CSC3121
Distributed Systems
Introduction and an Overview

Dr Paul Ezhilchelvan

Aim

- To expose the challenges in building distributed applications
- To understand the concepts behind the solutions
  - These solutions, on being implemented, form a part of middleware in distributed systems
  - SOAP, distributed transactions, etc.,
- Vendors implement these solutions
  - For various OS, process architecture
  - in various ways, in improved versions
  - Improvement requires conceptual understanding

Developing Distributed Apps

- What is a distributed system?
  - Autonomous Computers connected through network communication
    - LAN, Internet, Wireless…
- Developing a distributed application
  - Involves enabling a program in one computer to access 'resources' on other computers
- Middleware supports resource access in a transparent manner
  - *As though* the resources reside in the local computer
  - Creates a useful *illusion* to the developer

Being Transparent

- Hide resource distribution from developers and their applications
- Location transparency
  - An application need not be aware of resource location
  - Middleware sorts it out
- Migration transparency
  - Resource may have been moved, acquiring a new address
  - Middleware finds the new address in the next access
- Concurrent access -> sequential access
- Failure transparency

Access Transparency

- An app in computer C1 wants to read a file *fred* in C2
  - App is called the user or *client*
- If *fred* is local to C1, client can access it any time
- So, a program in C2 must be running 24/7
  - Dealing with access requests for *fred* from clients
  - It is called the *server*
- Say, client wants to
  - read *nbytes*
  - starting from position *pos*
  - Into local memory starting from address *buf*
- Client should do no more than what it would do, had *fred* been local

Access transparency in programming

- Had *fred* been local, the client would execute:
  read(*buf*, *fred*, *nbytes*, *pos*)
- Middleware in client machine
  - Is informed read(.., *fred*, ...) invocation cannot find *fred*
  - Creates a thread to deal with read(.., *fred*, ...)
  - The thread
    - locates *fred*
    - Turns the read instruction into a message
    - Sends the message to server machine
    - Puts the server response in *buf*
- Client programming is made easy by middleware
  - Remote procedure call (RPC)
  - Remote method invocation (RMI)
  - Simple Object Access protocol (SOAP)
  - Smart contracts in Hyperledger Blockchain systems

Transaction: Money Transfer

- Alice has a bank account **A**
- Bank's computer is **C1**
- Alice instructs **C1** to transfer £250 to Bob's account **B**
- Bob's bank's computer is **C2**
- **C1** executes this program:

  A = A-250 (A is 'local')

  B = B+250 (B is remote, so use SOAP/RPC)

  – Involves C1 sending a SOAP or RPC message to C2

Uncertainty Due to Crashes

- After sending RPC message but before receiving server response,
  – C1 detects that C2 had crashed
- Did C2 crash before or after doing B=B+250?
- C1 cannot know this.. until C2 recovers
- Detecting this, and repairing any damage
  – are not Alice's (user) problems
  – Application deals with them (e.g., using a Transaction Engine)
- Had A and B been objects in a single computer, recovery is not a problem
  – Atomic write
  – Distribution transaction must be atomic

Topics, Coursework & Pre-Requisite

- Topics covered:
  1. RPC
  2. Time, Clocks and Events in a Distributed System
  3. Atomic Transactions
  4. *Lock-Free Transactions??*
  5. *Computing a Global picture???*
- Module does not involve programming
- But has an important pre-requisite
  – Take notes / Ask questions
- I take delight in answering questions – an interesting part of my job!
- Assessment
  – **Two** exercises (summative)
  – Another design exercise in total order (formative)
  – very challenging exercises that test your understanding
  – Listen to the lecture videos and ask questions

Something to think about..

- Computer C1's clock reads
  11:31:53 when GMT = 11:30:00
- Computer C2's clock reads
  11:28:16 also when GMT = 11:30:00
- C1 and C2 are Internet connected
- Alice proposes the following to synchronize both clocks:
  – C1 sends its clock reading to C2;
  – C2 adjusts its clock to the reading it received;
- Under what condition(s), clocks of C1 and C2 are **guaranteed** to show the same value at the same GMT?
  – Note: Clocks reading 12:00:00 when GMT = 12:10:00 meets the condition of showing the same reading at the same GMT

References for Additional Reading

Chapters in Books:

- Andrew Tanenbaum and Marteen van Steen. *Distributed Systems: Principles and Paradigms*, Second Edition, Pearson/Prentice Hall (Chapter 4) (https://www.dropbox.com/s/0s94rmfo8op40yp/Distributed%20Systems_TanenbaumBook.pdf?dl=0)

- Marteen van Steen and Andrew Tanenbaum. *Distributed Systems*. Third Edition. (Chapter 4) Preliminary Version (https://komputasi.files.wordpress.com/2018/03/mvsteen-distributed-systems-3rd-preliminary-version-3-01pre-2017-170215.pdf)

- George Couloris, Jean Dillimore, Tim Kindberg and Gordon Blair. Distributed Systems: Concepts and Design. Fifth Edition, Addison Wesley. (Chapters 14-17) Preliminary Version. (http://bedford-computing.co.uk/learning/wp-content/uploads/2016/03/george-coulouris-distributed-systems-concepts-and-design-5th-edition.pdf)

Technical papers:

- *Time, clocks, and the ordering of events in a distributed system*, L Lamport. Communications of the ACM. (ftp://129.241.210.18/disk3/pub/vms/freevms/doc/lamport.pdf)
- *Distributed Snapshots: Determining Global States of Distributed Systems*, M Chandy and L Lamport, ACM Trans. On Computer Systems.