# CS 655: Advanced Topics In Distributed Systems [Introduction]

Shrideep Pallickara
Computer Science
Colorado State University



## Topics covered in this lecture

- Course outline
- Grading Policy
- Critical reviews
- Paper/Project
- Broad brushstroke coverage of issues in distributed systems
- List of papers that we will review



# Course Outline



## Course Outline

- We will be reviewing several papers in the area of distributed systems
- Look at the forces that drive evolution of distributed systems
- Understand not just the algorithms, frameworks and strategies that underlie the systems
  - But also the *trade-off space* for design decisions



## We will be looking at research in the following broad areas

- Data structures for large scale distributed systems
- Models for application development
- Process migrations
- Large-scale data storage systems
- Virtualization



## Grading Policy

- Critical Reviews: 40 %
- Class participation: 10%
- In-class presentations: 20%
- Final project and term-paper: 30%

- Final letter grades will be based on the *relative distribution* of total scores and not on any preset numerical scale.

## CRITICAL REVIEWS



## Critical reviews (1/3)

- This is around 2 pages
  - 1' margins, Times Roman 11, and single-spaced
- The objective is *not to defend* the paper
  - Poke and probe this paper to see if (where and when) things fall apart
- Include a 1 paragraph (¼ page) summary of the *key concepts* in the paper
  - This is not a rewording of the abstract of the paper
  - This something that your peers who have also read the paper can appreciate



## Critical reviews (2/3)

- Identify the **crux** of this paper
  - And also the *trade-off space* within which proposed solution will work
- What are the possible **inefficiencies** in this approach?
  - How can we address them?
  - *Construct situations* that cause things to break
  - If you could *redesign* components, how would you?
- What are the possible **extensions** to this work?
  - It is OK to come in from left-field and propose significant changes



## Critical reviews (3/3)

- You are required to turn in 8 critical reviews
  - Once every two weeks
  - You can choose which papers you will do this for



## Presentations (1/3)

- This will be a mini research exam
- You will meet with me regularly (*at least once a week*) to go over your presentation



## Presentations (2/3)

- Construct a hypothesis or a set of research questions that underpin the set of papers
- What are the settings that the proposed solutions work in?
  - This constrains both the solution space and the trade-offs that can be made
- The idea is to **weave a narrative** around 1 main paper and 1-2 related papers that I provide
  - The presenter is expected to contribute another relevant paper

## Presentations (3/3)

- Prepare for this as if it is your research exam
  - The presenter's critical review will be posted
- Make sure that you **rehearse** this presentation
- Each student is expected to *ask at least one question* in class.
  - The questions **should not** be
    - Softball questions
    - Questions for which you know the answer or are obvious
  - Be nice when you are asking questions



## Framing questions

- Let the presenter know where you are coming from
  - For e.g., "I always thought stream arrivals in sensor settings are stochastic. Wouldn't computing frameworks that go dormant for fixed intervals miss processing deadlines?"
- Have a clear objective when you pose a question?
  - You are trying to learn more or you disagree with the hypothesis or the methodology
  - Be specific
- Finally, a statement is not a question!



## Project and Paper

- You should **select an area** to do this paper/project
- Once you select an area, we can work on the components of the paper
  - This would include experiments
  - Comparable systems, etc
- We will have a **poster session** at the end of the semester



## At the end of each class, we will continue to have surveys

- 3 things that were clear
- 3 things that were difficult to understand



## Broad brushstroke coverage of distributed systems



## Distributed systems definition

- *A distributed system is a collection of independent computers that appears to its users as a single coherent system*
  - Andrew Tanenbaum
- *A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable*
  - Leslie Lamport

## Distributed Systems: GOALS

- Making resources accessible
- Distribution transparency
- Openness
- Scalability



## Making Resources Accessible



## Goal 1: Making resources accessible

- Share resources in a controlled and efficient fashion
  - Printers, computers, storage facilities, etc.
- Reasons for sharing?
  - Economics: Often cheaper to share than to have copies
  - Easy to collaborate and exchange information



## As connectivity and sharing increase, security issues become important

- Protection against intrusion and eavesdropping of communications
- Entities can track communications to build profiles of users
  - Violates privacy
  - Unwanted communications
    - Spam?



## Distribution Transparency



## Goal 2: Distribution transparency

- **Hide** the fact that the the constituent processes and resources are physically distributed
  - Across multiple computers
- System should present itself to the application & users as if it were a single computer system
  - **Transparency**

## Types of transparency

| Transparency | Description |
|---|---|
| Access | Hide differences in data representation and how a resource is accessed |
| Location | Hide where a resource is located |
| Migration | Hide the fact that a resource may move to another location |
| Relocation | Hide that a resource may be moved to another location while in use |
| Replication | Hide that a resource is replicated |
| Concurrency | Hide that a resource may be shared by several competitive users |
| Failure | Hide the failures and recovery of a resource |



## Degree of transparency: Sometimes we cannot hide all distribution aspects from users

- Time zone variations
- Communications are not instantaneous
  - Signal transmission is limited by the
    - Speed of light
    - Limited processing capabilities of intermediate switches



## Trade-off between transparency and performance of system

- Example: Mask transient server failure by *retrying* connections periodically
  - Maybe better to give up earlier or try another server
- Example: Several replicas need to be kept *consistent* at all times
  - If change is made to one copy it must be propagated to all copies before a new change can be made



## Distributed systems are expanding to devices that people carry around

- Notion of location and context awareness becomes even more important
  - Might be better to expose distribution rather than hide it
- Transparency must account for
  - Performance
  - Comprehensibility



## Openness



## Goal 3: Openness

- Offer services according to standard rules that describe syntax and semantics of those services
  - E.g., in networks, rules govern format, content, and meaning of messages
  - Formalized in protocols
- Services are generally specified through interfaces

## What are these interfaces?

- Specify names of functions
  - Along with types of parameters, return values, possible exceptions, etc.
  - IDL, WSDL
- Hardest part is specifying the semantics of these interfaces
  - What exactly does the service do?



## Interoperability

- The extent to which two implementations of components from different manufacturers
  - Coexist and *work together*
  - Merely by relying on each other's interfaces



## Portability

- Extent to which application developed for distributed system **A**
  - Can be executed, *without modification*, on a different distributed system **B** that implements the same interfaces as **A**



## Extensibility

- Configure system out of different components
  - Possibly from different developers
- Add new components, replace existing ones
  - Without affecting components that stay in place
- Examples:
  - Add parts that run on a different OS
  - Replace an entire file system



## Achieving flexibility

- System must be organized as a collection of **small**, replaceable or adaptable components
- Definitions for not only highest-level interfaces but also for internal parts of the system
  - Describe how these interact



## Flexibility example: Caching within a browser

- Browsers allow you to adapt caching policy
  - Specify size of the cache
  - Consistency check of cached document
- However you cannot influence other parameters such as:
  - How long can a document remain in the cache
  - Which document should be removed when the cache fills up
  - Cannot make caching decisions based on content
    - For e.g., metro train timetables rarely change

## What we need is a separation of policy and mechanism

- Browser should ideally provide facilities *only* for storing documents
  - Users decide which documents are stored and for how long
  - Perhaps user can implement policy as a pluggable browser component



## Scalability



## Goal 4: Scalability Dimensions

- **Size**
  - Should be possible to add *more users and resources* to the system
- **Geographically scalable**
  - Users and resources may be spatially far apart
- **Administrative scalability**
  - Should be easy to manage even if it spans many independent administrative domains



## Examples of scalability limitations

| Concept | Example |
|---|---|
| Centralized services | A single server for all users |
| Centralized data | A single on-line telephone book |
| Centralized algorithms | Doing routing based on complete information |



## Characteristics of decentralized algorithms

- No machine has complete information about system
- Machines make decisions based only on local information
- Failure of one machine does not ruin algorithm
- No implicit assumption that a global clock exists



## Scaling techniques: Asynchronous communications

- **Avoid waiting** for responses to requests that were issued
  - Works well in WAN and geographically dispersed settings
- Alternatively, use a **separate thread** of control for requests
- Does not work well in interactive applications

## Scaling techniques: Distribution

- Take a component and **split** them into smaller parts
  - **Disperse** these parts across the system
- Examples:
  - Domain Name System (DNS)
  - World Wide Web



## Scaling techniques: Replication

- Scaling problems usually manifest themselves as performance problems
  - Slow responses, and such
- Replication involves have dispersed copies
  - Increases **availability**
    - Better fault tolerance
  - **Balances load** between components
    - Leads to better performance



## Pitfalls



## Pitfalls: False assumptions made during development of a distributed application

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport costs is zero
- There is one administrator

List by:
Peter Deutch
Sun Microsystems



## Trade-off space in distributed systems

① CPU and memory utilization
② Synchronization overhead
  - Number, type, and size of messages exchanged
③ Consistency
  - Brewers CAP theorem
④ Time
  - Completion time, response times, etc.
⑤ Costs



## Ontogeny recapitulates Phylogeny

## After Charles Darwin's book ON THE ORIGIN OF SPECIES was published

- German zoologist Ernst Haeckl stated
  - Ontogeny recapitulates Phylogeny

*Development of an embryo* — *repeats* — *the evolution of the species*

- i.e. human egg goes through stages of being a fish, … , before becoming human baby
- Modern biologists think this is a gross simplification!



## Something vaguely similar has happened in the computer industry

- Each new species (*type of computer*) goes through the development that its ancestors did
  - Both in hardware and software
  - Mainframe, mini computers, PC, handheld, etc



## Much of what happens in computing and other fields is technology driven

- Ancient Romans lacked cars not because they liked walking
  - It is because they didn't know to build cars
- PCs exist not because people have a centuries-old pent-up desire to own one
  - It is now possible to manufacture them cheaply



## Technology affects our view of systems

- A **change** in technology renders some idea obsolete
  - Another change could revive it
- Especially true when change has to do with **relative performance**
  - Of different parts of the system



## Let's look at this relative performance

- When CPUs become faster than memories?
  - Caches become important to speed-up slow memory
- If new memory technology makes memories much faster than CPUs?
  - Caches will vanish!
- In biology extinction is forever
  - In computer science it is sometimes only for a few years



## Systems stand on the shoulders of those that have come before it

- Client-Server
- 3-tier (or N tiers)
- Distributed Objects (DCOM, CORBA, RMI)
- Message Passing based
  - Queuing, Publish/Subscribe, Peer-2-Peer
- Grid Computing
- Service Oriented Architectures (XML )
- Cloud Computing

**PAPERS THAT WE WILL LOOK AT ARE AVAILABLE ON THE SCHEDULE PAGE**



## Tasks for you

- Look at the papers
- You will be doing presentations at regular intervals



## References

- *Distributed Systems: Principles and Paradigms. Andrew S. Tanenbaum and Maarten Van Steen. 2nd Edition. Prentice Hall. ISBN: 0132392275/978-0132392273*
- *Andrew S Tanenbaum. Modern Operating Systems. 3rd Edition, 2007. Prentice Hall. ISBN: 0136006639/978-0136006633.*