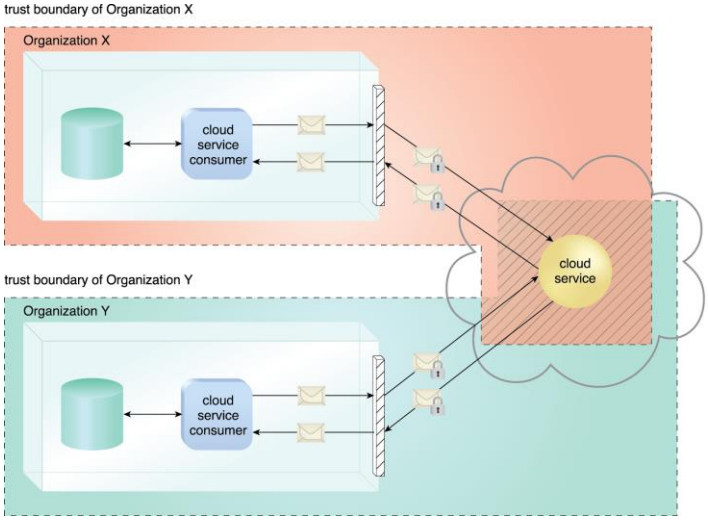


Figure 3.9 The shaded area with diagonal lines indicates the overlap of two organizations' trust boundaries.

1.3.10 Unreliable Network Effects

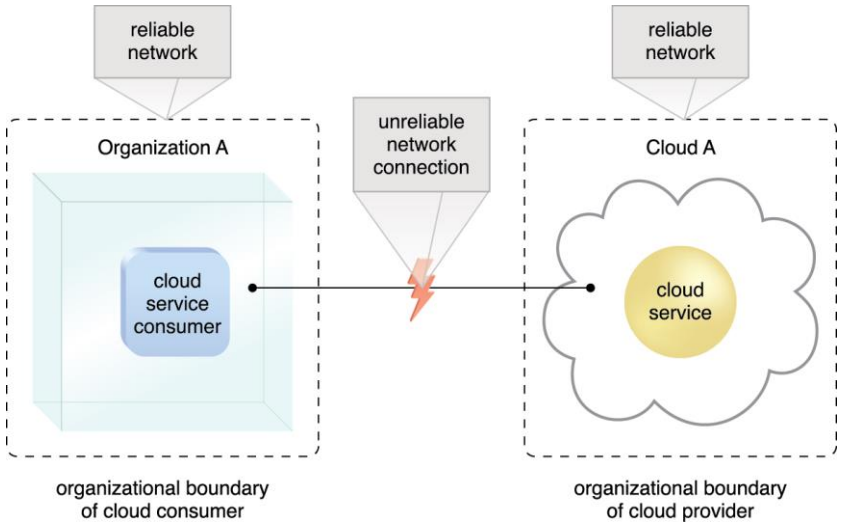


Figure 3.10 An unreliable network connection compromises the quality of communication between cloud consumer and cloud provider environments.

1.4.1 What do DS's want to achieve?

- DS design goals

1. Support sharing of resources
2. Distribution transparency
3. Openness
4. Scalability

1.4.2 Sharing resources

- Canonical examples
  - Cloud-based shared storage and files
  - Peer-to-peer assisted multimedia streaming
  - Shared mail services
    - E.g. outsourced mail systems
  - Shared Web hosting
    - E.g. content distribution networks
- Observation:
  - "*The network is the computer*." [John Gage, while at Sun Microsystems]

1.4.3 Distribution Transparency

Transparency	Description (What is Hidden)
Access	Hide differences in data representation and how an object is accessed
Location	Hide where an object is located
Relocation	Hide that an object may be moved to another location while in use
Migration or **Mobility**	Hide that an object may move to another location
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by several independent users
Failure	Hide the failure and recovery of an object

Transparencies: Alternate & Additional Definitions

- *Access transparency*: enables local and remote resources to be accessed using identical operations.
- *Location transparency*: enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).
- *Concurrency transparency*: enables several processes to operate concurrently using shared resources without interference between them.
- *Replication transparency*: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
- *Failure transparency*: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
- *Mobility transparency*: allows the movement of resources and clients within a system without affecting the operation of users or programs.
- ***Performance transparency*: allows the system to be reconfigured to improve performance as loads vary.**
- ***Scaling transparency*: allows the system and applications to expand in scale without change to the system structure or the application algorithms.**

Degree of transparency

- Observation: Aiming at full distribution transparency may be too much
  - There are communication latencies that cannot be hidden
  - *Completely hiding failures* of networks and nodes is (theoretically and practically) **impossible**
    - You cannot distinguish a slow computer from a failing one
    - You can never be sure that a server actually performed an operation before a crash
  - Full transparency will **cost performance**, exposing distribution of the system
    - Keeping replicas *exactly* up-to-date with the master **takes time**
    - Immediately flushing write operations to disk for fault tolerance

Degree of transparency (cont.)

- Exposing distribution may be good
  - Making use of location-based services (finding nearby friends)
  - When dealing with users in different time zones
  - When it makes it easier for a user to understand what's going on (e.g. when a server does not respond for a long time, report it as failing)
- Conclusion:
  - Distribution transparency is a nice goal, but achieving it is a different story, and often it should not even be aimed for

1.4.3 Openness of distributed systems

- Be able to interact with services from other open systems, irrespective of the underlying environment:
  - Systems should conform to well-defined **interfaces**
  - Systems should easily **interoperate**
  - Systems should support **portability** of applications
  - Systems should be easily **extensible**

Policies vs. Mechanisms

- Implementing Openness: Policies
  - What level of consistency do we require for client-cached data?
  - Which operations do we allow downloaded code to perform?
  - Which QoS requirements do we adjust in the face of varying bandwidth?
  - What level of secrecy do we require for communication?
- Implementing Openness: Mechanisms
  - Allow (dynamic) setting of caching policies
  - Support different levels of trust for mobile code
  - Provide adjustable QoS parameters per data stream
  - Offer different encryption algorithms

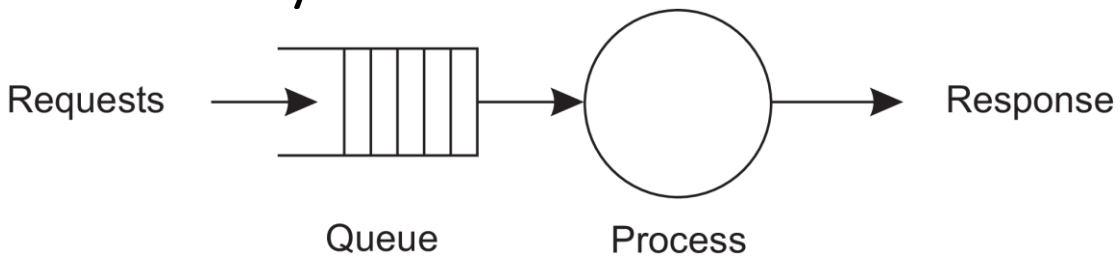
On strict separation

- The stricter the separation between policy and mechanism, the more we need to ensure proper mechanisms, potentially leading to many configuration parameters and complex management
- Finding a balance
  - Hard-coding policies often simplifies management **and** reduces complexity at the price of less flexibility
  - There is no obvious solution

1.4.4 Scaling in distributed systems

- Many developers of modern DS's easily use the adjective "scalable" without making clear **why** their system actually scales
- At least three components
  - Number of users and processes (size scalability)
  - Maximum distance between nodes (geographical scalability)
  - Number of administrative domains (administrative scalability)
- Most systems account only for size scalability, to a certain extent
  - Often a solution: multiple powerful servers operating independently in parallel
  - Today, the challenge still lies in geographical and administrative scalability

Formal analysis



- A centralized service can be modeled as a simple queuing system
- Assumptions & notations:
  - The queue has infinite capacity → arrival rate of requests is not influenced by current queue length or what is being processed
  - Arrival rate requests: λ. Processing capacity service: μ requests per second
- Fraction of time having *k* requests in the system: $p_k = \left(1 - \frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^k$

Formal analysis (cont.)

- Arrival rate requests: λ. Processing capacity service: μ requests per sec
- Fraction of time having *k* requests in the system: $p_k = \left(1 - \frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^k$

- Utilization *U* of a service is the fraction of time that it is busy:

$$U = \sum_{k>0} p_k = 1 - p_0 = \frac{\lambda}{\mu} \Rightarrow p_k = (1-U)U^k$$

- Average number of requests in the system:

$$\bar{N} = \sum_{k\geq 0} k \cdot p_k = \sum_{k\geq 0} k \cdot (1-U)U^k = (1-U)\sum_{k\geq 0} k \cdot U^k = \frac{(1-U)U}{(1-U)^2} = \frac{U}{1-U}$$

- Average throughput: $X = \underbrace{U \cdot \mu}_{\text{Server at work}} + \underbrace{(1-U)\cdot 0}_{\text{Server idle}} = \frac{\lambda}{\mu} \cdot \mu = \lambda$

Formal analysis (cont.)

- Utilization: $U = (1-U)U^k$
- Avg. #requests in system: $\bar{N} = \frac{U}{1-U}$
- Average throughput: $X = \lambda$

- Response time: total time to process a request after submission:

$$R = \frac{\bar{N}}{X} = \frac{S}{1-U} \Rightarrow \frac{R}{S} = \frac{1}{1-U}$$

where $S = 1/\mu$ is the service time

- If *U* is small, response-to-service time is close to 1: a request is immediately processed
- If *U* goes up to 1, the system comes to a grinding halt
  - Solution: decrease *S*

Problems with geographical scalability

- Cannot simply go from LAN to WAN
  - Many distributed systems assume **synchronous client-server interactions**, in which client sends request and waits for an answer
  - **Latency** may easily prohibit this scheme
- WAN links are often inherently **unreliable**
  - Simply moving streaming video from LAN to WAN is bound to fail
- Lack of **multipoint communication**
  - A simple search broadcast cannot be deployed
  - Solution is to develop separate **naming** and **directory services** (each having their own scalability problems)

Problems with administrative scalability

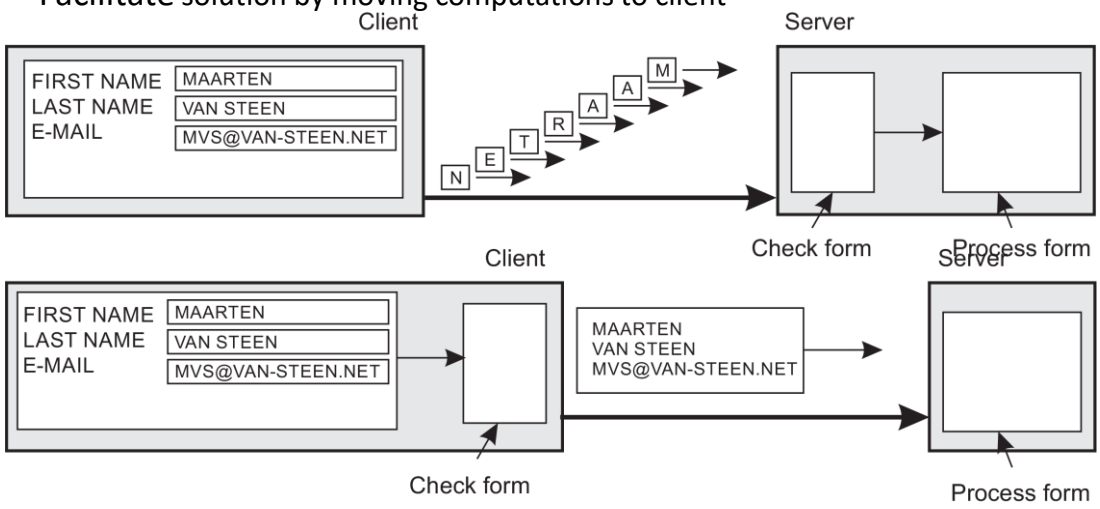
- Conflicting policies concerning usage (and thus payment), management, and security
- Examples:
  - **Computational grids**: share expensive resources between different domains
  - **Shared equipment**: how to control, manage, and use a shared radio telescope constructed as a large-scale shared sensor network
- Exception: several peer-to-peer networks
  - **End users** collaborate, not **administrative entities**
  - Includes File-sharing systems (e.g. BitTorrent), Peer-to-Peer telephony (Skype), and Peer-assisted audio streaming (Spotify)

Techniques for scaling

- Hide communication latencies
  - Make use of **asynchronous communication**
  - Have separate handler for incoming response
  - *Problem*: not every application fits this model
- Partition data and computations across multiple machines
  - Move computations to clients (Java applets)
  - Decentralized naming services (DNS)
  - Decentralized information systems (WWW)

Techniques for scaling (cont.)

- Facilitate solution by moving computations to client



Techniques for scaling (cont.)

- Replication and caching: Make copies of data available at different machines
  - Replicated file servers and databases
  - Mirrored Web sites
  - Web caches (in browsers and proxies)
  - File caching (at server and client)
- Applying replication is easy, except for one problem:
  - Having multiple copies (cached or replicated) leads to **inconsistencies**: modifying one copy makes that copy different from the rest
  - Always keeping copies consistent and in a general way requires **global synchronization** on each modification
  - Global synchronization precludes large-scale solutions
  - If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application-dependent

1.4.5 Pitfalls in developing distributed systems

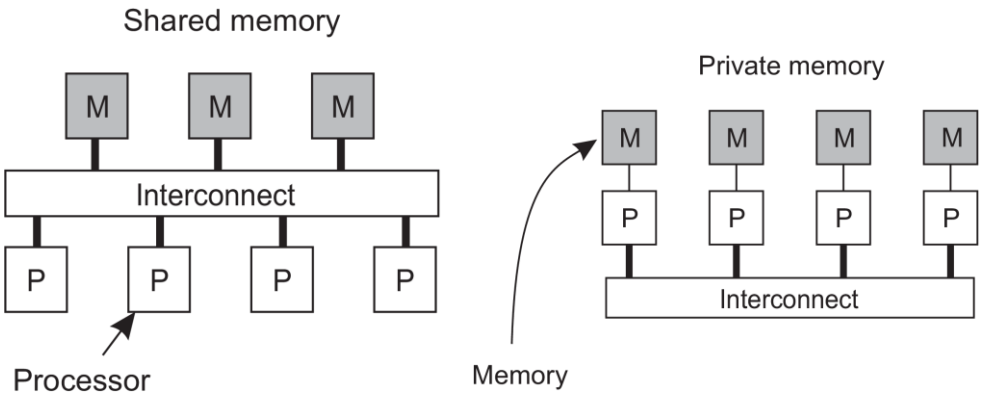
- Many distributed systems are needlessly complex due to mistakes that required patching later on
- Many **false assumptions** (and often hidden) are often made:
  - The network is reliable
  - The network is secure
  - The network is homogeneous
  - The topology does not change
  - Latency is zero
  - Bandwidth is infinite
  - Transport cost is zero
  - There is one administrator

1.5 Three types of distributed systems

1. High-performance distributed computing (HPDC) systems
2. Distributed information systems
3. Distributed systems for pervasive computing

1.5.1 HPDC and Parallel computing

- High-performance distributed computing started with parallel computing
- Multiprocessor and multicore versus multicomputer:

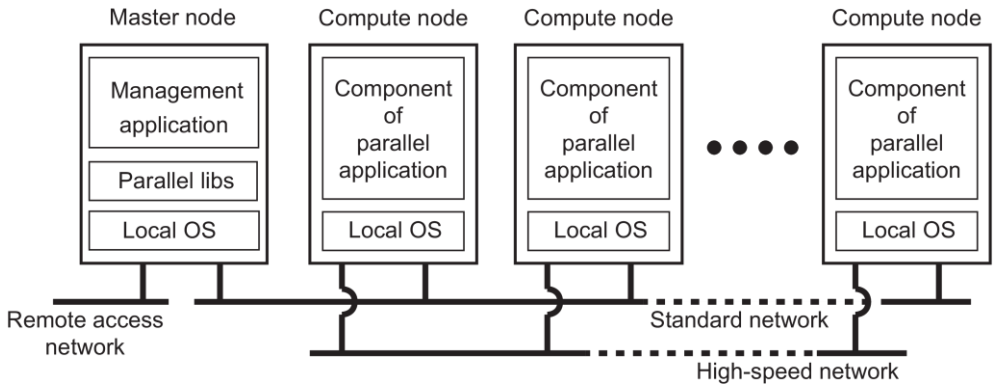


Distributed shared memory systems

- Multiprocessors are relatively easy to program in comparison to multicomputers, yet have problems when increasing the number of processors (or cores)
  - Solution: Try to implement a shared-memory model on top of a multicomputer
- Example through Virtual Memory techniques
  - Map all main-memory pages (from different processors) into one single virtual address space
  - If process at processor *A* addresses a page *P* located at processor *B*, the OS at *A* traps and fetches *P* from *B*, just as it would if *P* had been located on local disk
- Problem: performance of distributed shared memory could never compete with that of multiprocessors
  - Failed to meet expectations of programmers; widely abandoned by now

Cluster computing

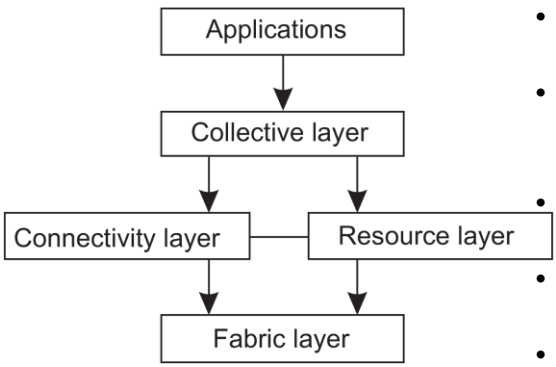
- Essentially a group of high-end systems connected through a LAN
  - Homogeneous: same OS, near-identical hardware
  - Single managing node



Grid computing

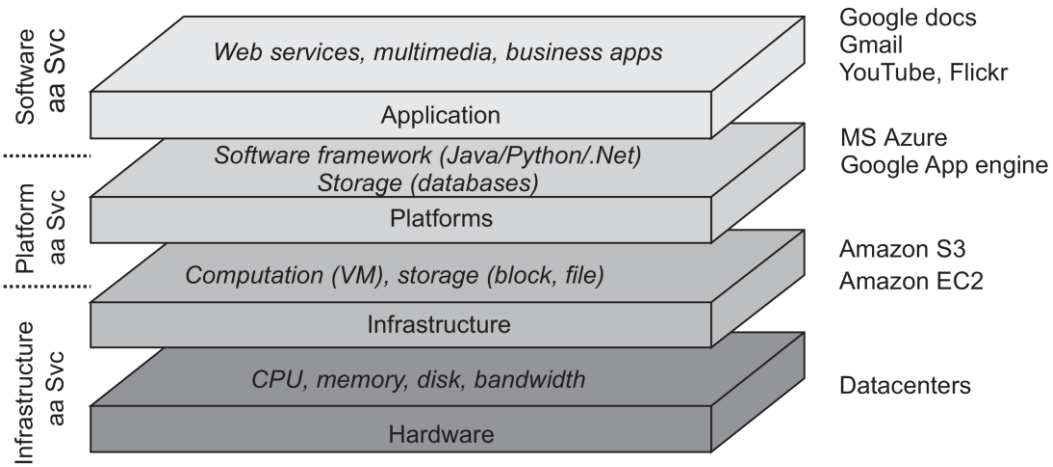
- The next step: lots of nodes from everywhere
  - Heterogeneous
  - Dispersed across several organizations
  - Can easily span a wide-area network
- To allow for collaborations, grids generally use **virtual organizations**
  - This is a grouping of users (or better: their IDs) that will allow for authorization on resource allocation

Architecture for grid computing



- The layers:
  - **Fabric**: provides interfaces to local resources (for querying state & capabilities, locking, etc.)
  - **Connectivity**: communication/ transaction protocols (e.g. for moving data b/w resources); also various authentication protocols
  - **Resource**: manages a single resource, such as creating processes or reading data
  - **Collective**: handles access to multiple resources: discovery, scheduling, replication
  - **Application**: contains actual grid applications in a single organization

Cloud computing



Cloud computing (cont.)

- Make a distinction between four layers:
  - **Hardware**: Processors, routers, power, and cooling systems; customers normally never get to see these
  - **Infrastructure**: Deploys virtualization techniques; evolves around allocating and managing virtual storage devices and virtual servers
  - **Platform**: Provides higher-level abstractions for storage and such; for example, Amazon S3 storage system offers an API for (locally created) files to be organized and stored in so-called **buckets**
  - **Application**: Actual applications, such as office suites (text processors, spreadsheet applications, presentation applications); comparable to the suite of apps shipped with OS's

Is cloud computing cost-effective?

- An important reason for the success of cloud computing is that it allows organizations to **outsource** their IT infrastructure: HW & SW
- *Is outsourcing also cheaper?*
- Consider **enterprise applications**, modeled as a collection of components, each component C_i requiring N_i servers
- Application now becomes a **directed graph**, with a vertex representing a component, and an arc $\langle \overrightarrow{i,j} \rangle$ representing data flowing from C_i to C_j
- Two associated weights per arc:
  - $T_{i,j}$ is # transactions per time unit that causes a data flow from C_i to C_j
  - $S_{i,j}$ is total amount of data associated with $T_{i,j}$

Is cloud computing cost-effective? (cont.)

- Migration plan:
  - Figure out for each component C_i, how many n_i of its N_i servers should migrate, such that the monetary benefits reduced by additional costs for Internet communication are maximal
- Requirements migration plan:

1. Policy constraints are met
2. Additional latencies do not violate specific delay constraints
3. All transactions continue to operate correctly; requests or data are not lost during a transaction

Computing benefits & Internet costs

- Computing benefits: Monetary savings:
  - B_c: benefits of migrating a compute-intensive component
  - M_c: total number of migrated compute-intensive components
  - B_s: benefits of migrating a storage-intensive component
  - M_s: total number of migrated storage-intensive components
- Total benefits are: $B_c \cdot M_c + B_s \cdot M_s$
- Internet costs: Traffic to/from the cloud:

$$Tr_{local,inet} = \sum_{C_i} (T_{user,i} S_{user,i} + T_{i,user} S_{i,user})$$

  - $T_{user,i}$: transaction per time unit causing data flow from user to C_i
  - $S_{user,i}$: amount of data associated with $T_{user,i}$

Rate of transactions after migration

- $C_{i,local}$: set of servers of C_i that continue locally
- $C_{i,cloud}$: set of servers of C_i that are placed in the cloud
- Assume traffic distribution is the same for local and cloud server
- Note that $|C_{i,cloud}| = n_i$. Let $f_i = \frac{n_i}{N_i}$, and s_i be a server of cloud C_i

$$T^*_{i,j} = \begin{cases} (1-f_i) \cdot (1-f_j) \cdot T_{i,j} & \text{when } s_i \in C_{i,local} \text{ and } s_j \in C_{j,local} \\ (1-f_i) \cdot f_j \cdot T_{i,j} & \text{when } s_i \in C_{i,local} \text{ and } s_j \in C_{j,cloud} \\ f_i \cdot (1-f_j) \cdot T_{i,j} & \text{when } s_i \in C_{i,cloud} \text{ and } s_j \in C_{j,local} \\ f_i \cdot f_j \cdot T_{i,j} & \text{when } s_i \in C_{i,cloud} \text{ and } s_j \in C_{j,cloud} \end{cases}$$

Overall Internet costs

- $cost_{local,inet}$: per unit Internet costs to local part
- $cost_{cloud,inet}$: per unit Internet costs to cloud
- Assume traffic distribution is the same for local and cloud server
- Costs and traffic before and after migration:

$$Tr^*_{local,inet} = \sum_{C_{i,local},C_{j,local}} \left(T^*_{i,j}S^*_{i,j} + T^*_{j,i}S^*_{j,i}\right) + \sum_{C_{j,local}} \left(T^*_{user,j}S^*_{user,j} + T^*_{j,user}S^*_{j,user}\right)$$

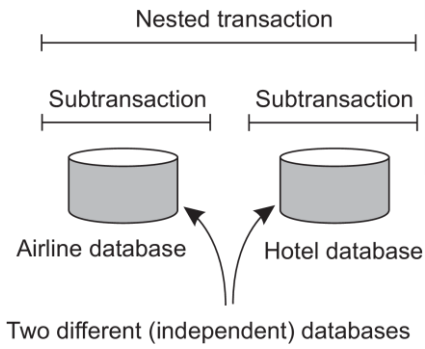
$$Tr^*_{cloud,inet} = \sum_{C_{i,cloud},C_{j,cloud}} \left(T^*_{i,j}S^*_{i,j} + T^*_{j,i}S^*_{j,i}\right) + \sum_{C_{j,cloud}} \left(T^*_{user,j}S^*_{user,j} + T^*_{j,user}S^*_{j,user}\right)$$

$$cost_{total} = cost_{local,inet}\left(Tr^*_{local,inet} - Tr_{local,inet}\right) + cost_{cloud,inet}Tr^*_{cloud,inet}$$

1.5.2 Distributed Information Systems: Integrating applications

- Organizations confronted with many **networked applications**, but achieving interoperability was painful
- Basic approach:
  - A **networked application** is one that runs on a **server** making its services available to remote clients
- Simple integration:
  - Clients (1) Combine requests for (different) applications; (2) Send that off; (3) Collect responses, and (4) Present a coherent result to the user
  - At lowest level, Integration lets clients wrap multiple requests, possibly for different servers, into a single larger request and have it executed as a **distributed transaction**
- Next step:
  - Allow direct application-to-application communication, leading to **Enterprise Application Integration (EAI)**

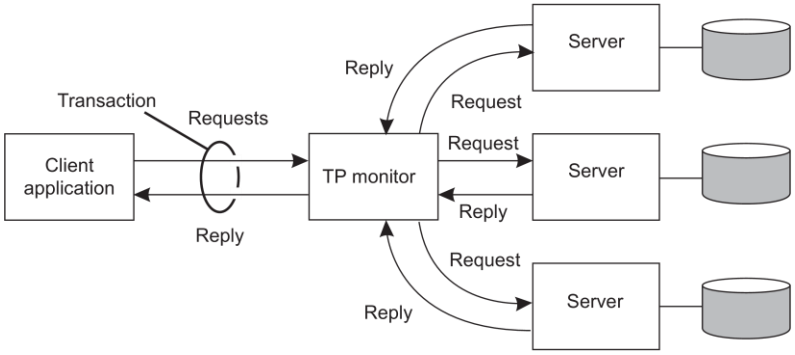
Example: EAI (nested) transactions



Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

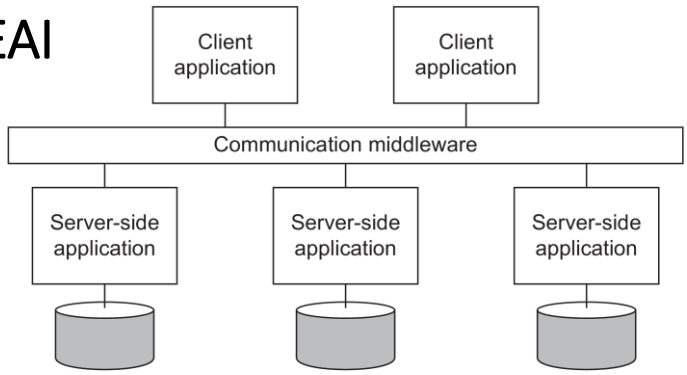
- Issue: all-or-nothing
  - **Atomic**: happens indivisibly (seemingly)
  - **Consistent**: does not violate system invariants
  - **Isolated**: not mutual interference
  - **Durable**: commit means changes are permanent

Transaction Processing Monitor (TPM)



- Often, data involved in a transaction is distributed across several servers
- A **TP Monitor** is responsible for coordinating the execution of a transaction

Middleware and EAI



- Middleware offers communication facilities for integration
- Remote Procedure Call (RPC): Requests are sent through local procedure call, packaged as a message, responded through message, and result returned as return from call
- Message Oriented Middleware (MOM): Messages are sent to logical contact point (published), and forwarded to subscribed applications

How to integrate applications

- File transfer: Technically simple, but not flexible
  - Figure out file format and layout
  - Figure out file management
  - Update propagation, and update notifications
- Shared database:
  - Much more flexible
  - But still requires common data scheme next to risk of bottleneck
- Remote procedure call (RPC):
  - Effective when execution of a series of actions is needed
  - But requires caller and callee to be up and running at the same time
- Messaging:
  - Does not require both running
  - Messaging allows decoupling in time and space

1.5.3 Distributed pervasive systems

- Emerging next generation of DS's in which nodes are small, mobile, and often **embedded** in a larger system, characterized by the fact that the system *naturally blends into the user's environment*
- Three (overlapping) subtypes
  1. **Ubiquitous computing systems**: pervasive and *continuously present* (i.e. there is a continuous interaction between system and user)
  2. **Mobile computing systems**: pervasive, but emphasis is on the fact that devices are *inherently mobile*
  3. **Sensor (and actuator) networks**: pervasive, with emphasis on the actual (collaborative) *sensing* and *actuation* of the environment

Ubiquitous systems

- Core elements:
  1. **Distribution**: Devices are networked, distributed, and accessible in a transparent manner
  2. **Interaction**: Interaction between users and devices is highly unobtrusive
  3. **Context awareness**: System is aware of a user's context to optimize interaction
  4. **Autonomy**: Devices operate autonomously without human intervention, and are therefore highly self-managed
  5. **Intelligence**: The system as a whole can handle a wide range of dynamic actions and interactions

Mobile computing

- Distinctive features:
  - A myriad of different mobile devices:
    - Smartphones
    - Tablets
    - GPS devices
    - Remote controls
    - Active badges
  - *Mobile* implies that a device's location is expected to change over time, thus requiring a change of local services, reachability, etc. Keyword: **discovery**
  - Communication may become more difficult, due to no stable route, but also perhaps no guaranteed connectivity: **disruption-tolerant networking**

Mobility patterns

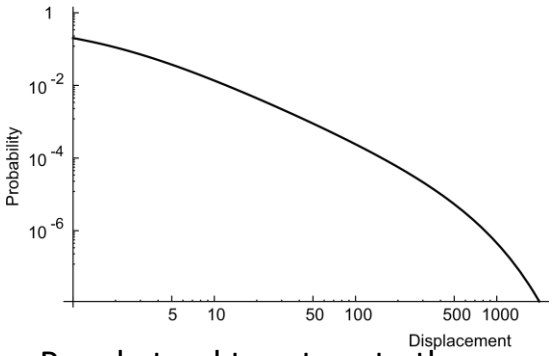
- What is the relationship between information dissemination and human mobility?
  - **Basic idea**: an encounter allows for the exchange of information (**packet-switched networks**)
- A successful strategy:
  - Alice's world consists of friends and strangers
  - If Alice wants to get a message to Bob, hand it out to all her friends
  - Friend passes message to Bob at first encounter
- This strategy works because (apparently) there are relatively closed communities of friends

Community detection

- How to detect your community without having global knowledge?
- Gradually build your list:
  1. Node *i* maintains a **familiar set** F_i and a **community set** C_i, initially both empty
  2. Node *i* adds *j* to C_i when $\frac{|F_j \cap C_i|}{|F_j|} > \lambda$
  3. Merge two communities when $|C_i \cap C_j| > \gamma |C_i \cup C_j|$
- Experiments show that values of $\lambda = \gamma = 0.6$ are reasonable

How mobile are people?

- Experimental results: Tracing 100k cellphone users for 6 months:



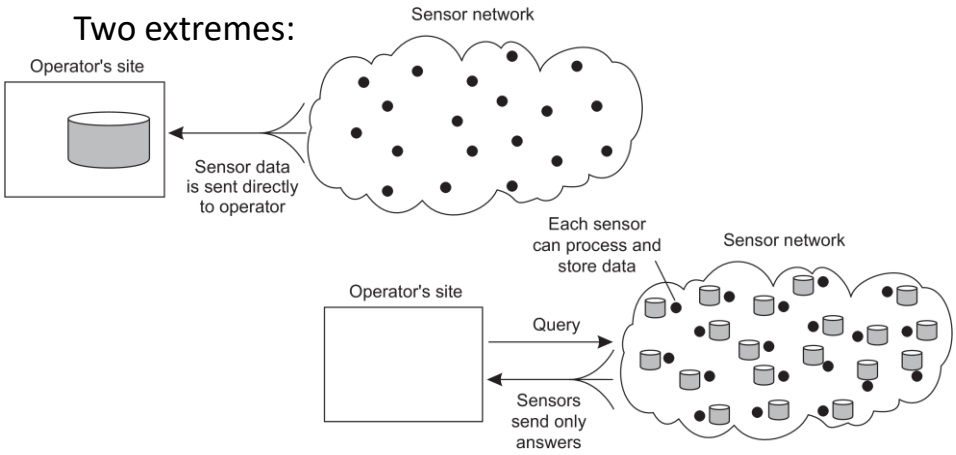
- People tend to return to the same place after 24, 48, or 72 hours
  - Thus, we're not really that mobile

Sensor networks

- The nodes to which sensors are attached are:
  - Many (10s-1000s)
  - Simple:
    - Small memory
    - Low compute resources
    - Basic communication capacity
  - Often battery-powered (or even battery-less)

Sensor networks as distributed databases

Two extremes:



Duty-cycled networks

- Issue: many sensor networks need to operate on a strict energy budget
  - Solution: Introduce **duty cycles**
- A node is **active** during T_{active} time units, and then **suspended** for $T_{suspended}$ time units, before it becomes active again
- Thus, its **Duty cycle τ** is:

$$\tau = \frac{T_{active}}{T_{active} + T_{suspended}}$$

- Typical duty cycles are 10 – 30%, but can also be lower than 1%

Keeping duty-cycled networks in sync

- If duty cycles are low, sensor nodes may not wake up at the same time anymore and become permanently **disconnected**
  - When disconnected, they are active during different, nonoverlapping slots
- Solution:
  - Each node *A* adopts a **cluster ID** C_A, where C_A is an integer
  - During its suspended period, node sends a **join message**
  - When *A* receives a join message from *B*, and $C_A < C_B$, it sends a join message to its neighbors (in cluster C_A) before joining *B*
  - When $C_A > C_B$, it sends a join message to *B* during *B*'s active period
- Once a join message reaches a whole cluster, merging 2 clusters is very fast
  - Merging means nodes must ***readjust clocks***

Ongoing Research: A Smart Campus Framework

- Louisiana Tech University is striving to undertake changes to improve its capacity to react to:
  - Natural disasters: hurricanes, floods, tornados
  - Water and Air pollution
  - Agricultural phenomena
  - Active shooter situations
- The College of Engineering and Science (Cyber Eng. dept. in particular) proposes to effect this using IoT devices and wireless sensor networks

89

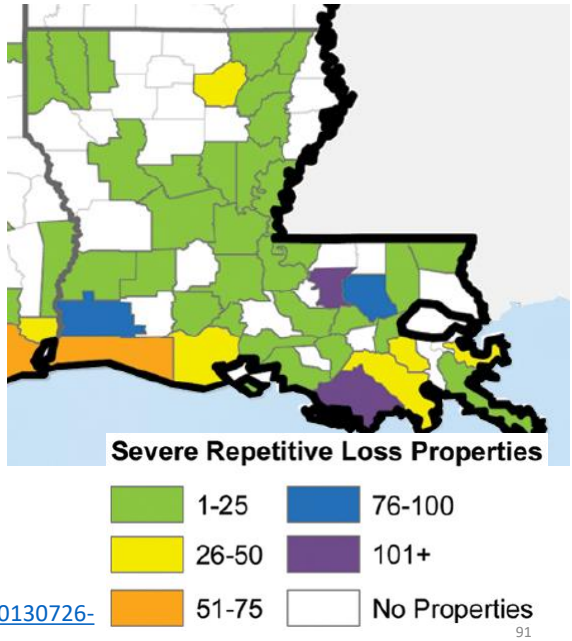
Motivation: Hurricanes

- In August 2017, Hurricane Harvey struck Texas and Louisiana shores
  - In Texas, over 300,000 people were left without electricity and billions of dollars of property damage was sustained
  - 88 fatalities were confirmed, most of which were drownings
  - Approximately 13,000 people had been rescued across the state while an estimated 30,000 were displaced
- In September 2017, Hurricane Irma affected areas of Florida, Georgia, and South Carolina, causing 90 deaths and at least $50 billion in damages
- These natural disasters have caused devastating damage to the environment
  - In the aftermath of Hurricane Harvey, two ExxonMobil refineries had to be shut down following related storm damage and releases of hazardous pollutants
  - Two oil storage tanks owned by Burlington Resources Oil and Gas collectively spilled 30,000 gallons of crude in DeWitt County
  - An additional 8,500 gallons of wastewater was spilled in the incidents
- The impact of events like these on the human population living in surrounding neighborhoods cannot be ignored
  - Measures must be taken to minimize human exposure to poisonous chemicals in the air and water, especially combustible variety, and ensure cleanliness of the potable water supply
  - In the presence of fires, ensuring that the local populace is not exposed to harmful gases is also a priority

90

Motivation: Floods

- Louisiana leads the country in properties that are repeatedly swamped by water and rebuilt using flood insurance payments
- State residents received $1.2 billion since 1978 for damage to houses & businesses that have flooded multiple times
- The Federal Emergency Management Agency (FEMA) recognizes Severe Repetitive Loss (SRL) areas in LA here:

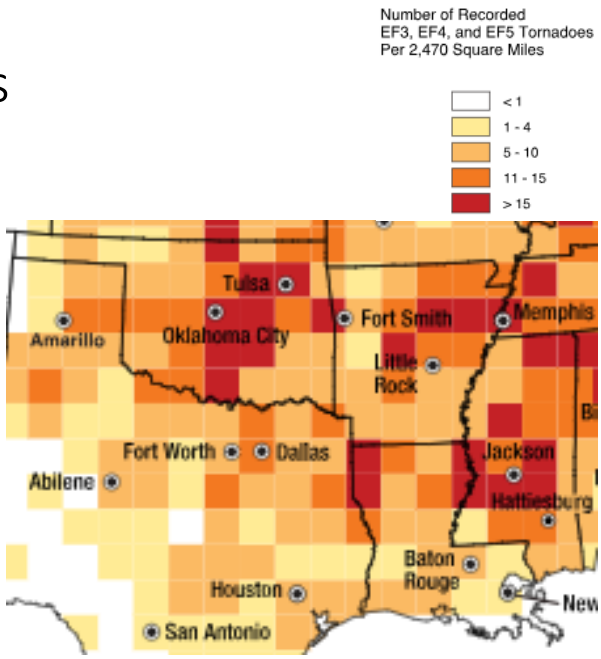


Source: https://www.fema.gov/media-library-data/20130726-1709-25045-4851/2_severerepetetiveloss.pdf

91

Motivation: Tornadoes

- Tornado alley refers to an area of the US where tornadoes are most frequent
- On April 13-14, Northern LA experienced storms that spawned 14 tornados in 8 hours[9]
- Residents notified via their Smartphones using Wireless Emergency Alerts (WEA)
  - But vague messages gave alerts to several parishes all at once, and residents had no inkling as to where the tornados actually were



Source: https://en.wikipedia.org/wiki/Tornado_Alley

92

Motivation: Gun Rights and School Shootings

Date	Location	Deaths	Injuries	Description
April 20, 1999	Littleton, Colorado	15	21	Columbine High School massacre
March 21, 2005	Red Lake, Minnesota	10	7	Red Lake shootings
October 2, 2006	Nickel Mines, PA	6	3	West Nickel Mines School shooting
April 16, 2007	Blacksburg, Virginia	33	23	Virginia Tech School shooting
February 14, 2008	DeKalb, Illinois	6	21	Northern Illinois University shooting
April 2, 2012	Oakland, CA	7	3	Oikos University shooting
December 14, 2012	Newtown, Connecticut	28	2	Sandy Hook Elementary School shooting
June 7, 2013	Santa Monica, CA	6	4	2013 Santa Monica shooting
October 1, 2015	Roseburg, Oregon	10	9	Umpqua Community College shooting
November 14, 2017	Rancho Tehama, CA	6	18	Rancho Tehama Reserve shooting
February 14, 2018	Parkland, Florida	17	17	Marjory Stoneman Douglas High School
May 18, 2018	Santa Fe, Texas	10	13	Santa Fee High School

Source: https://en.wikipedia.org/wiki/List_of_school_shootings_in_the_United_States

93

Motivation: Smart Agriculture (cont.)

- Climatic factors such as sunlight, rainfall and wind also affects the qualitative characteristics of fruit such as shape and color of the peach produced [5], [6]
- If these environmental and climatic factors are not monitored properly, these will have detrimental effects on the peach fruit production and may even damage the whole peach tree
- These factors call for an automated solution to take care of many parameters involved and Smart Agriculture is a solution for it
- Smart Agriculture uses Wireless Sensor Networks (WSN), the Internet of Things (IoT), and cloud storage to help farmers make more intelligent decisions about optimal time for irrigating, pruning, and harvesting based on collected data
  - Will eventually help in stable production of peach fruits that will be ecologically competitive
  - Can predict the probability of the near frost event by analyzing the real time readings of the collected data for soil temperature, soil moisture, air temperature and relative humidity
- Also be able to get good market value for their crops because low chill peach cultivars produce fruit earlier since they start blooming earlier than the high chill peach cultivars

[5] T. W. Wert, J. G. Williamson, J. X. Chaparro, E. P. Miller, and R. E. Rouse, "The influence of climate on fruit shape of four low-chill peach cultivars," *HortScience*, vol. 42, no. 7, pp. 1589–1591, 2007.
[6] ——, "The influence of climate on fruit development and quality of four low-chill peach cultivars," *HortScience*, vol. 44, no. 3, pp. 666–670, 2009.

94

Chapter 1 Summary

- DS's must combine autonomous nodes, but appear as a single, coherent system
  - Requires a layer of middleware much like the OS to a computer
- Many relevant modern applications
- Use common notation & symbols for clouds and their resources
- Certain concepts and goals are common to all DS's
  - Shared resources
  - Distribution transparency
  - Openness
  - Scalability
- Types of DS's
  - HPDC
  - Distributed Info Systems
  - Pervasive Systems