Distributed Systems (COMP9243)

Lecture 1 (B): Introduction

Slide 1
A Distributed Systems - what and why
B Hardware and Software
B Goals
B Overview - principles and paradigms
B Course details
2 Erlang

Reminder:

Slide 2
A distributed system is a collection of independent computers that are used jointly to perform a single task or to provide a single service.

Basic Goals of Distributed Systems

We want distributed systems to have the following properties:

- Transparency
- Dependability
- Scalability
- Performance
- Flexibility

This course will examine approaches and techniques for designing and building distributed systems that achieve these goals.

Transparency

Concealment of the separation of the components of a distributed system (single image view).

There are a number of forms of transparency:

- Access: Local and remote resources accessed in the same way
- Location: Users unaware of location of resources
- Migration: Resources can migrate without name change
- Replication: Users unaware of existence of multiple copies
- Failure: Users unaware of the failure of individual components
- Concurrency: Users unaware of sharing resources with others

Is transparency always desirable? Is it always possible?
**Dependability**

- Dependability of distributed systems is a double-edged sword:
  - Distributed systems promise higher availability:
    - Replication
  - But availability may degrade:
    - More components → more potential points of failure
- Dependability requires consistency, security, and fault tolerance

**Scalability**

A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity

(B. Clifford Neuman)

Scale has three dimensions:

- **Size**: number of users and resources (problem: overloading)
- **Geography**: distance between users and resources (problem: communication)
- **Administration**: number of organisations that exert administrative control over parts of the system (problem: administrative mess)

Note:
- Scalability often conflicts with (small system) performance
- Claim of scalability is often abused

**Scaling Up or Out?**

**Vertical Scaling: Scaling UP** Increasing the resources of a single machine

**Horizontal Scaling: Scaling OUT** Adding more machines

**Techniques for scaling:**
- Hiding communication latencies (asynchronous communication, reduce communication)
- Distribution (spreading data and control around)
- Replication (making copies of data and processes)
- Decentralisation
Decentralisation

Avoid centralising:

- Services (e.g., single server)
- Data (e.g., central directories)
- Algorithms (e.g., based on complete information).

With regards to algorithms:

- Do not require machine to hold complete system state Why?
- Allow nodes to make decisions based on local info Why?
- Algorithms must survive failure of nodes Why?
- No assumption of a global clock Why?

Decentralisation is hard

- Any system should strive for maximum performance
- In distributed systems, performance directly conflicts with some other desirable properties

Numbers every programmer should know

- L1 cache reference .................. 0.5 ms
- Branch mispredict .................. 5 ms
- L2 cache reference .................. 7 ms
- Mutex lock/unlock .................. 25 ms
- Main memory reference ............ 100 ms
- Compress 1K bytes with Zippy .... 3,000 ns = 3 us
- Send 2K bytes over 1 Gbps network .. 20,000 ns = 20 us
- Read 1 MB sequentially from memory .. 250,000 ns = 250 us
- Round trip within same datacenter ... 500,000 ns = 0.5 ms
- Disk seek .......................... 10,000,000 ns = 10 ms
- Read 1 MB sequentially from disk . 20,000,000 ns = 20 ms
- Send packet CA->Netherlands->CA . 150,000,000 ns = 150 ms

(from Peter Norvig, Jeff Dean; see also http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html)

Flexibility

- Build a system out of (only) required components
- Extensibility: Components/services can be changed or added
- Openness of interfaces and specification
  - allows reimplementation and extension
  - Interoperability
  - Separation of policy and mechanism
  - standardised internal interfaces
**COMMON MISTAKES**

False assumptions commonly made:
1. Reliable network
2. Zero latency
3. Infinite bandwidth
4. Secure network
5. Topology does not change
6. One administrator
7. Zero transport cost
8. Everything is homogeneous

**PRINCIPLES**

Several key principles underlying the functioning of all distributed systems:
- System Architecture
- Communication
- Partitioning, Replication and Consistency
- Synchronisation & Coordination
- Naming
- Fault Tolerance
- Security

Discussion of these principles will form the core content of the course

**PARADIGMS**

Most distributed systems are built based on a particular paradigm (or model):
- Shared memory
- Distributed objects
- Distributed file system
- Distributed coordination
- Service Oriented Architecture and Web Services
- Distributed Database
- Shared documents
- Agents

This course will cover the first five in detail.

**MISCELLANEOUS 'RULES OF THUMB'**

**Trade-offs** Many of the challenges provide conflicting requirements. For example, better scalability can cause worse overall performance. Have to make trade-offs - what is more important?

**Separation of Concerns** Split a problem into individual concerns and address each separately

**End-to-End Argument** Some communication functions can only be reliably implemented at the application level

**Policy vs. Mechanism** A system should build mechanisms that allow flexible application of policies. Avoid built-in policies.

**Keep It Simple, Stupid** Make things as simple as possible, but no simpler.
**Reading List**

*End-to-end Arguments in System Design* A classic paper arguing the end-to-end argument with excellent examples.

*A Note on Distributed Computing* Another classic paper showing the dangers of too much transparency in RPC-based distributed systems.

*Fallacies of Distributed Computing Explained* A good explanation of the 8 common mistakes made by architects and designers of distributed systems.

*Scale in Distributed Systems* A really good paper to read if you are interested in understanding more about scalability in distributed systems.

**Overview of Course**

1. Introduction and Erlang
2. System Architecture and Communication
3. Replication and Consistency, Distributed Shared Memory
4. Synchronisation and Coordination
5. Dependability and Fault Tolerance
6. Security
7. Naming
8. Distributed File Systems
9. Middleware, Distributed Objects, Publish/Subscribe, SOA, Web Services
10. Cloud Computing

**Extras:**

1. Distributed Systems in Practice (@ Google)
2. Blockchain

**Practical Course Details**

- Papers: classic and research: some mandatory, some optional
- Homework/Exercises: Familiarisation, DS programming
- Assignments: 2 assignments. 100 marks total.
- Optional assignment: no marks.
- Exam: Open book exam, 100 marks
- Final Mark:
  - weighted average: exam mark (65%) and total assignment mark (35%).
  - Exam mark must be at least 50% of maximum possible exam mark.

Note:

*Difficult course. Lots of work. Be prepared.*

And start the assignments on time!
**Homework**

Examples of Distributed Systems:
- Choose an existing distributed system and
  1. Research its structure (i.e. what is its internal architecture?)
  2. Evaluate how it satisfies each of the goals discussed

**Hacker’s edition:**
- For your chosen system:
  1. Are there any obvious mistakes in the architecture and design?
  2. Are there any strange design decisions? Why might they have been made?