

Introduction

PEC-CS-601G Parallel & Distributed System

Soumadip Biswas



Outline of the Course

- Course Webpage: <https://soumadip.github.io/courses/DS/>
- Broad overview
 - **Introduce you to distributed system design**
 - **Algorithms and system design issues**
 - **Real world Examples**
- Internals evaluation
 - Term projects, Case studies
- Reference books
 - Advanced Operating Systems by Mukesh Singhal and Niranjan Shivaratri
 - Distributed Systems: Principles and Paradigms by Andrew Tanenbaum and Maarten van Steen
 - Distributed Database Systems by D.Bell and J. Grimson, Addison

What is a Distributed System?

“A distributed system is a system where I can’t get my work done because a computer has failed that I’ve never even heard of.”

- Leslie Lamport

What is a Distributed System?

A network of autonomous machines/devices that communicate to perform some task

Modes of communication

- Message passing
- Distributed Shared Memory

Major Components

- Machines, Devices
 - PCs, servers, specialized devices
 - Commonly called **nodes**
- Network
 - links, switches, routers
- Storage
 - local, distributed
- Systems software/ applications/ tools
 - Distributed OS, databases, filesystems, load balancer, event, performance monitoring tools, security software...
 - Not all distributed system will need all
- Middleware
- User applications

Advantages

- Resource Sharing

- Example: share devices, software, services...across networks

- Better Performance

- Example: Parallel execution of tasks, load sharing between multiple servers, data replication closer to user....

- Fault Tolerance

- Example: Increase system availability by putting redundant/backup servers

- Handle inherently distributed data

- Example: Internet routing, distributed data mining

- Scalability

- Example: add more servers as needed if load increases

Common Characteristics

- Heterogeneous in many cases
 - Different architectures, operating systems....
 - Can be geographically distributed
 - Network delays play an important role
 - Faults are common
 - Larger the system is, more chance of something failing at any one time
- Replication is very widely used for better availability and performance
 - Need for maintaining data consistency
 - Trade-off between performance and consistency

Examples of Distributed Systems

- Almost every large system that you use is distributed
 - Online stores like Amazon, Flipkart,....
 - Content delivery services like Netflix, Hotstar,.....
 - Social networks like Facebook, Twitter,...
 - Google and its services
 - Cloud Services like Amazon AWS, Microsoft Azure
 - Travel/Ticket booking services like IRCTC, makeMyTrip, ...
 - Internet!
 - IEM CRP
- Not all build/manage their own distributed systems, rather use services provided by others.
- We will learn more later

Why are They Harder to Design?

- Lack of global shared memory
 - Hard to find the global system state at any point
 - Lack of global clock
 - Events cannot be started at the same time
 - Events cannot be ordered in time easily
 - Hard to verify and prove
 - More complex atomicity issues
 - Arbitrary interleaving of actions makes the system hard to verify
- Same problem is there for multi-process programs on a single machine
 - Harder here due to communication delays

Example: Lack of Global Memory

- Problem of Distributed Search

- A set of elements distributed across multiple machines
- A query comes at any one machine A for an element X
- Need to search for X in the whole system

- Sequential algorithm is very simple

- Search and update done on a single array in a single machine
- No. of elements also known in a single variable

Example: Lack of Global Memory (contd.)

- How to send the query to all other machines?
- Do all machines even know all other machines?
- How to get back the result of the search in each machine?
- Handling updates (both add/delete of elements at a machine and add/remove of machines) – adds more complexity

- **Main problem**

No one place (*global memory*) that a machine can look up to see the current system state (*what machines, what elements, how many elements*)

Example: Lack of Global Clock

- Problem of Distributed Replication
 - 3 machines A, B, C have copies of a data X, say initialized to 1
 - Query/Updates can happen in any m/c
 - Need to make the copies consistent within short time in case of update at any one machine
- Naïve algorithm
 - On an update, a machine sends the updated value to the other replicas
 - A replica, on receiving an update, applies it

Example: Atomicity Issues

- Problem of Symmetry Breaking
 - 2 nodes, each with a value 1
 - Need to get to a final state with one node having value 0 and one node having value 1

Algorithm

- Each node sends a message to the other to know its value
- Each node sets its own value to the complement of the value received from the other node if the two values are the same
 - May never terminate if send-receive-set is not atomic

Distributed Scheduler (not atomic) vs. Central Scheduler (atomic)

- Is a central scheduler in a distributed system practical?