# Contents

Chapter 1

# Chapter 1: Overview of Distributed Systems

## 1.1 Role of Middleware in A Distributed System

Middleware aims at improving the single-system view that a distributed system should have. In other word, to enhance the distribution transparency that is missing in network operating systems.

## 1.2 Distribution Transparency and Example

Distribution transparency is the phenomenon by which distribution aspects in a system are hidden from users and applications.
Examples include access transparency, location transparency, migration transparency, relocation transparency, concurrency transparency and failure transparency.

## 1.3 Reason for Why It Is Hard to Hide the Occurrence and Recovery From Failures in a Distributed System

It is generally impossible to detect whether a server is actually down, or that it is simply slow in responding.

## 1.4 Why It Is Not always a Good Idea to Aim at Implementing the Highest Degree of Transparency Possible

There is a trade-off between high degree of transparency and users accepted performance. If you try to aim to implement the highest degree of transparency, it could lead to slower-than-accepted performance.

## 1.5 An Open Distributed System and Benefits

An open distributed system offers services according to clearly defined rules. An open system is not only capable of easily interoperating with other open systems but also allows applications to be easily ported between different implementations of the same system.

## 1.6 A Scalable System

A system is scalable when it can grow in one or more dimensions, with respect to either its number of components, geographical size or number and size of administrative domains, without an unacceptable loss of performance.

## 1.7 Different Techniques of Scalability

Scaling can be achieved through distribution, replication, and caching.

Chapter 2

# Chapter 2: Architectures

## 2.1 Problem of Physical Distance

From what I gathered, the physical distance only has a significant affect on latency if you are to implement a system that capture a live data and than process it in almost real time for scientific purpose. If you built that kind of system (strongly depend on time), a simple solution would be to put the processing station near the source of information and then transmit a processed file.
Some of the common strategies to deal with latency are:

- caching data on the client - i.e. avoiding to make request again if possible
- combining several smaller requests into bigger ones
- compressing the request
- reusing the same connection for many requests
- using different protocols (i.e. a lot of games use UDP to avoid latency at the cost of making other things much more complex)
- caching data on the server
- redesigning the way to make requests (pre-loading data, pre-allocating resources, creating parallel requests)
- using content delivery networks (CDN) for static data
- running a code through a profiler and redesigning it to eliminate bottlenecks
- minimizing critical sections of code
- designing the service as stateless so that it can be deployed on many servers in parallel
- correctly sizing, designing and maintaining the system (OS, database, virtualization, load balancers, DNS caches, etc.)
- correctly sizing, designing and maintaining the network (having enough bandwidth, using fast routers, traffic prioritization, resource reservation)

## 2.2 Three-tiered Client-server Architecture

A three-tiered client-server architecture consists of three logical layers, where each layer is implemented at a separate machine, in principle. The highest layer consists of a client user interface, the middle layer contains the actual application, and the lowest layer implements the data that are being used.

## 2.3 Differences Between a Vertical Distribution and a Horizontal Distribution

- Vertical distribution refers to the distribution of the different layers in a multitiered architectures across multiple machines. In principle, each layer is implemented on a different machine.
- Horizontal distribution deals with the distribution of a single layer across multiple machines, such as distributing a single database.

## 2.4 Disadvantage of Topology of the Overlay

The main problem here is dealing with logical paths. In practice, two nodes A and B which are neighbours in the overlay network may be physically placed far apart. As a result, the logically short path between A and B in this instance required routing a message along a very long path in the underlying physical network.

## 2.5 Main Problems with Sequential Organization when Taking a Look at the Request-reply Performance at Process $P_1$ (The Last One)

- Performance can be expected to be bad for large n since the problem is that each communication between two successive layers is, in principle, between two different machines.
- The performance between $P_1$ and $P_2$ may also be determined by $n-2$ request-reply interactions between the other layers.
- Another problem is that if one machine in the chain performs badly or is even temporarily unreachable, then this will immediately degrade the performance at the highest level.

## 2.6 Goodness of Routing Policy that Forward messages to the Closest Node Toward the Destination

According to the pictures of the question sheet, consider forwarding message from node $(0.2; 0.3)$ (called $A$) to node $(0.9, 0.6)$ (called $B$). There are 2 routes:

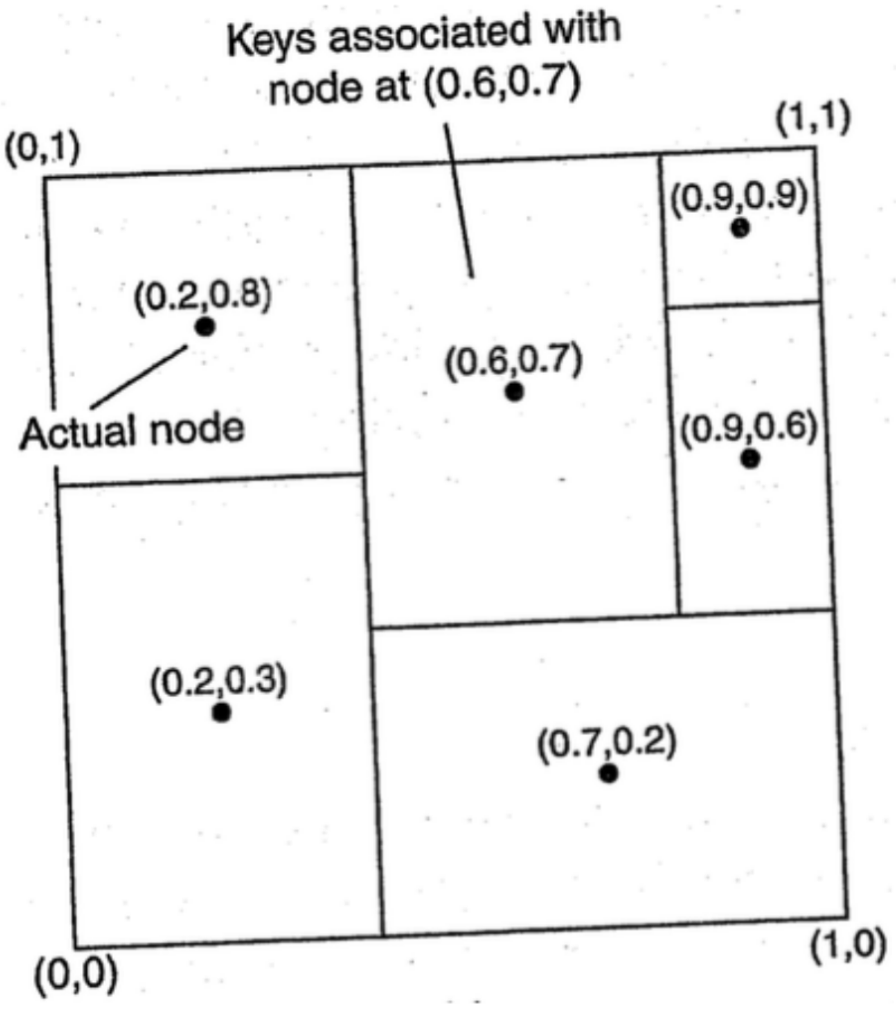

Figure 2.1: CAN network

1. $A$, through $(0.6; 0.7)$ (called $C$), to $B$:

$$AC = \frac{2\sqrt{2}}{5} \approx 0.566$$

$$CB = \frac{\sqrt{10}}{10} \approx 0.316$$

$$TotalDistance = \frac{\sqrt{10} + 4\sqrt{2}}{10} \approx 0.882$$

2. $A$, through $(0.7; 0.2)$ (called $D$), to $B$:

$$AD = \frac{\sqrt{26}}{10} \approx 0.510$$

$$DB = \frac{\sqrt{5}}{5} \approx 0.447$$

$$TotalDistance = \frac{\sqrt{26} + 2\sqrt{5}}{10} \approx 0.957$$

Because our policy is to forward a message to the closest node toward the destination, which in this case D, we will actually go a little longer than if we take route 1. In conclusion, this policy is a reasonable one, but it is not always the best one.

## 2.7 Benefits of Microservices Architecture Compared To Monolithic Architecture

- Microservices architecture makes systems and applications much faster to develop, and much easier to understand and maintain since it tackles the problem of complexity by decomposing application into a set of manageable services.
- Microservices architecture enables several teams to works on a projects, each can focus on one service or serveral closely-related services thus allowed services to be developed independently.
- Microservices architecture reduces barrier of adopting new technologies since the developers are free to choose whatever technologies make sense for their service and not bounded to the choices made at the start of the project.
- Microservice architecture makes continuous deployment possible for complex applications because it enables each microservice to be deployed independently.
- Microservice architecture enables each service to be scaled independently.

## 2.8 Design an E-commerce System Using Microservices Architecture

Services in the system:

- HTTP apache: Get the HTTP request and forward it to the right service
- Customer: Handling customer data like: name, address, age, gender , preferences et cetera

- Order: Handling order information: who buying, from whom, what sort of item/items, how many item and for how much
- Catalogue: Handling item information: of what category, how many variants (colour, size, shape, version), contain what items (for bundle), how many items left, unit price, rating
- Vendor: Handling vendor information: name, address, vendor rating, what items are they selling, number of items sold in a time period (for customer to know if they have been receiving lots of requests)

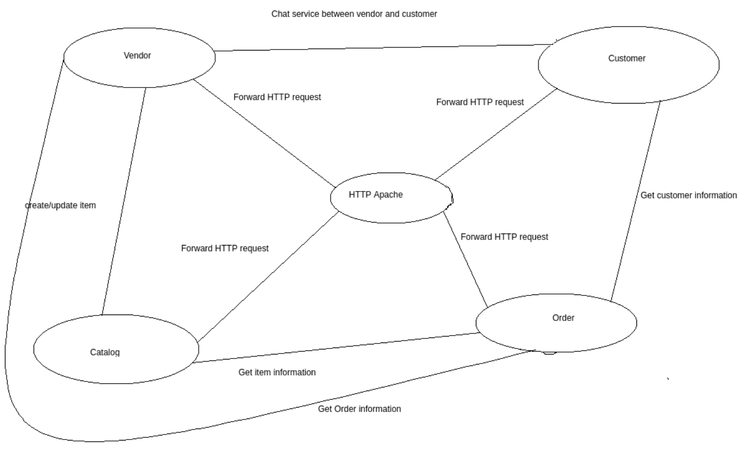

Figure 2.2: Figure to illustrate how the system interact

Chapter 3

# Chapter 3: Processes and Threads

## 3.1 Compare Process with Thread

| **Process** | **Thread** |
|---|---|
| System calls needed. | No system calls involved. |
| Context switching required. | No context switching required. |
| Different process have different copies of code and data. | Sharing same copy of code and data can be possible among different threads. |
| Operating system treats different process differently. | All user level threads treated as single task for operating system. |
| If a process got blocked, remaining process continue their work | If a user level thread got blocked, all other threads get blocked since they are single task to OS. |
| Processes are independent. | Threads exist as subsets of a process. They are dependent. |
| Process run in separate memory space. | Threads run in shared memory space. memory space. And use memory of process which it belong to. |
| Processes have their own program counter (PC), register set and stack space. | Threads share Code section, data section, address space with other threads. |
| Communication between processes requires some time. | Communication between processes requires less time than processes. |
| Processes don't share the memory with any other process. | Threads share the memory with other threads of the same process |
| Process have overhead. | Threads have no overhead. |

## 3.2 Limit the Number of Threads in a Server Process

Yes, for two reasons.

- First, threads require memory for setting up their own private stack. Consequently, having many threads may consume too much memory for the server to work properly.
- Secondly, to an operating system, independent threads tend to operate in a chaotic manner. In a virtual memory system it may be difficult to build a relatively stable working set, resulting in many page faults and thus I/O. Having many threads may thus lead to a performance degradation resulting from page thrashing. **(more important)**.

## 3.3 Lightweight Process

### 3.3.1 As a Solution of Combining Advantages of Deploying Thread Package in User and Kernel Space

- Responsiveness
- Resource Sharing
- Economy
- Utilization of Multiprocessor Architectures

### 3.3.2 A Single Lightweight Process per Process Is Not A Good Idea

Such an association effectively reduces to having only kernel-level threads, implying that much of the performance gain of having threads in the first place, is lost.

## 3.4 Advantages of Multi-threaded Server Compared to Single-threaded Server

- All the threads of a process share its resources thus making it more economical.
- Program responsiveness allows a program to run even if part of it is blocked using multithreading.
- Increase concurrency of the system since each thread can run on different processor.

## 3.5 Advantages and Disadvantages of Each of Three Models of Multi-threaded Server

### 3.5.1 Thread-per-request

**Advantages**

- The main advantage of thread-per-request is that it is straightforward to implement.
- This architecture is particularly useful to handle long-duration requests, such as database queries, from multiple clients.

**Disadvantages**

- Thread-per-request can consume a large number of OS resources if many clients make requests simultaneously.
- It is inefficient for short-duration requests because it incurs excessive thread creation overhead.
- It architectures are not suitable for real-time applications since the overhead of spawn a thread for each request can be non-deterministic.

### 3.5.2 Thread-per-connection

**Advantages**

- Thread-per-connection is straightforward to implement.
- It is well suited to perform long-duration conversations with multiple clients.

**Disadvantages**

- Thread-per-connection does not support load balancing effectively.
- For clients that make only a single request to each server, thread-per-connection is equivalent to the thread-per-request architecture.

### 3.5.3 Thread-per-object

**Advantages**

- Thread-per-object is useful to minimize the amount of rework required to multi-thread existing single-threaded servants given that all methods in a servant only access servant-specific state there is no need for explicit synchronization operations.

**Disadvantages**

- Thread-per-object does not support load balancing effectively since it serializes request processing in each servant.
- Therefore, if one servant receives considerably more requests than others it can become a performance bottleneck.
- It is prone to deadlock on nested callbacks.

## 3.6 Server Following Finite State Machine

### 3.6.1 Example

Node.js is an example of a server following Finite state machine

### 3.6.2 Reason for its High Scalability

While single threaded, Node.js is asynchronous so a lot of requests can run on the single thread very quickly.

## 3.7 The Need of Multi-threaded for Client in Distributed Systems

In a single-process, single-threaded environment all requests to, e.g., the I/O interface blocks any further progress in the process. Any combination of a multi-process or multi-threaded implementation of the library makes provision for the user of the client application to request several independent documents at the same time without getting blocked by slow I/O operations. **Example:** As a World-Wide Web client is expected to use much of the execution time doing I/O operation such as "connect" and "read", a high degree of optimization can be obtained if multiple threads can run at the same time.

## 3.8 The Need of Virtualization Technology

- Reduce Downtime
- Save Money and Time
- Better Recovery When Critical Error Happened

## 3.9 Reason Why X-Window system Suitable for Thin Client Architecture

- X is an architecture-independent system for remote graphical user interfaces and input device capabilities.
- Each person using a networked terminal has the ability to interact with the display with any type of user input device.
- X provides the basic framework, or primitives, for building such GUI environments: drawing and moving windows on the display and interacting with a mouse, keyboard or touchscreen

- A Thin Client Architecture is system where the clients (the application that runs on a personal computer or workstation and relies on a server to perform some operations) does very little. They basically just act as terminal for the server.
- This architecture allows disk-less client and make data storage more centralized.
- Since X is a windowing system, meaning it manages separated screens, it is suitable for thin client architecture as X-Window system does not require much client-side

## 3.10 Compare the Daemon Server and Superserver

### 3.10.1 Daemon

In multitasking computer operating systems, a daemon is a computer program that runs as a background process, rather than being under the direct control of an interactive user. It means that daemon will always run as long as the system is still up.
When finding a server, since the daemon is always running, you just need to call that particular daemon. However, many daemon may limit the performance of the hardware system. In addition, when needed, application may have to 'remember' the exact daemon for the job.

### 3.10.2 Superserver

Superserver is a kind of daemon that called necessary servers when they are needed.
When finding a server, client will call directly to superserver. The downside of this is that the response time might be high.

Chapter 4

# Chapter 4: Communication

## 4.1 Why in many layered protocols, each layer has its own header

- Each layer must be independent of the other ones.
- The data passed from the upper layer down to the lower layer contains both header and data, but the lower layer cannot tell which is which.
- Having a single big header that all the layers could read and write would make changes in the protocol of one layer visible to other layers, thus making the whole protocols unsafe.

## 4.2 Socket representation; Socket connection representation? Why do we need 4 values to represent a socket connection?

A Socket is represented by:

- address
- port

A Socket Connection is represented by;

- local-address
- local-port
- foreign-address
- foreign-port

Why do we need 4 values to represented a socket connection:

- Each socket has 2 fields: address and port
- A Socket Connection is defined by 2 sockets: local and foreign
- To represent a Connection, we need to represent that 2 sockets

## 4.3 Three characteristics of TCP protocol

- ***Connection-Oriented***: TCP requires that devices first establish a connection with each other before they send data. The connection creates the equivalent of a circuit between the units, and is analogous to a telephone call. A process of negotiation occurs to establish the connection, ensuring that both devices agree on how data is to be exchanged.

- ***Reliable***: Communication using TCP is said to be *reliable* because TCP keeps track of data that has been sent and received to ensure it all gets to its destination. As we saw in the previous topic, TCP can't really "guarantee" that data will always be received. However, it *can* guarantee that all data sent will be checked for reception, and checked for data integrity, and then retransmitted when needed. So, while IP uses "best effort" transmissions, you could say TCP *tries harder*, as the old rent-a-car commercial goes.
- ***Synchronous***: All transmissions in TCP are *acknowledged* (at the TCP layer—TCP cannot guarantee that all such transmissions are received by the remote application). The recipient must tell the sender *"yes, I got that"* for each piece of data transferred, which is different from typical messaging protocols where the sender never knows what happened to its transmission

On UDP, they are:

- Connection-less
- Unreliable: since it did not provide acknowledgement messages, it can not check whether the messages are received
- Does not provide acknowledgement messages

## 4.4 Two main issues of RPC

Issues:

- ***Marshalling***:Parameters must be marshalled into a standard representation.
- ***Binding***: How does the client know who to call, and where the service resides?

## 4.5 Consider a procedure incr with two integer parameters. The procedure adds ONE to each parameter. Now suppose that it is called with the same variable twice, for example, as incr(i, i). If i is initially 0, what value will it have afterward if call-by-reference is used? How about if copy/restore is used?

If call by reference is used, a pointer to *i* is passed to *incr*. It will be incremented two times, so the final result will be two. However, with copy/restore,$i$ will be passed by value twice, each value initially 0. Both will be incremented, so both will now be 1. Now both will be copied back, with the second copy overwriting the first one. The final value will be 1, not 2.

## 4.6 Explain why transient synchronous communication has inherent scalability problems, and how these could be solved

Problems:

- Scalability is the ability for a distributed application to expand without affecting any of the application algorithms.
- The transient synchronous communication would have issues pertaining to geographical scalability. If the client and the server are placed in two far-away geographical locations it would take a considerable amount of time for the client to send a request and receive a reply.
- Because the client remains idle until it receives the reply there is a considerable waste of time, since the client can only resume after receiving the reply. If the system is always in idle mode all of the time as it is synchronous there would not be any room for scalability.
- This issue is a performance issue which creates unnecessary network latency.
- There is set back when it comes to expanding of the system because it is transient system there is no storage. If there was to be a malfunction or failure of one component the whole system would be down. This is because the system is not persistent.

Solutions:

- We can use network optimization tools. These tools help to minimize the amount traffic being built up on the network.
- Another solution would be to change to a new upgraded network connection. A faster network connection is key factor to reduce the time taken to receive a reply for request sent to the server from the client.

## 4.7 Can we apply the persistent asynchronous communication to RPC? Explain it

Yes, but only on a hop-to-hop basis in which a process managing a queue passes a message to a next queue manager by means of an RPC.

Effectively, the service offered by a queue manager to another is the storage of a message. The calling queue manager is offered a proxy implementation of the interface to the remote queue, possibly receiving a status indicating the success or failure of each operation. In this way, even queue managers see only queues and no further communication.

## 4.8 In message-oriented transient communication, why the client doesn't need to call the bind primitive to bind its socket to a port?

Because the operating system can dynamically allocate a port when the connection is set up.

## 4.9 In message-oriented transient communication, explain the role of the two queues (completed and incomplete connection queue) maintained by TCP for a listening socket. Explain the backlog parameter of the function listen called by a TCP server.

Role of the two queues:

- Transient communication means the way by which the messages are not saved into a buffer to wait for its delivery at the message receiver, which means the messages will be delivered only if both the systems (sender and receiver) are running.
- The role of completed connections queue: store the connections that are ready to send message.
- The role of incomplete connections queue: store the connections that are not ready to send message.
- Together, they will prevent sending messages over when the receiver is not ready thus making the messages lost in the process

Role of *backlog* parameter of the function *listen* called by a TCP server

- The backlog argument specifies the maximum number of queued connections and should be at least 0; the maximum value is system-dependent (usually 5), the minimum value is forced to 0.
- In simple words, the backlog parameter specifies the number of pending connections the queue will hold.

## 4.10 What is the role of Message-Queuing System in persistent communication?

Roles:

- Offer the intermediate-term storage capacity for messages
- Target to support message transfers that are allowed to take minutes instead of seconds or milliseconds
- No guarantees about when or even if the message will be actually read
- Allow the sender and receiver can execute completely independently

## 4.11 Bit-rate, Delay and Jitter

1. Bit-rate
    - The number of bits that are conveyed or processed per unit of time
    - Used to determine the bandwidth (the maximum rate that information can be transferred
    - Higher bit-rate means better bandwidth thus could led to better quality of the content transfer through the channel
2. Delay
    - Also called Latency, is defined as the delay between the sender and the receiver decoding it
    - Mainly a function of the signals travel time, and processing time at any nodes the information traverses
    - Lower delay means better service. High latency can render an application with high priority on synchronization such as VoIP or online gaming unusable.
3. Jitter
    - Variation in packet delay at the receiver of the information.
    - Can be quantified in the same terms as all time-varying signals
    - High jitter may cause a display monitor to flicker, affect the performance of processors in personal computers, introduce clicks or other undesired effects in audio signals, and cause loss of transmitted data between network devices

## 4.12 How we can use a buffer at the receiver to reduce the jitter

- Jitter buffers eliminate jitter by queuing a number of packets to absorb the delay differences between them and then playing them out at a constant rate
- This gives the more delayed packets a chance to catch up so that all packets are output smoothly
- As with any type of buffering, the larger the jitter buffer is, the longer a packet's playout delay becomes

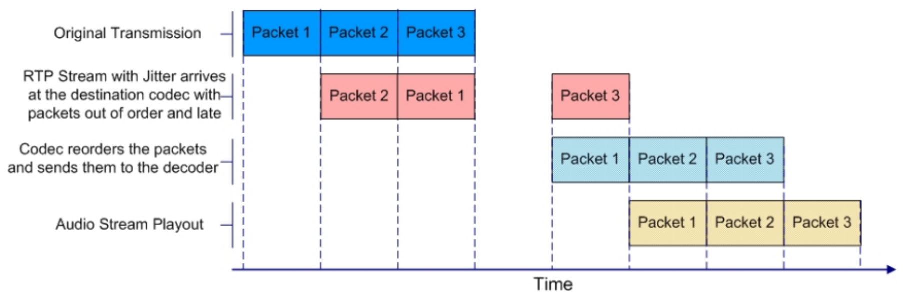
Original Transmission
Packet 1
Packet 2
Packet 3
RTP Stream with Jitter arrives at the destination codec with packets out of order and late
Packet 2
Packet 1
Packet 3
Codec reorders the packets and sends them to the decoder
Packet 1
Packet 2
Packet 3
Audio Stream Playout
Packet 1
Packet 2
Packet 3
Time

Figure 4.1: Visualization of how the buffer works

## 4.13 Forward error correction (FEC)

- Forward error correction (FEC) is an error correction technique to detect and correct a limited number of errors in transmitted data without the need for retransmission.
- The sender sends a redundant error-correcting code along with the data frame
- The receiver performs necessary checks based upon the additional redundant bits. If it finds that the data is free from errors, it executes error-correcting code that generates the actual frame then removes the redundant bits before passing the message to the upper layers.
- Error-correcting code can be broadly categorized into two types, namely, block codes and convolution codes
  - ***Block codes***: The message is divided into fixed-sized blocks of bits to which redundant bits are added for error correction

- ***Convolutional codes***: The message comprises of data streams of arbitrary length and parity symbols are generated by the sliding application of a Boolean function to the data stream

# Chapter 5

# Chapter 6: Synchronization

## 5.1 Give two examples to demonstrate the importance and the need of synchronization mechanism between processes in distributed systems

- Distributed System is a collection of computers connected via the high speed communication network.
- The hardware and software components communicate, coordinate their actions and share their resources to other nodes by message passing.
- There is need of proper allocation of resources to preserve the state of resources and help coordinate between the several processes.
- Example: If a distributed system is not synchronized, for example computer A is 10 minutes behind computer B and both computer is on the same timezone (say ICT) with computer B is synchronized with the standard time in timezone. At 7:00 am Hanoi time, computer A try to write something to the shared resources and register the timestamp as 7:00 Hanoi time, but since it is technically 7:10 am, the system will not let the user from A to write anything because the timestamp is invalid. In fact, users from computer A cannot write anything until the administrator synchronizes that machine. Timestamp invalidation can also caused lagging when using real time application or application that shared the resources to a lot of users like online games or chat applications.
- Benefit of synchronization through example: With synchronization, when playing online games for example, since most packages have to have a timestamp or a related time series to determine the player model action sequences, with synchronization, there can be less lag and enable better quality for player. In fact, nowadays, most online games required you to synchronized before playing.

## 5.2 Compare Network Time Protocol and Berkeley algorithm

| Berkeley Algorithm | Network Time Protocol |
|---|---|
| Having 1 Master to be standard | Having 3 classes with class 1 is the highest accuracy and class 3 is the lowest |
| Master calculate the average time and discard outliers | Class 1 is the standard. Class 2 get time from Class 1 and Class 2 server. Class 3 get time from any server when they attempted connecting to one |
| Send time adjustment to clients | Send timestamp to be changed into |
| Round trip message | One way message to dictate time |

## 5.3 What is the typical characteristic of synchronization algorithm for wireless networks

Since the primary concern of all wireless applications is energy conservation, the typical characteristics of synchronization algorithm for wireless networks are :

- It must carefully regiment its frequency of resynchronization, and avoid flooding.
- In addition, the algorithm cannot typically rely on a power-hungry source of real time such as GPS.
- Another characteristic of wireless networks is unexpected and possibly frequent changes in network topology. Thus, a CSA in a wireless medium must continue to function in the face of node failures and recoveries.
- Lastly, many applications in wireless settings are local in nature. That is, only nearby nodes in the network need to participate in some activity. Thus, a desirable property for a CSA is that it closely synchronizes nodes which are nearby, while possibly allowing faraway nodes to be more loosely synchronized.

## 5.4 What is the difference between physical synchronization and logical synchronization

| Physical Synchronization | Logical Synchronization |
|---|---|
| Synchronize the exact timestamp | Synchronize by order of action |
| Must change the clock in each systems based on a standard | No need to change the clock |
| Maintain the same notion of time | Keep track of information pertaining to the order of events |
| Expensive to maintain | Inexpensive to maintain |
| Inherently inaccurate | Fairly inherently accurate |

## 5.5 What are the update steps of counters to implement Lamport's logical clock

- Once the logical clock function has an event, it looks at the time on that event, and compares it to the time on the clock's process
- It chooses the larger of the options, and increments it arbitrarily

## 5.6 Answer questions based on an algorithm for the physical clock synchronization

a) Is the value of $T_P$ calculated by the above formula absolutely accurate?

- Given that there is a constant delay on the medium, the value $T_P$ can be consider absolutely accurate because $RTT$ is double the one way time in that condition and $T_P$ will be equal with the timer on server S at that time.
- In practice, the value $T_P$ is not absolutely accurate but it can be consider as accurate enough to use.

b) Let $\delta$ be the deviation of time value and *min* the minimum time value it takes to transmit a message one-way. Determine the value $\delta$ using only 2 variables $RTT$ and *min*

- If there is no deviation of time value, $RTT = 2 * min$
- The deviation of time value is the different between $RTT$ and two time the *min*. So the equation would be:

$$\delta = RTT - 2 * min$$

- $\delta$ should be positive but it can be negative when you upgrade to a better system and/or better medium. When that happened, you need to update your *min* value.

## 5.7 Answer the following questions in regard of using the Vector Clock concept for enforcing causal communication

a) List two conditions the receiving process use to check whether the message satisfies causality.

$$\begin{cases} V_{Pj}[i] = V_S[i] - 1 \\ V_{Pj}[k] \geq V_S[k] \forall k \in [1, 2, 3, ..., n] - \{i\} \end{cases}$$

b) Vector clock values for 4 points $X1$, $X2$, $X3$, and $X4$

$$X1 = (0, 1, 0)$$

$$X2 = (1, 1, 0)$$

$$X3 = (2, 1, 1)$$

$$X4 = (2, 1, 2)$$

c) Message will be kept at the middleware level: Message b

## 5.8 What is a mutual exclusion algorithm in a distributed system

- Mutual exclusion: Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner.
- Only one process is allowed to execute the critical section (CS) at any given time.
- In a distributed system, shared variables (semaphores) or a local kernel cannot be used to implement mutual exclusion.
- Message passing is the sole means for implementing distributed mutual exclusion.
- Distributed mutual exclusion algorithms is the family of algorithms that deals with unpredictable message delays and incomplete knowledge of the system state.

## 5.9 What is the drawback of the centralized algorithm for the mutual exclusion

- If the coordinator crashes, the entire system may go down with it since it is a single point of failure.
- If processes normally block after making a request, they cannot distinguish a dead coordinator from "permission denied" since no message comes back.
- Bottleneck when deploy on the large system with a single coordinator (scalability)

## 5.10 What is the drawback of the distributed algorithm for the mutual exclusion

- Algorithm is suitable only for small group of processes.
- It is highly complex due to the need of identify all processes needed.

## 5.11 Propose a solution for the problem of lost token in Token Ring mutual exclusion algorithm

When a process (say) P1 wants to enter into its critical section, it sends request to the coordinator. If the coordinator retains the token, it then sends the token to the requesting process (P1). After getting the token, the process will send an acknowledgment to the coordinator, and enters the critical section. During the execution of the critical section, P1 will continually send an EXISTS signal to the coordinator at certain time interval, so that, the coordinator becomes acquainted that the token is alive and it has not lost. As a reply to every EXISTS signal, the coordinator sends back an OK signal to that particular process (P1), so that the process that is executing in its critical section (P1), gets to know that the coordinator is alive also. Now, suppose, the coordinator is not receiving the EXISTS signal from that process P1. Here two cases may appear:

1. The coordinator assumes that the token has lost. Then the coordinator will regenerate a new token and sends it to that process (P1) and again it starts executing its critical section.
2. The process P1 may crash or fail while executing in it critical section and consequently, the coordinator does not receiving any EXISTS signal from P1. Hence, the coordinator will identify it as a crashed process and update the ring configuration table and send the UPDATE

signal with update information to other processes to update their own ring configuration tables.

Again it may be the case, that the process P1 (which is currently executing in its critical section), is not receiving the OK signal form the coordinator. So, P1 would assume that the coordinator is somehow crashed. At this moment, the process P1 will become the new coordinator and complete the critical section execution. The new coordinator will send a message [COORDINATOR (PID)] to every other process, that it becomes the new coordinator and send the UPDATE signal with update information, to update the ring configuration tables maintained by all other processes.
The algorithm also overcomes the overhead of token circulation in the ring. If no processes in the ring want to enter in its critical section, then there is no meaning of circulating the token throughout the ring. Rather, in this approach, the coordinator will keep the token, until any other process requests it.

1. The algorithm guarantees mutual exclusion, because at any instance of time only one process can hold the token and can enter into the critical section.

2. Since a process is permitted to enter one critical section at a time, starvation cannot occur.

## 5.12 How many messages does the system need to vote the coordinator

When P3 starts the election after node P4 and P7 was broken, it will take the system a total of 10 messages to elect a new coordinator.

- The first round, P3 sent 4 ELECTION messages to P4, P5, P6 and P7. P3 would receive 2 ANSWER messages from P5, P6.

- The second round, P5 sent 2 ELECTION messages to P6 and P7 and received 1 ANSWER message from P6.

- The final round, P6 sent 1 ELECTION message to P7 and received 0 ANSWER message thus making it the coordinator.

## 5.13 Answer questions with a system the has N nodes and each node has a status table of other nodes and the whole system on election algorithm

a) How many messages do we need to vote the coordinator

$$number\_of\_messages = 2(N - i - 1) + (i + 1)N$$

b) When a broken node become working again

- That broken node first broadcast a WORKING message to each of the other nodes for update their tables. The other working nodes will send a acknowledge message back for that previously broken nodes to update its table.
- If the said node has higher id than the current coordinator, the coordinator will sent the ELECTION message to said node, otherwise, the system will continue to run

## 5.14 Answer questions regrading a mutual exclusion algorithm

a) How many messages does the system need to successfully let a process use SR.

- Suppose there are $k$ REQUEST messages on the queue already $0 \leq k$ (no upper limit due to Pi can post multiple REQUEST messages). Pi will wait $k$ RELEASE messages.
- Pi sends $N - 1$ REQUEST messages and receives $N - 1$ REPLY messages.
- The total amount of messages would be:

$$k + 2(N - 1)$$

b) The improvement will cut all the unnecessary REPLY messages since the purpose of the REPLY was to reorder the priority queues of all processes to timestamp order. The new algorithms can cut at most $N - 1$ REPLY messages. In the worst case scenario, when $ts_i$ is the latest, it will perform like the original algorithms.