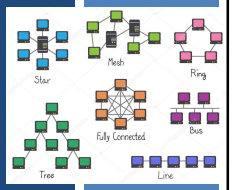
TCSS 558: APPLIED DISTRIBUTED COMPUTING

Introduction - II

Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma



1

DEMOGRAPHICS SURVEY

SURVEY LINK AT:
http://faculty.washington.edu/wlloyd/courses/tcss558/announcements.html

2

OBJECTIVES

- Course demographics survey
- Chapter 1 - What is a distributed system?
- Design goals of distributed systems:
  - Resource sharing / availability
  - Distribution transparency
  - Openness
  - Scalability
- Activity: Design goals of distributed systems (1/9)
- Research directions

3

FEEDBACK FROM 1/7

- **Daily Course Feedback**: 25 respondents
- Perspective on material from class:
  - 6.68 (Equal New and Review (5) → **Mostly New to Me (10)**)
- Pace of class:
  - 5.58 (Just Right (5) – Fast (10))
- **Do we get to choose partners for project(s)?**
  - *Yes*
  - *Also option to work independently*
- **Will we have the same members for project(s)?**
  - If so chosen

4

UNCLEAR POINTS:

- **Location transparency**
  - Users can not tell where an object is physically located
  - The server location is ABSTRACTED from users
  - URLs and URIs help do this: these are **logical** names
- **Relocation transparency**
  - Entire website or server may be moved by distributed system at any time
  - Movement may be related to server maintenance, etc.
  - Users should not be able to tell

5

FEEDBACK - 3

- **Migration transparency**
  - Feature of distributed systems: support for mobile processes
  - A user process begins on one server, migrates to another to reduce network latency
  - Example: video/audio stream to mobile device
  - live audio stream initially streams from one server, and is switched to another as mobile device moves
  - i.e. driving from Seattle to Portland on Interstate 5
  - Process migrates to minimize latency over changing networks

6

FEEDBACK - 4

- **Failure transparency**
  - A server (or entire distributed system) that provides a (web)service to a client fails while actively in use
  - The user does not notice, but continues to use the service
  - The system recovers from failure with no loss of data
  - Performance loss is ok – user does not identify the slowdown as being associated with a failure
  - User maintains confidence in the distributed system (or service)

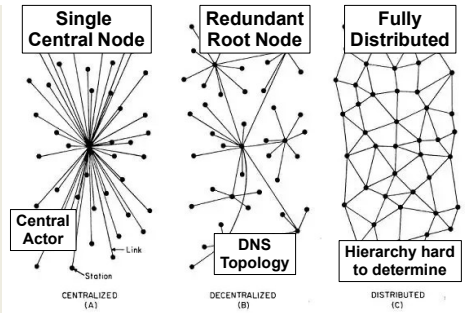
7

FAILURE TRANSPARENCY

- **Failure recovery:**
  - Arguably the most difficult to provide
  - Distributed systems with replication are designed to be able to withstand the loss 1 or more nodes
  - Consensus protocols (Ch. 8) are used to converse among remaining nodes to determine which has the most up-to-date version of the data
  - New nodes can replace failed nodes, and data is replicated to them

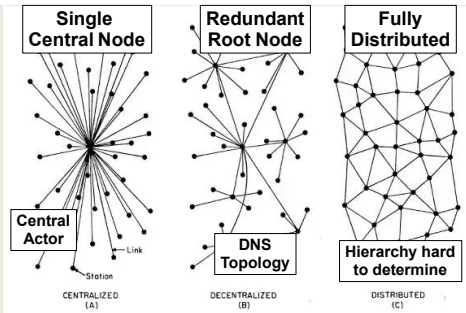
8

SYSTEM ARCHITECTURES



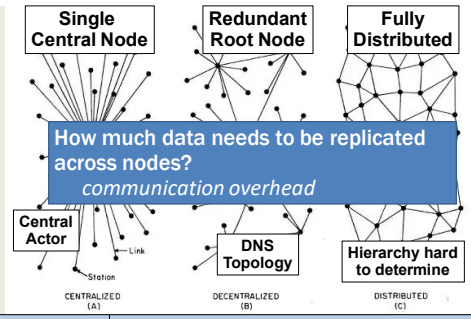
9

Consider implications for:
- State tracking
- Membership tracking
- Authentication



10

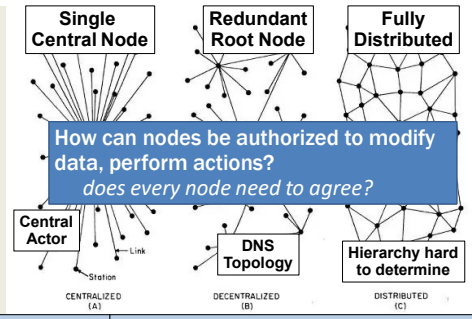
Consider implications for:
- State tracking
- Membership tracking
- Authentication



How much data needs to be replicated across nodes?
communication overhead

11

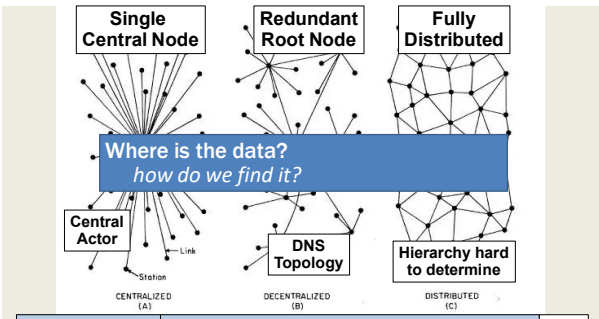
Consider implications for:
- State tracking
- Membership tracking
- Authentication



How can nodes be authorized to modify data, perform actions?
does every node need to agree?

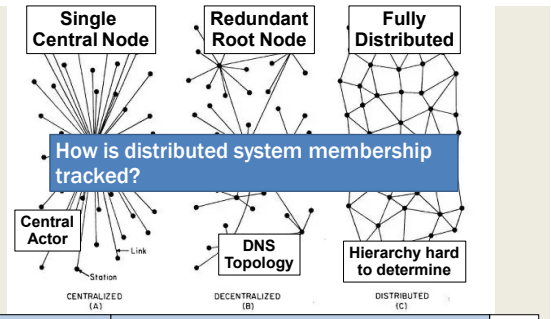
12

Consider implications for:
- State tracking
- Membership tracking
- Authentication



13

Consider implications for:
- State tracking
- Membership tracking
- Authentication



14

WHAT IS A DISTRIBUTED SYSTEM?

- Definition:
- A ***collection of autonomous computing elements*** that appears to users as a single coherent system.
- How nodes collaborate / communicate is **key**
- Nodes
  - Autonomous computing elements
  - Implemented as hardware or software processes
- Single coherent system
  - Users and applications perceive a single system
  - Nodes collaborate, and provide "abstraction"

15

CHARACTERISTICS OF DISTRIBUTED SYSTEMS - 1

- **#1: Collection of autonomous computing elements**
  - Node synchronization
    - *data replication, transactional states*
  - Node coordination
    - *group membership, authentication*
  - Overlay networks – enable node connectivity
    - *communication*
- **#2: Single coherent system**

16

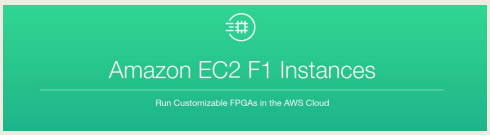
DESIGN GOALS OF DISTRIBUTED SYSTEMS

- **Accessibility**: support for sharing resources
- **Distribution transparency**
- **Openness**: avoiding vendor lock-in
- **Scalability**

17

ACCESSIBILITY: RESOURCE SHARING

- Easy for users (and applications) to share remote resources
  - Storage, compute, networks, services, peripherals, …
  - Field programmable arrays (FPGAs) "as a service":



  - https://aws.amazon.com/ec2/instance-types/f1/
  - Nearly any resource can be shared

18

DISTRIBUTION TRANSPARENCY

- In distributed systems, aspects of the implementation are hidden from users
- End users can simply use / consume the resource (or system) without worrying about the implementation details
- Technology aspects required to implement the distribution are abstracted from end users
- **The distribution is transparent to end users.**
- End users are not aware of certain mechanisms that do not appear in the distributed system because transparency confines details into layer(s) below the one users interact with. *(abstraction through layered architectures)*
- Users perceive the system as a single entity even though it's implementation is spread across a collection of devices.

19

DISTRIBUTION TRANSPARENCY - 2

- Types of distribution transparency
- Object is a resource or a process

Transparency	Description
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	Hide that an object may move to another location
Replication	Hide that an object is replicated
Concurrency	Hide than an object may be shared by several independent users
Failure	Hide the failure and recovery of an object

20

DISTRIBUTION TRANSPARENCY - 3

- Why would we want **location transparency**?
  - Uniform resource locator (URL) …
  - Where is it?
- **Relocation transparency:**
- Cloud application is moved from one server to another
- Initiated by the distributed system, possibly for maintenance
- Users should not notice
- **Migration transparency:**
- Feature offered by distributed systems
- User processes may move to new servers with no loss of availability
  - e.g. mobile phone client streaming audio while driving on highway
  - Server providing live stream audio to client changes to minimize latency

21

DISTRIBUTION TRANSPARENCY - 4

- **Replication transparency:**
- Hide the fact that several copies of a resource exist
- What if a user is aware of, or has to interact with the copies?
- **Reasons for replication:**
- Increase availability
- Improve performance
- Fault tolerance: a replica can take over when another fails

22

DISTRIBUTION TRANSPARENCY - 5

- **Concurrency transparency:**
- Concurrent use of ***any*** resource requires synchronization via locking
- Transactions can be used
- **Failure transparency:**
- Masking failures is one of the hardest issues in dist. systems
- How do we tell the difference between a failed process and a very slow one?
- When do we need to "fail over" to a replica?
- Subject of chapter 8…

23

DEGREES OF DISTRIBUTION TRANSPARENCY

- Full distribution transparency may be impractical
- Communication latencies cannot be hidden
- Completely hiding failures of networks and nodes is impossible
  - Difference between slow computer and failing one
  - Transactions: did operation complete before crash?
- Full transparency will lead to slower performance:
  - Performance vs. transparency tradeoff
- Synchronizing replicas with a master requires time
- Immediately commit writes in fear of device failure

24

DEGREES OF DISTRIBUTION TRANSPARENCY - 2

- Abstracting location when user desires to interact intentionally with local resources / systems
- **Exposing** the distribution may be good:
  - Location-based-services (find nearby friends)
  - Help a user understand what's going on
  - When a server doesn't respond for a long time – is it far away?
  - Users in different times zones?
- Can you think of examples where distribution is not hidden?
  - Eventual consistency
  - Many online systems no longer update instantaneously
  - Users are getting accustomed to delays

25

OPENNESS

- System with components that are easily used by, or integrated into other systems
- **Key aspects of openness:**
- Interoperability, portability, extensibility
- **Interfaces**: provide general syntax and semantics to interact with distributed components
- Services expose interfaces: functions, parameters, return values
- Semantics: describe what the services do
  - Often informally specified (via documentation)
- General interfaces enable alternate component implementations

26

OPENNESS - 2

- **Interoperability**: ability for components from separate systems to work together (different vendors?)
- Though implementation of a common interface
- How could we measure interoperability of components?
- **Portability**: degree that an application developed for distributed system A can be executed without modification on distributed system B
- How could we evaluate portability of a component?
- What percentage of portability is expected?

27

OPENNESS - 3

- **Extensible**: easy to reconfigure, add, remove, replace components from different developers
- Example: replace the underlying file system of a distributed system
- To be open, we would like to ***separate policy from mechanism***
- Policy may change
- Mechanism is the technological implementation
- Avoid coupling policy and mechanism
- Enables flexibility
- Similar to separation of concerns, modular/OO design principle

28

SEPARATING POLICY FROM MECHANISM

- Example: **web browser caching**
- **Mechanism:** browser provides facility for storing documents
- **Policy:** Users decide which documents, for how long, …
- Goal: Enable users to set policies dynamically
- For example: browser may allow separate component plugin to specify policies
- **Tradeoff**: management complexity vs. policy flexibility
- Static policies are inflexible, but are easy to manage as features are barely revealed.
- AWS Lambda (Function-as-a-Service) abstracts configuration polices from the user resulting in management simplicity

29

OPENNESS EXAMPLE

- **Which of the following designs is more open?**
- Acme software corporation hosts a set of public weather web services (e.g. web service API)
- **DESIGN A:** API is implemented using MS .NET Remoting
- .NET Remoting is a mechanism for communicating between objects which are not in the same process. It is a generic system for different applications to communicate with one another. .NET objects are exposed to remote processes, thus allowing inter process communication. The applications can be located on the same computer, different computers on the same network, or on computers across separate networks.

30

OPENNESS EXAMPLE - 2

- **DESIGN B**: API is implemented using Java RMI
- The Java Remote Method Invocation (RMI) is a Java API that performs remote method invocation to allow Java objects to be distributed across different Java program instances on the same or different computers. RMI is the Java equivalent of C remote procedure calls, which includes support for transfer of serialized Java classes and distributed garbage-collection.
- **DESIGN C**: API is implemented as HTTP/RESTful web interface
- A RESTful API is an API that uses HTTP requests to GET, PUT, POST and DELETE data. RESTful APIs are referred to as a RESTful web services

31

TYPES OF SCALABILITY

- **Size scalability**: distributed system can grow easily ***without*** impacting performance
  - Supports adding new users, processes, resources
- **Geographical scalability**: users and resources may be dispersed, but communication delays are negligible
- **Administrative scalability:** Policies are scalable as the distributed system grows to support more users… (security, configuration management policies are agile enough to deal with growth) ***Goal: have administratively scalable systems !***
- Most systems only account for size scalability
- One solution is to operate multiple parallel independent nodes

32

SIZE SCALABILITY

- Centralized architectures have limitations
- At some point a single central coordinator/arbitrator node can't keep up
  - Centralized server: limited CPU, disk, network capacity
- Scaling requires surmounting bottlenecks

Lloyd W, Pallickara S, David O, Lyon J, Arabi M, Rojas K. Migration of multi-tier applications to infrastructure-as-a-service clouds: An investigation using kernel-based virtual machines. InGrid Computing (GRID), 2011 12th IEEE/ACM International Conference on 2011 Sep 21 (pp. 137-144). IEEE.

33

GEOGRAPHIC SCALABILITY

- Nodes dispersed by great distances
  - Communication is slower, less reliable
  - Bandwidth may be constrained
- How do you support synchronous communication?
  - Latencies may be higher
  - Synchronous communication may be too slow and timeout
  - WAN links can be unreliable

34

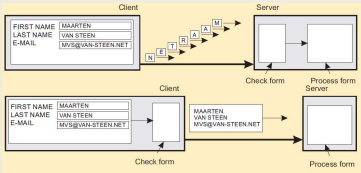
ADMINISTRATIVE SCALABILITY

- Conflicting policies regarding usage (payment), management, and security
- How do you manage security for multiple, discrete data centers?
- Grid computing: how can resources be shared across disparate systems at different domains, etc. ?

35

APPROACHES TO SCALING

- **Hide communication latencies**
  - Use asynchronous communication to do other work and hide latency
  - Remote server runs in parallel in the background – client not locked
  - Separate event handler captures return response from server
- Hide latency by moving key press validation to client:



36

APPROACHES TO SCALING - 2

- Partitioning data and computations across machines
- Just one copy
  - Where is the copy?
- Move computations to the client
  - Thin client → thick client
  - Edge, fog, cloud….
- Decentralized naming services (DNS)
- Decentralized information services (WWW)

37

APPROACHES TO SCALING - 3

- 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 caches (at server and client)
- **LOAD BALANCER** (or proxy server)
  - Commonly used to distribute user requests to nodes of a distributed system

38

PROBLEMS WITH REPLICATION

- Having multiple copies leads to inconsistency (cached or replicated)
- Modifying one copy invalidates all of the others
- Keeping copies consistent requires global synchronization
- Global-synchronization prohibits large-scale up
  - Best to synchronize just a few copies or synchronization latency becomes too long, entire system slows down!
  - ***Consider how synchronization time increases with system size***
- Can these inconsistencies be tolerated?

1. Current temperature and wind speed from weather.com
2. Bank account balance – for a read only statement
3. Bank account balance – for a transfer/withdrawal transaction

39

DEVELOPING DISTRIBUTED SYSTEMS

- Developing a distributed system is a formidable task
- Many issues to consider:
- Reliable networks do not exist
- Networked communication is inherently insecure

40

FALSE ASSUMPTIONS ABOUT DISTRIBUTED SYSTEMS

- 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

41

QUESTIONS

42