BMCS3003 Distributed Systems and Parallel Computing

L01 - Introduction to Distributed Systems

Presented by

Assoc Prof Ts Dr Tew Yiqi
May 2023

Table of contents

01

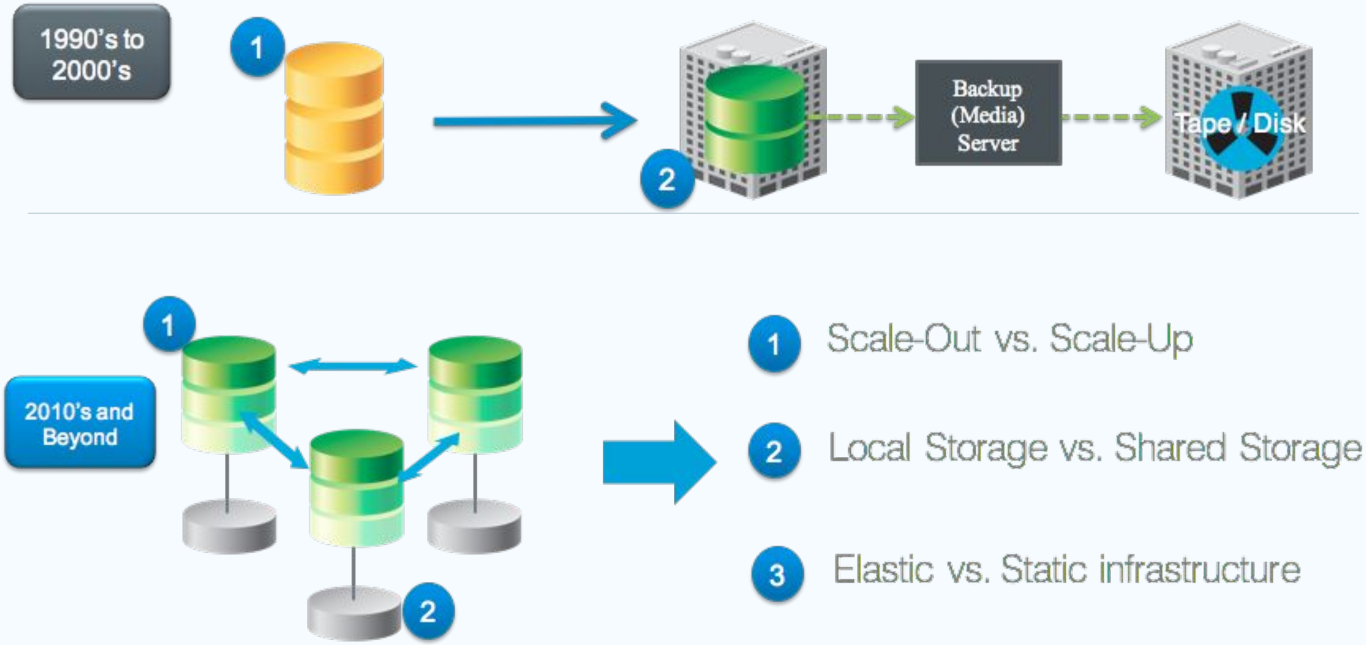
Distributed Systems

A computing environment in which various components are spread across multiple computers.

Tradisional vs Distributed

Tradisional databases

Distributed databases



Source: insidebigdata.com

Basic Concept

- A distributed computing system is one in which the computation is distributed / spread across multiple processing entities.
- Various aspects of the system can be distributed:

Database / data

Operating System

File System

Business Logic

Authentication

Workload
Resoures allocation / load sharing

Benefits

Scalability

Continuously evolve in order to support the growing amount of work.

Reliability

Keep delivering its services even when one or several of its software / hardware components fail

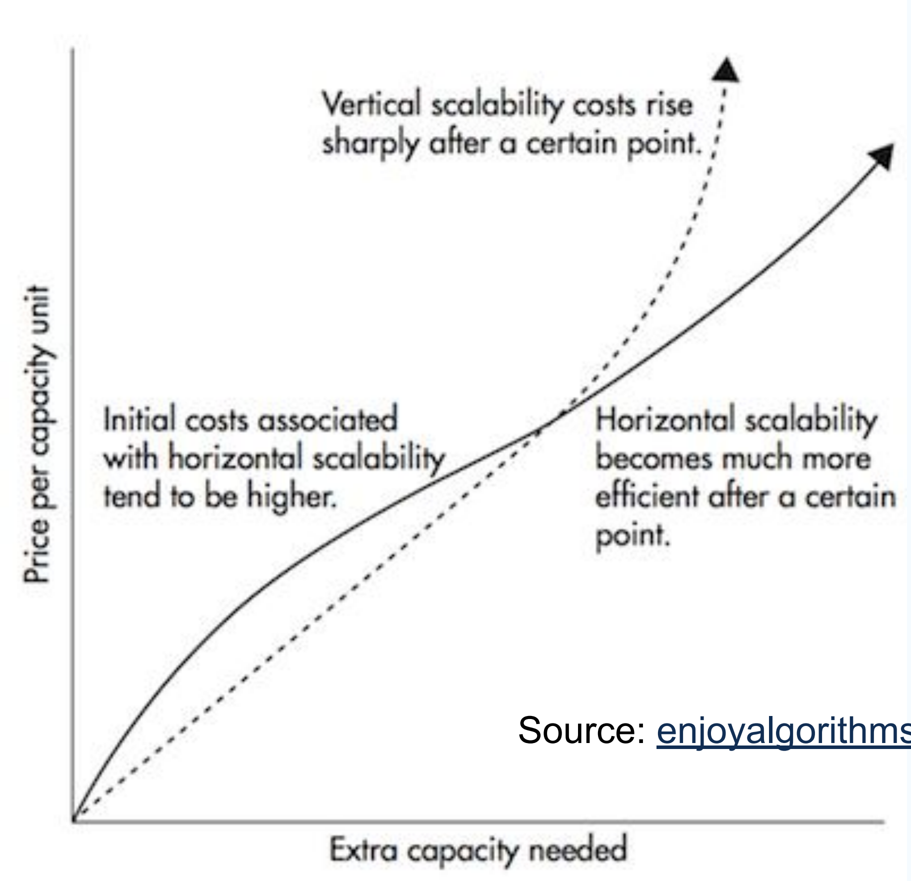
Performance

Availability to interact and coordinate actions to appear to the end-user as a single system

Geographical

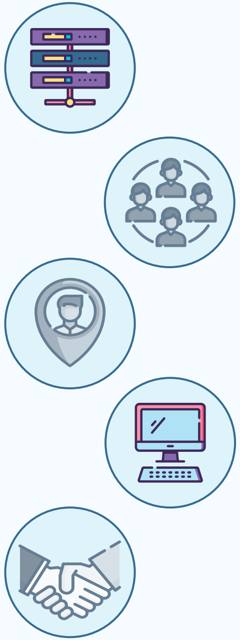
Distributed processing and resources based on application specific requirements and user locality.

Benefits - Scalability



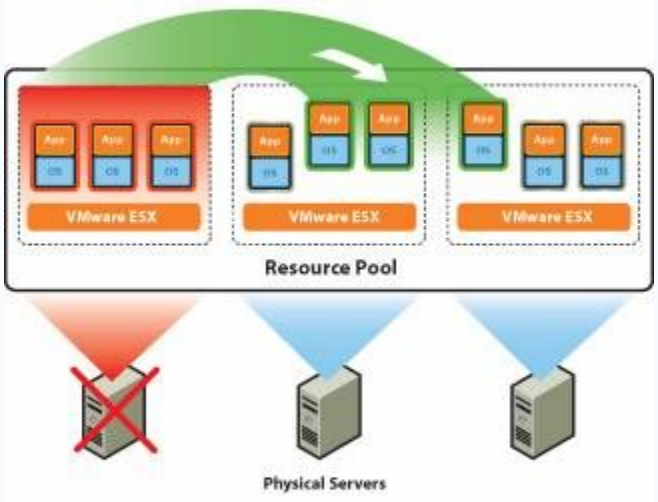
Source: enjoyalgorithms.com

Challenges when building Distributed Systems



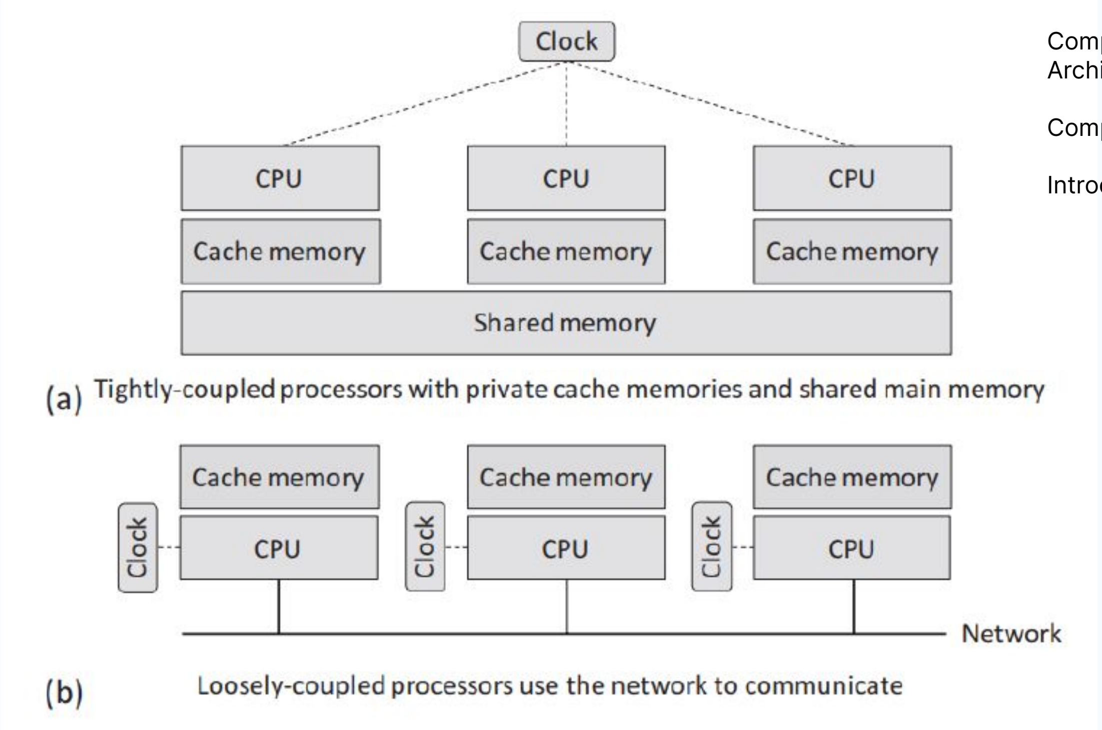
- Avoid single point of failure.
- Replication.
- Availability and performance.
- Resource naming, addressing and location of resources.
- Binding (mapping between parts of the system).

Benefits - Fault Tolerance



Case Study: [Facebook's Maelstrom system]

Tightly and loosely coupled hardware architectures



Recall the syllabus in

Computer Organisation and Architecture /

Computer System Architecture /

Introduction to Computer System

Tightly-coupled systems

The processor units are physically part of the same computer.

Processors are connected:
By a high-speed blackplane bus, or are on the same “motherboard”,
or in the same integrated circuit (Chip)

Specialised hardware:

- Fixed architecture (number of processors)
- Expensive
- Multi-core (typically 2 or 4 CPUs in PCs at present)
- Large scale (of order of 64 processors and greater) not common (also referred to as parallel processing systems)

Shared clock - synchronisation is possible.
Shared memory - fast / reliable inter-processor communication.

Activity

What is the output of the program ?

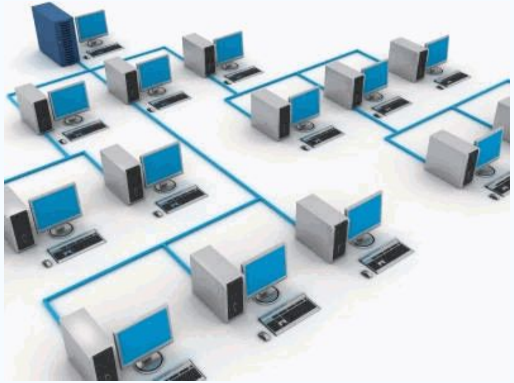
```cpp
#include "stdafx.h"
#include <iostream>
#include <omp.h>
using namespace std;

int main()
{
#pragma omp parallel
	{
		cout << "Hello World\n";
	}
	return 0;
}
```

Loosely-coupled systems

The processor units are within separated computers.

The computers are connected by a network technology:

General purpose hardware:
- Cheap
- Abundant



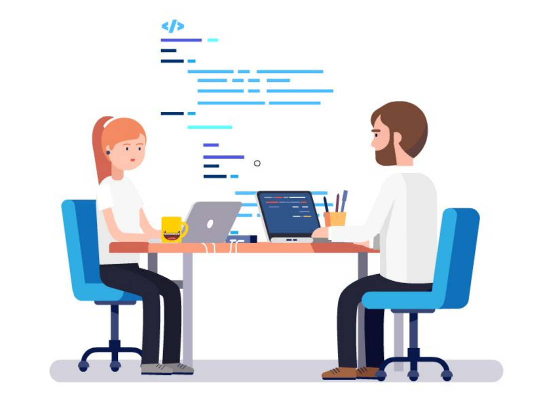
Tightly-coupled systems (Challeges)

- Each computer has its own clock:
  - absolute synchronisation NOT possible.
  - ‘loose’ synchronisation is necessary.
- Computers have separate memory not suitable for inter-processor communication.
- The individual computers are Autonomous need some overall guidance.
- The individual computers are Heterogeneous different memory size, disk size, processor speed, hardware platform, operating system etc.

Motivation for Loosely-coupled Distributed Systems (1/2)

The Interest in distributed systems has grown because of:

- The need to share large amounts of data
- The availability of cheap workstations
- The need to share expensive peripherals
- The availability of cheap high speed networks



Motivation for Loosely-coupled Distributed Systems (2/2)

- The need for local control but overall access
- The need to communicate and interact
- The need for flexibility of growth
- The need to provide users with facilities with realistic response times.

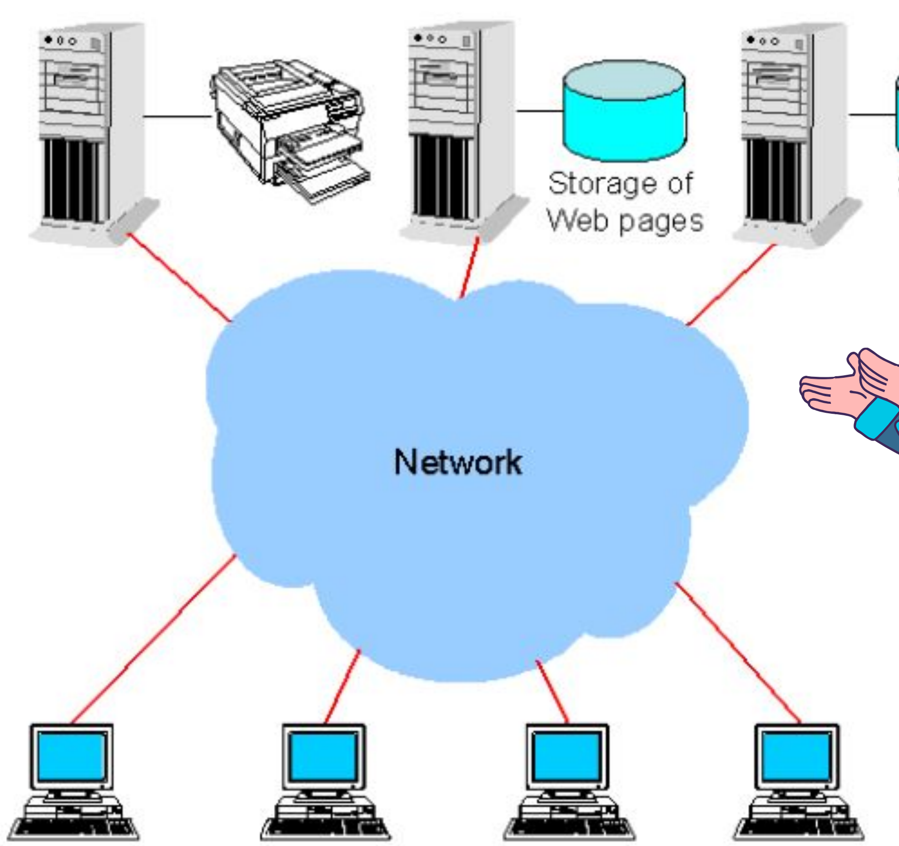
More example on Loosely vs Tightly: embedded.com

Distribution System Architecture

The Workstation and "Server" model

A server is a process, not a computer !

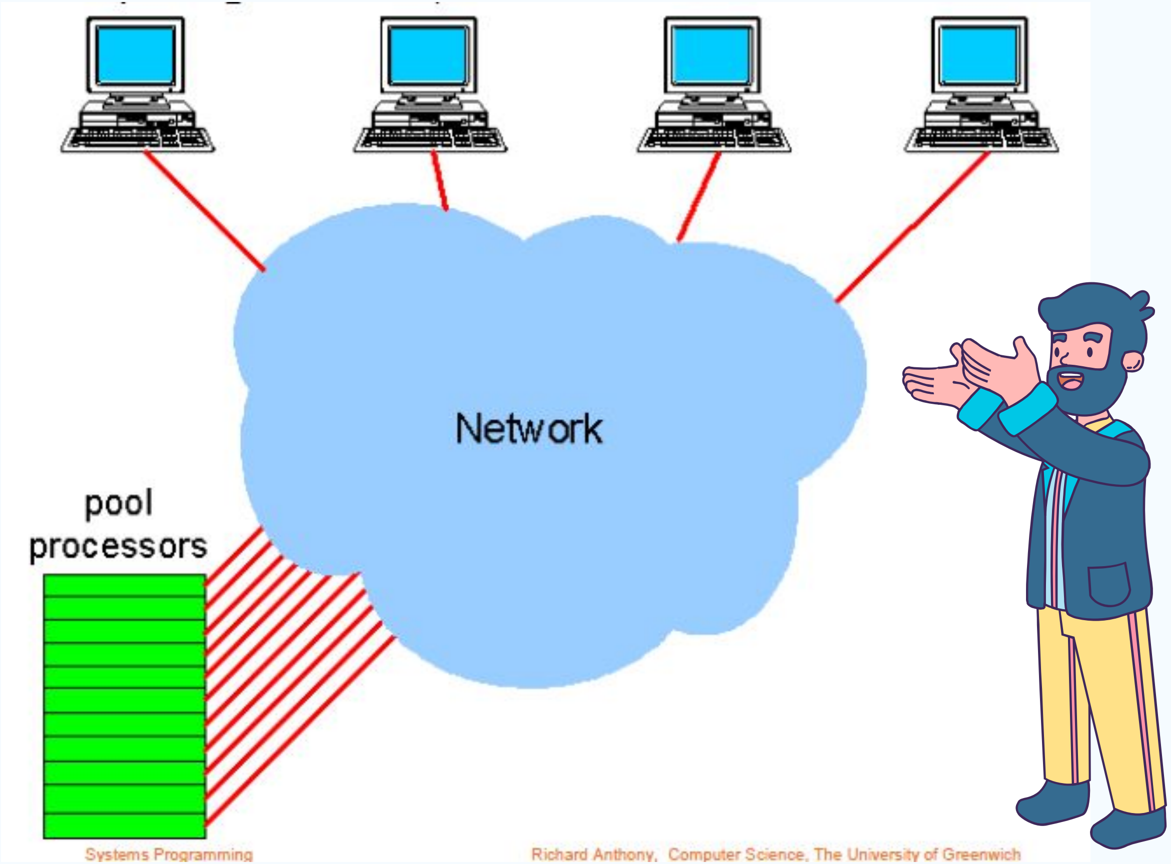
Powerful computers host services (file service, database, web service etc.)



Distribution System Architecture

The Processor-Pool model

"Grid Computing" is based on this model, but tends to be larger scale and can be across multiple organisations



Transparency (1/5)

- Distributed systems present numerous **challenges** to the developer, such as
- Where is the process ? Can it be moved ?
- Where is the data / resource ? Can it (data) be moved?
- Providing robustness, and dealing with failures
- Ensuring consistency
- Building scalable systems (communication efficiency, interaction model).

Transparency (2/5)

- Transparency means ***hiding*** the details of distribution
- The goal is to ***reduce the burden on developers*** so that they can focus their efforts on the 'business logic' of the application and not have to deal with all the vast array of technical issues arising because of distribution.

Transparency (3/5)

Access transparency

- Local and remote objects may be accessed with the same operations

Location transparency

- Objects can be accessed without knowledge of their location

Concurrency transparency

- Concurrent processes can use shared objects without interference.

Transparency (4/5)

Replication transparency

- Multiple copies of objects can be created without any effect of the replication seen by applications that use the objects.

Failure transparency

- Faults are concealed such that applications can continue without knowledge that a fault has occurred.

Migration transparency

- (For data objects) Objects can be moved without affecting the operation of applications that use those objects.
- (For processes) Processes can be moved without affecting their operations or results.

Transparency (5/5)

Performance transparency

- The performance of systems should degrade gracefully as the load on the system increases.

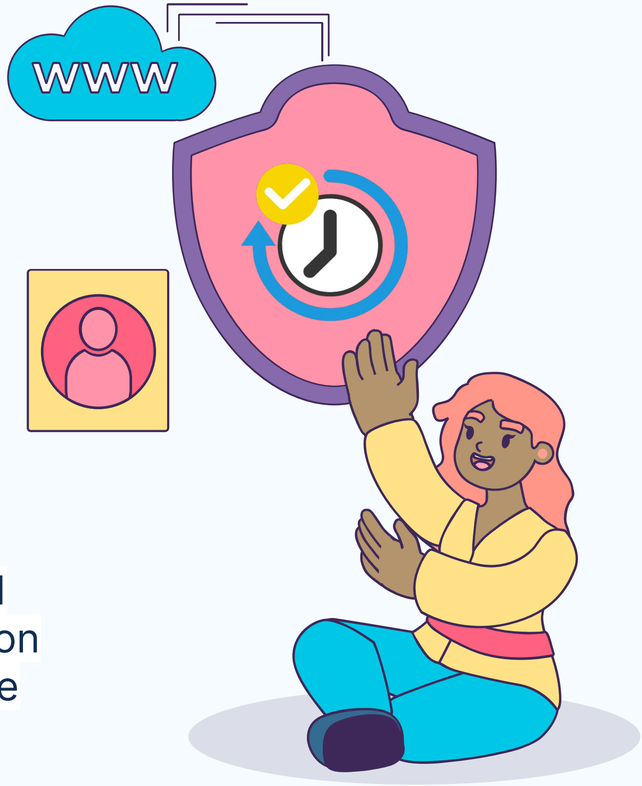
Scaling transparency

- It should be possible to scale up an application, service or system without changing the system structure or algorithms.

02

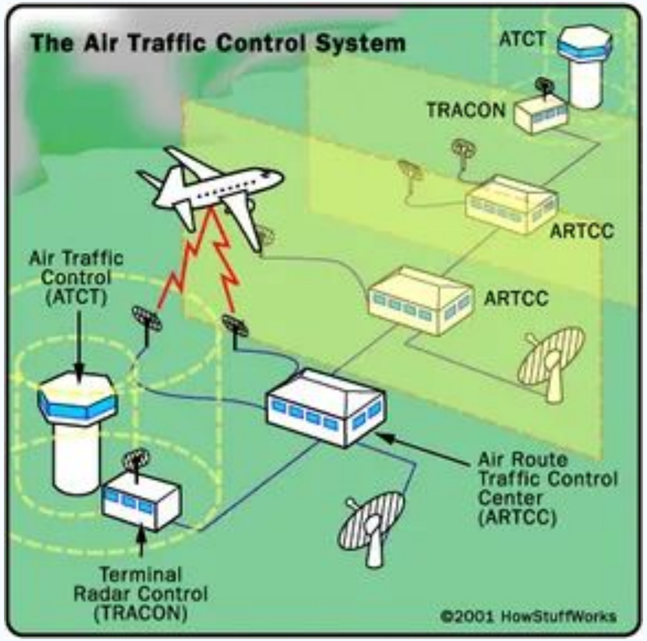
Real-time System

Any information processing system with hardware and software components that perform real-time application functions and can respond to events within predictable and specific time constraints



Real-time System

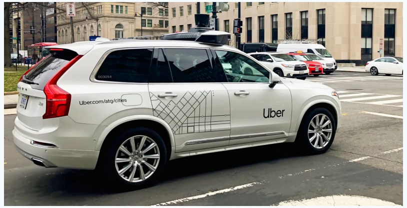
A real time system consists of a controlling system (computer) and a controlled system (environment)

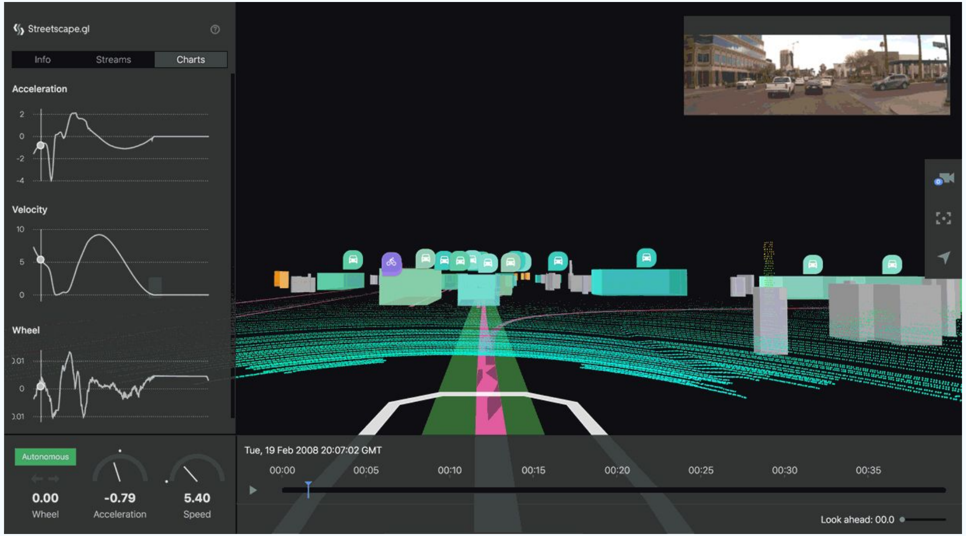


Source: howstuffworks.com

Real-time System

A real time system consists of a controlling system (computer) and a controlled system (environment)





Source: uber.com , AVS streetscape

Typical Feature of Real-time system

- Time critical
- Made up of concurrent processes (Tasks)
- Share resources (e.g. processor) and communicate with each other
- Reliability and fault tolerance are essential
- Perform a certain specific job
- Car, CD player, phone, camera etc

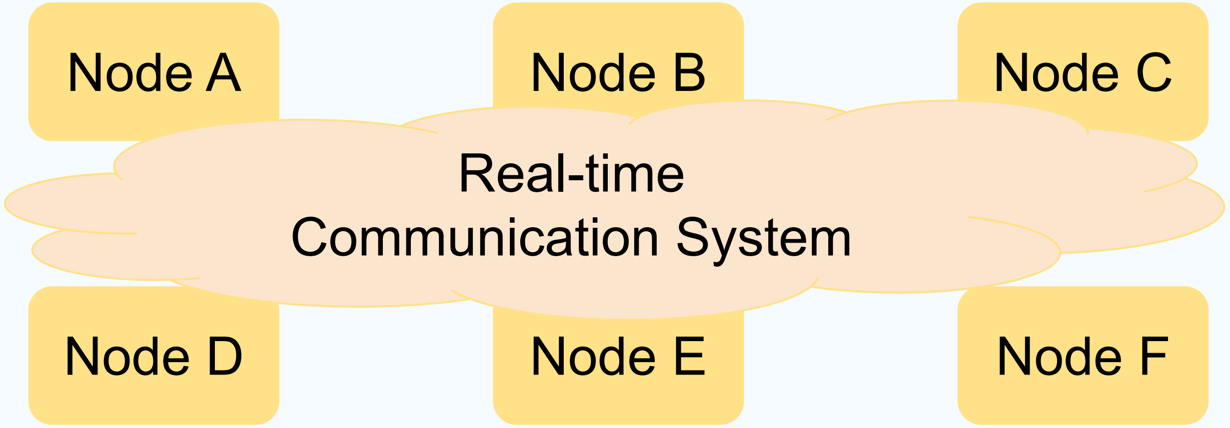
Distributed Real-time system

- Real-time systems very often are implemented as distributed systems. Some reasons:
  - Fault tolerance
  - Certain processing of data has to be performed at the location of the sensors and actuators.
  - Performance issues

Distributed Real-time system

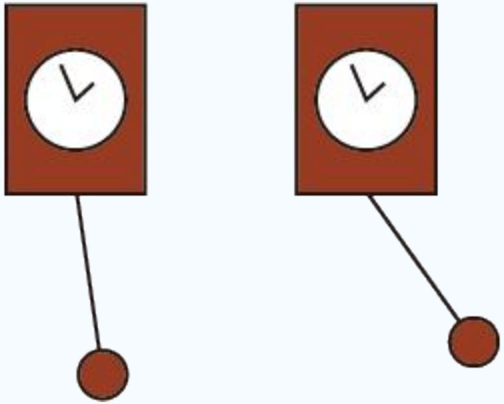
- If the real time computer system is distributed, it consists of a set of (computer) nodes interconnected by a real time communication network.

Source: seas.upenn.edu



Specific Issues Concerning Distributed Real-time system

- Clock synchronisation
- Real-time communication

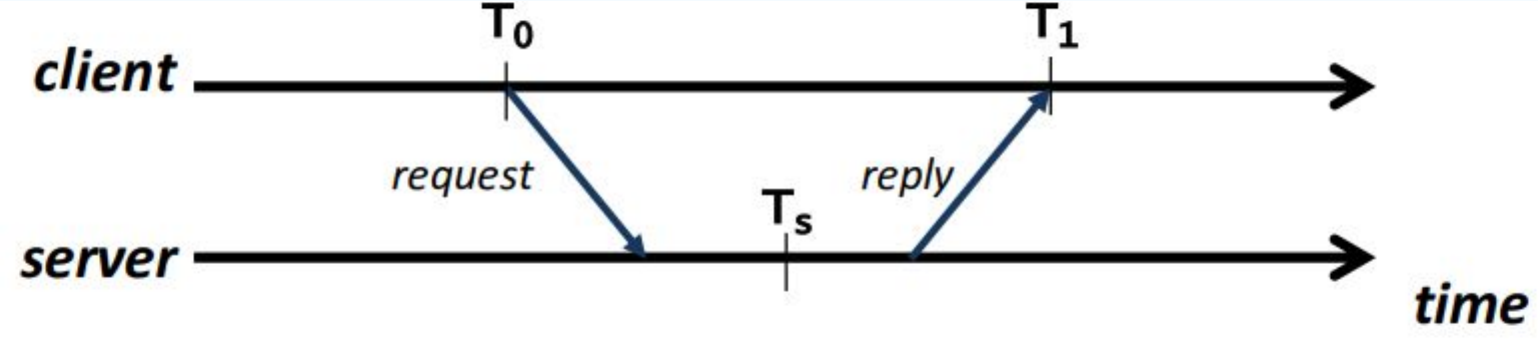


Understanding the clock synchronisation

- Started to look at time in distributed systems
  - Coordinating actions between processes
- Physical clocks ‘tick’ based on physical processes (e.g. oscillations in quartz crystals, atomic transitions)
  - Imperfect, so gain/ lose time over time
  - wrt nominal perfect ‘reference’ clock (such as UTC)
- The process of gaining/ losing time is **clock drift**
- The difference between two clocks is called **clock skew**
- **Clock synchronization** aims to minimize clock skew between two (or a set of) different clocks.

Clock synchronisation: Cristian's Algorithm (1989)

- Uses a time server that is synchronized to Coordinated Universal Time (UTC)

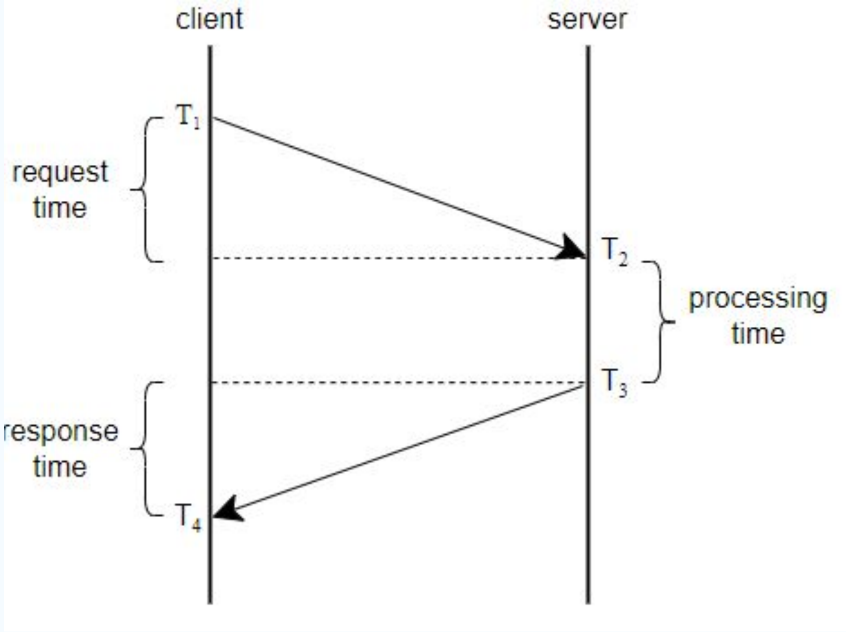


Clock synchronisation: Cristian's Algorithm (1989)

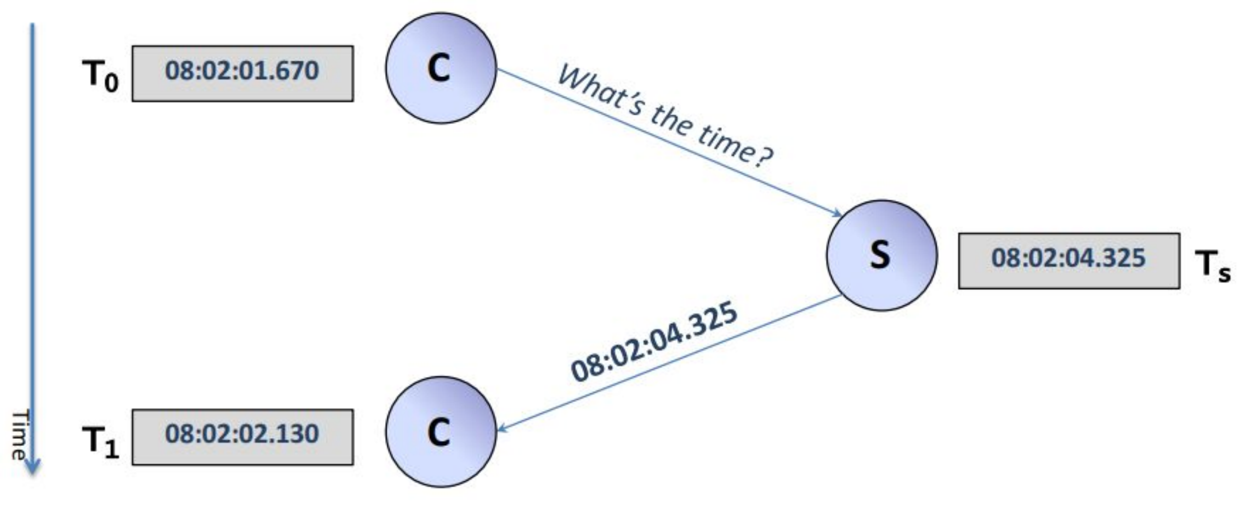
Attempt to compensate for network delays:

- Remember local time just before sending: T_1
- Server gets request, and puts T_s into response (processing time, $T_s = T_3 - T_2$)
- When client receives reply, notes local time: T_4
- Correct time is then approximately $(T_s + (T_4 - T_1) / 2)$

* assumes symmetric behaviour...



Clock synchronisation: Cristian's Algorithm (1989) Example:



Gain = 08:02:02.130 - 08:02:04.555 = 2.425s

Round Trip Time (RTT) = 460ms, so one way delay is [approx] 230ms.

Estimate correct time as (08:02:04.325 + 230ms) = 08:02:04.555

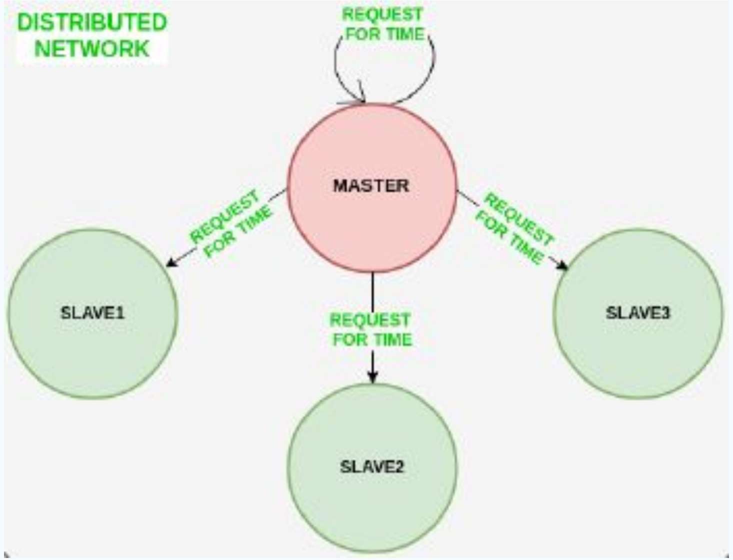
Client gradually adjusts local clock to gain 2.425 seconds

Clock synchronisation:
Cristian's Algorithm (1989)

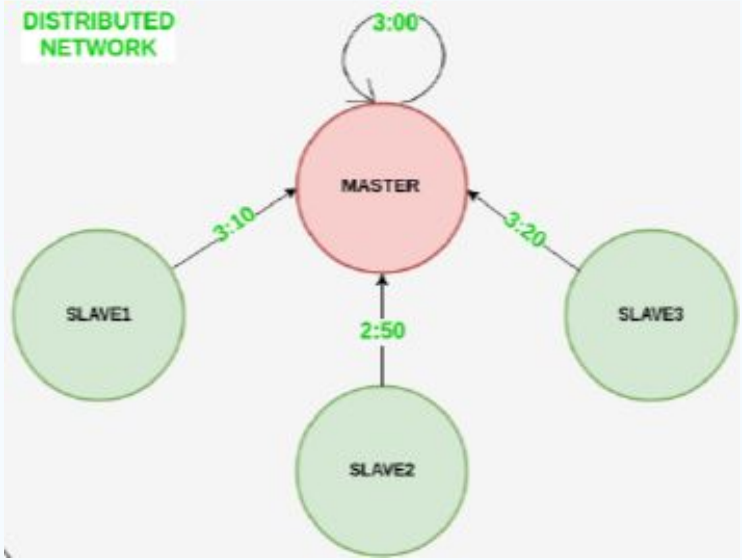
Problem

1. Network delays are time varying
2. Network delays can be different in each direction, even on the same link
3. The processing delay on each computer is also time varying

Clock synchronisation: Berkeley Algorithm (1989)

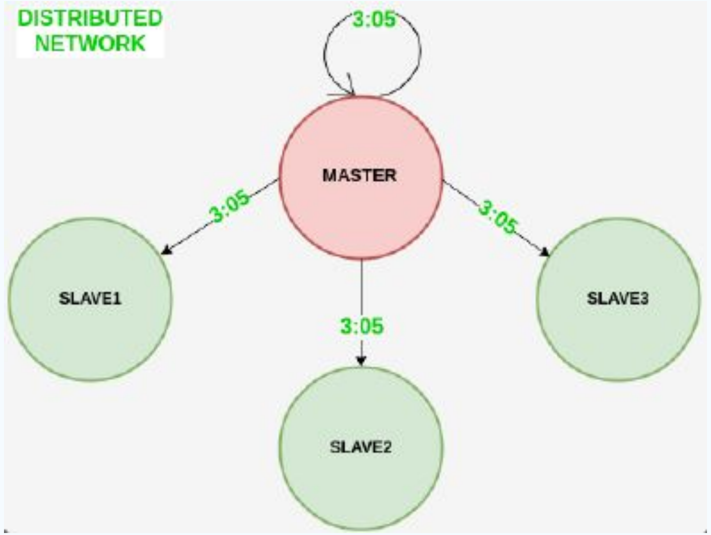


1) The master sends request to slave nodes.



2) The slave nodes send back time given by their system clock

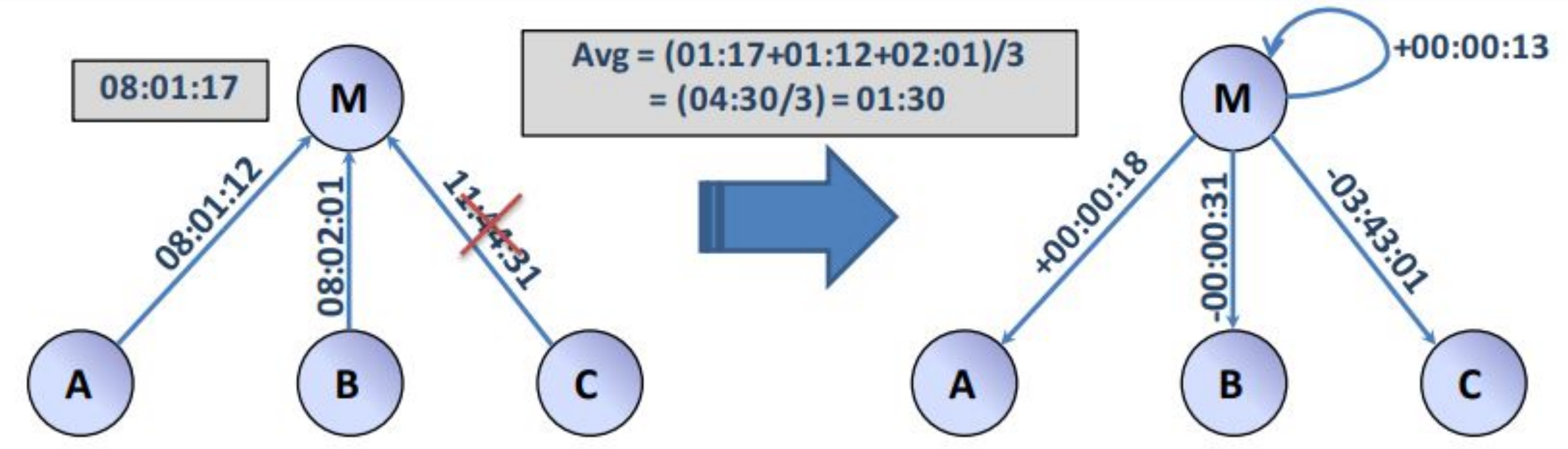
Clock synchronisation: Berkeley Algorithm (1989)



3) Broadcasting synchronised time to whole network

Clock synchronisation: Berkeley Algorithm (1989) Example

Master computes average (including itself, but ignoring outliers), and sends an adjustment to each machine.



Clock synchronisation: Berkeley Algorithm (1989)

Pseudocode

More info : [cl.cam.ac.uk](cl.cam.ac.uk)

```
# receiving time from all slave nodes
repeat_for_all_slaves : time_at_slave_node =
receive_time_at_slave ()

#calculating time difference time_difference =
time_at_master_node time_at_slave_node

#average time difference calculation
average_time_difference = all_time_differences ) /
number_of_slaves
synchronized_time = current_master_time +
average_time_difference

#broadcasting synchronized to whole network
broadcast_time_to_all_slaves synchronized_time
```

03

Operating System

manages all of the software and hardware on the computers.

What does an OS do ?

Process / Thread Management

Scheduling / Communication / Synchronization

Memory Management

Storage Management

File Systems Management

Protection and Security

Networking

Types of OS with distributed processing features

Network Operating Systems

Microsoft Windows Server, UNIX, Linux, Mac 0S X.

Distributed Operating System

Solaris, Micros, Mach.

Difference between the two types

Autonomy

System Image

Fault Tolerance Capability

Reference: [tutorialpoints]

Difference between types of DOS

Item	Distributed OS		Network OS
	Multiproc.	Multicomp.	
Degree of transparency	Very High	High	Low
Same OS on all nodes	Yes	Yes	No
Number of copies of OS	1	N	N
Basis for communication	Shared memory	Messages	Files
Resource management	Global, central	Global, distributed	Per node
Scalability	No	Moderately	Yes
Openness	Closed	Closed	Open

Enjoy the chronology

Thank you