RGPVNOTES.IN

Subject Name: **Distributed System**

Subject Code: **IT-6005**

Semester: **6th**

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in

Downloaded from be.rgpvnotes.in

RGPVNOTES.IN

Notes of Distributed System

UNIT 1

Characterization of Distributed Systems:

Introduction:

A distributed system is a collection of independent computers that appear to the users of the system as a single system.

Examples:

- Network of workstations
- Distributed manufacturing system (e.g., automated assembly line)
- Network of branch office computers

Other typical properties of distributed systems include the following:

- The system has to tolerate failures in individual computers.
- The structure of the system (network topology, network latency, number of computers) is not known in advance, the system may consist of different kinds of computers and network links, and the system may change during the execution of a distributed program.
- Each computer has only a limited, incomplete view of the system. Each computer may know only one part of the input.

Applications of Distributed System:

- Automated Banking Systems
- Tracking Roaming Cellular
- Telephones
- Air-Traffic Control Systems
- Retail Point-of-Sale Terminals
- Global Positioning Systems
- The World-Wide Web?
- Research and development projects

RGPVNOTES.IN

Resource sharing and the Web Challenges

The resource sharing can be done by client server model where the clients are active & server are passive. The client sends the request to the server then it gives the response. If the server is off then the request is terminates to other server.

Web challenges of distributed system are:

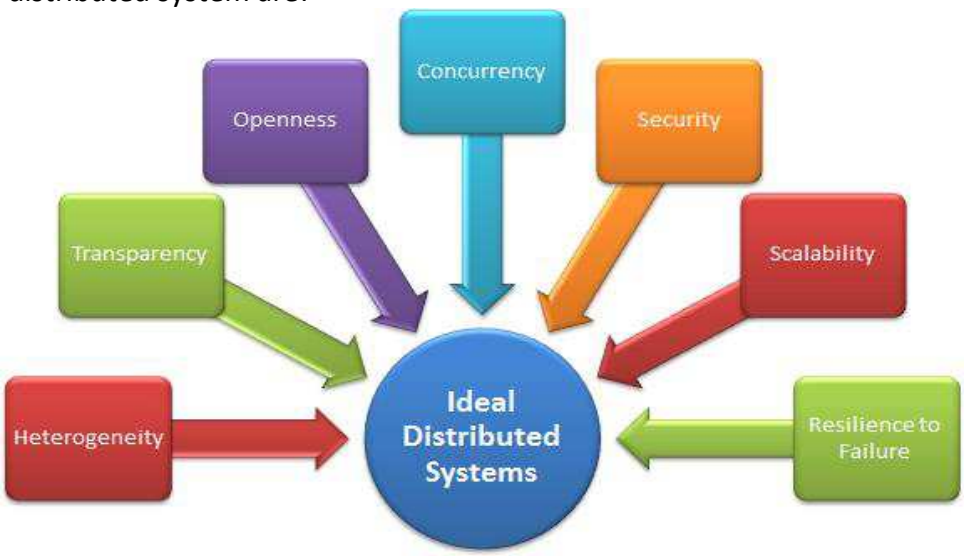


Fig 1.1 Challenges of Distributed System

Follow us on facebook to get real-time updates from RGPV

Downloaded from be.rgpvnotes.in

RGPVNOTES.IN

Designing the distributed systems does not come for free. Some challenges need to be overcome in order to get the ideal systems shown in fig 1.1.

1. Heterogeneity

This term means the diversity of the distributed systems in terms of hardware, software, platform, etc. Modern distributed systems will likely span different:

- Hardware devices: computers, tablets, mobile phones, embedded devices, etc.
- Operating System: Ms Windows, Linux, Mac, Unix, etc.
- Network: Local network, the Internet, wireless network, satellite links, etc.
- Programming languages: Java, C/C++, Python, PHP, etc.
- Different roles of software developers, designers, system managers

2. Openness:

If the well-defined interfaces for a system are published, it is easier for developers to add new features or replace sub-systems in the future. Example: Twitter and Facebook have API that allows developers to develop theirs own software interactively.

3. Transparency

Distributed systems designers must hide the complexity of the systems as much as they can. Adding abstraction layer is particularly useful in distributed systems. While users hit search in google.com, they never notice that their query goes through a complex process before google shows them a result. Some terms of transparency in distributed systems are:

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be copied in several places
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or a disk

4. Concurrency

Distributed Systems usually is multi-users environment. In order to maximize concurrency, resource handling components should be anticipated as they will be accessed by competing users. Concurrency is a tricky challenge, and then we must avoid the system's state from becoming unstable when users compete to view or update data.

5. Security

Every system must consider strong security measurement. Distributed Systems somehow deals with sensitive information; so secure mechanism must be in place.

6. Scalability

Distributed systems must be scalable as the number of user increases.

7. Resilience to Failure

Distributed Systems involves a lot of collaborating components (hardware, software, communication). So there is a huge possibility of partial or total failure.

System Models:

1. Architectural System model:

Architectural model describes responsibilities distributed between system components and how are these components placed.

a) Client–server: The system is structured as a set of processes, called servers that offer services to the users, called clients.

Follow us on facebook to get real-time updates from RGPV

Downloaded from be.rgpvnotes.in

RGPVNOTES.IN

- The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls (RPC) or remote method invocation (RMI):
- The client sends a request (invocation) message to the server asking for some service;
- The server does the work and returns a result (e.g. the data requested) or an error code if the work could not be performed.

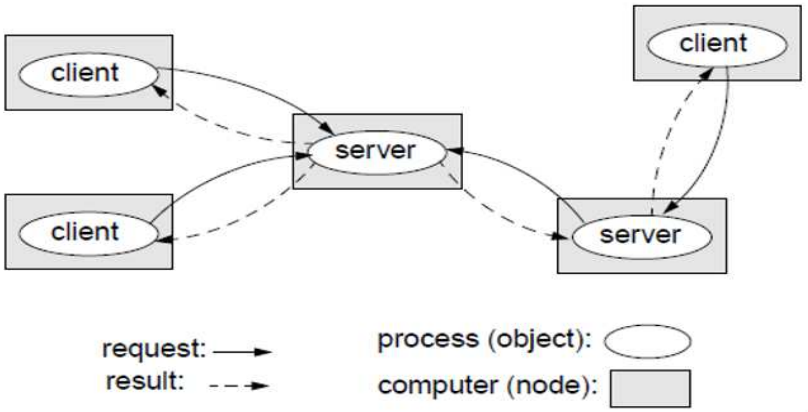


Fig 1.2 client server architecture

A server can itself request services from other servers; thus, in this new relation, the server itself acts like a client shown in fig 1.2..

b) Peer-to-peer

All processes (objects) play similar role.

- Processes (objects) interact without particular distinction between clients and servers.
- The pattern of communication depends on the particular application.
- A large number of data objects are shared; any individual computer holds only a small part of the application database.
- Processing and communication loads for access to objects are distributed across many computers and access links.
- This is the most general and flexible model. Shown in fig 1.3.

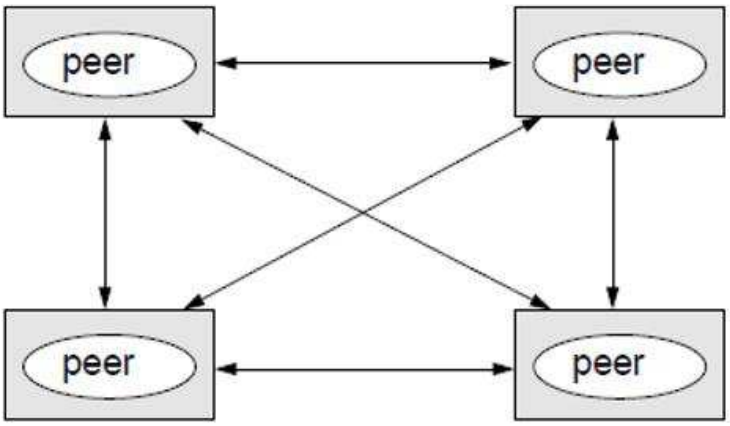


Fig 1.3 peer to peer architecture

Problems with peer-to-peer:

- High complexity due to
- Cleverly place individual objects
- retrieve the objects
- Maintain potentially large number of replicas.

Three-tier: architectures that move the client intelligence to middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are three-tier.

Follow us on facebook to get real-time updates from RGPV

Downloaded from be.rgpvnotes.in

RGPVNOTES.IN

3-tier architecture is a very common architecture. 3-tier architecture is typically split into a presentation or GUI tier, an application logic tier, and a data tier shown in fig 1.4.

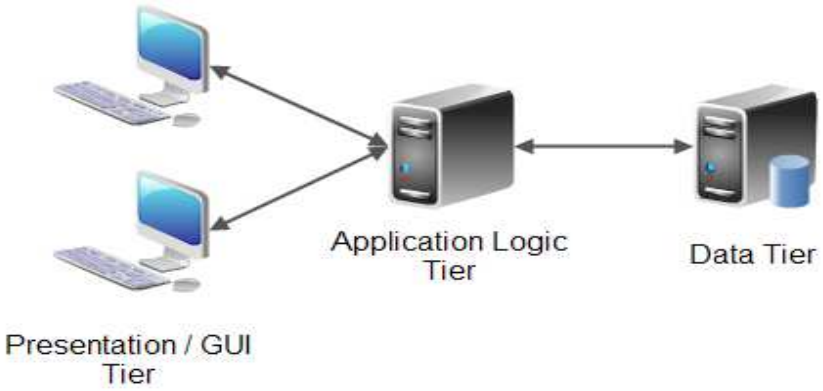


Fig 1.4 3-tier architecture

The presentation or GUI tier contains the user interface of the application. The presentation tier is "dumb", meaning it does not make any application decisions. It just forwards the user's actions to the application logic tier. If the user needs to enter information, this is done in the presentation tier too.

The application logic tier makes all the application decisions. This is where the "business logic" is located. The application logic knows what is possible, and what is allowed etc. The application logic reads and stores data in the data tier.

The data tier stores the data used in the application. The data tier can typically store data safely, perform transactions, search through large amounts of data quickly etc.

Example: Web and Mobile Applications

Web applications are a very common example of 3 tier applications. The presentation tier consists of HTML, CSS and JavaScript, the application logic tier runs on a web server in form of Java Servlets, JSP, ASP.NET, PHP, Ruby, Python etc., and the data tier consists of a database of some kind (mysql, postgresql, a noSQL database etc.) shown in fig 1.5.

Fig 1.5 example of 3 tier architecture

Actually, it is the same principle with mobile applications that are not standalone applications. A mobile application that connects to a server typically connects to a web server and sends and receives data. For typical 3 tier mobile application diagram shown in fig 1.6:

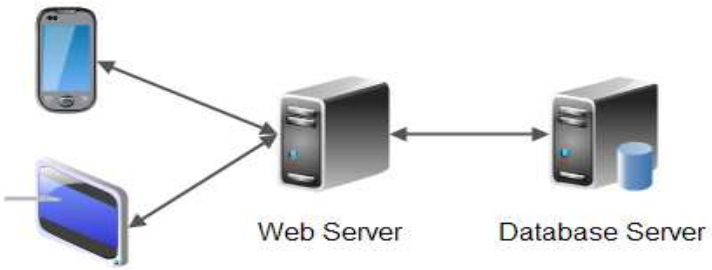


Fig 1.6 example of 3 tier architecture

Follow us on facebook to get real-time updates from RGPV

Downloaded from be.rgpvnotes.in

RGPVNOTES.IN

N-tier: N tier architecture means splitting up the system into N tiers, where N is a number from 1 and up. A 1-tier architecture is the same as a single process architecture. A 2-tier architecture is the same as a client / server architecture etc.

Fundamental Models

Description of properties those are present in all distributed architectures.

- Interaction Models – Issues dealing with the interaction of process such as performance and timing of events.
- Failure Models – Specification of faults that can be exhibited by processes and communication channels.
- Security Models – Threats to processes and communication channels

Theoretical Foundation for Distributed System

Limitation of distributed system:

A distributed system is a set of computers that communicate over a network, and do not share a common memory or a common clock

1. **Absence of a common (global) clock:**

Global clock cannot be solved the problem for the following reasons:

- Suppose a global (common) clock is available for all the processes in the system, two different processes can observe a global clock value at different instants due to unpredictable message transmission delays.
- If we provide each computer in the system with a physical clock and try to synchronize them, these physical clocks can drift from the physical time and the drift rate may vary from clock to clock due to technological limitations.

Impact of the absence of global time

- Difficult to reason about the temporal order of events in a distributed system (scheduling processes).
- Difficult to design and debug compared to algorithms for centralized systems.
- Harder to collect up-to-date information on the state of the entire system.

2. **Absence of shared memory:**

- Since the computations in a distributed system do not share common memory, an up-to-date state of the entire system is not available to any individual process.
- A process in a distributed system can obtain a coherent but partial view of system or a complete but incoherent view of the system. A view is said to be coherent if all the observations of different processes (computers) are made at the same physical time

Distributed shared memory:

Distributed Shared Memory (DSM) is a resource management component of a distributed operating system that implements the shared memory model in distributed systems, which have no physically shared memory. The shared memory model provides a virtual address space that is shared among all computers in a distributed system. An example of this layout can be seen in the fig 1.7.

In DSM, data is accessed from a shared address space similar to the way that virtual memory is accessed. Data moves between secondary and main memory, as well as, between the distributed main memories of different nodes. Ownership of pages in memory starts out in some pre-defined state but changes during the course of normal operation. Ownership changes take place when data moves from one node to another due to an access by a particular process.

Follow us on facebook to get real-time updates from RGPV

Downloaded from be.rgpvnotes.in

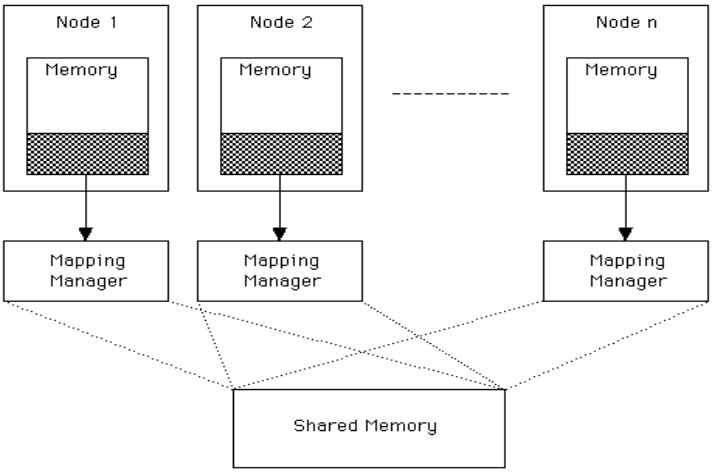


Fig 1.7 shared memory

Advantages of Distributed Shared Memory:

- Hide data movement and provide a simpler abstraction for sharing data. Programmers don't need to worry about memory transfers between machines like when using the message passing model.
- Allows the passing of complex structures by reference, simplifying algorithm development for distributed applications.
- Takes advantage of "locality of reference" by moving the entire page containing the data referenced rather than just the piece of data.
- Cheaper to build than multiprocessor systems. Ideas can be implemented using normal hardware and do not require anything complex to connect the shared memory to the processors.
- Larger memory sizes are available to programs, by combining all physical memory of all nodes. This large memory will not incur disk latency due to swapping like in traditional distributed systems.
- Unlimited number of nodes can be used. Unlike multiprocessor systems where main memory is accessed via a common bus, thus limiting the size of the multiprocessor system.
- Programs written for shared memory multiprocessors can be run on DSM systems,

Logical clocks

A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Distributed systems may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.

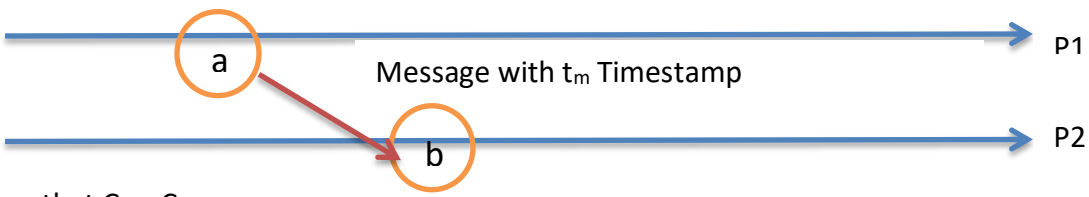
In logical clock we need to find a relation between two events that is called “happened before” relation.

Example: p is a process and it has two events a, b then there is a two case two defined a relation between both events:

Case 1: a and b both events belong to same process p1 and happened like that:

So we can say that $C_a < C_b$

Case 2: If both events (a, b) belong to the different process p1 and p2 accordingly and event a send a message to event b then:



So we can say that $C_a < C_b$

Follow us on facebook to get real-time updates from RGPV

Downloaded from be.rgpvnotes.in

Three Terminology Defined by "Happened before" relation:

1. Transitive relation: If a,b and c are the event of process p then

   If a<b and b<c then we can say that a<c

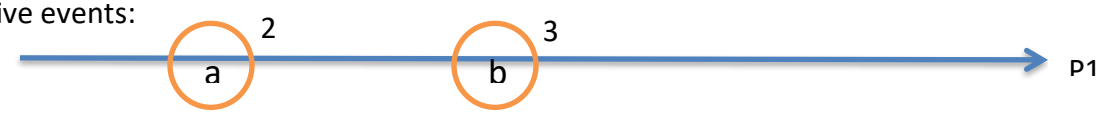
2. Casually ordered event: if a<b or a->b then we can say that a is casually effected b. Means whatever changes happened in a that will effect b.
3. Concurrent effects: if a!->b and b!->a than we can say that both events are concurrent or happened simultaneously.

Lamport's clock:

The algorithm of Lamport timestamps is a simple algorithm used to determine the order of events in a distributed computer system. As different nodes or processes will typically not be perfectly synchronized, this algorithm is used to provide a partial ordering of events with minimal overhead.

Implementation rules:

1. Successive events:



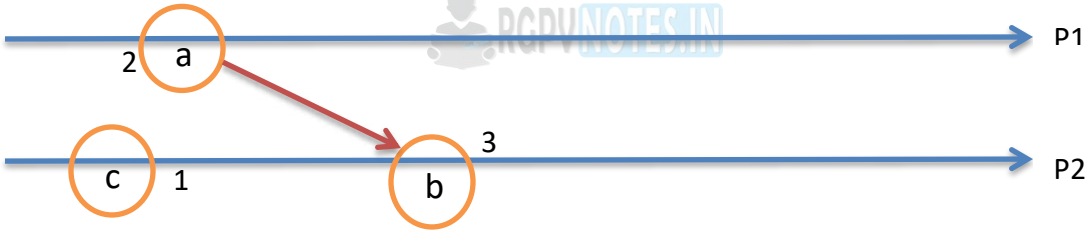
   Here a and b are the events of the same process and both are the successive events so if time stamp of the a is 2 then for b it will be (increment by 1) 3.

   Means $C_i = C_i + 1$

2. Message M sent from one process to another process:

   If Sent message time stamp is t_m then time stamp of the receiving end is:

   $Max(C_i , t_m) + 1$



Algorithm

The algorithm follows some simple rules:

1. A process increments its counter before each event in that process;
2. When a process sends a message, it includes its counter value with the message;
3. On receiving a message, the counter of the recipient is updated, if necessary, to the greater of its current counter and the timestamp in the received message. The counter is then incremented by 1 before the message is considered received.

In a Pseudocode format, the algorithm for sending:

```
time = recive_time+1;    //recive_time represent is a time_samp of previous event
time_stamp = time;
send(message, time_stamp);
```

The algorithm for receiving a message:

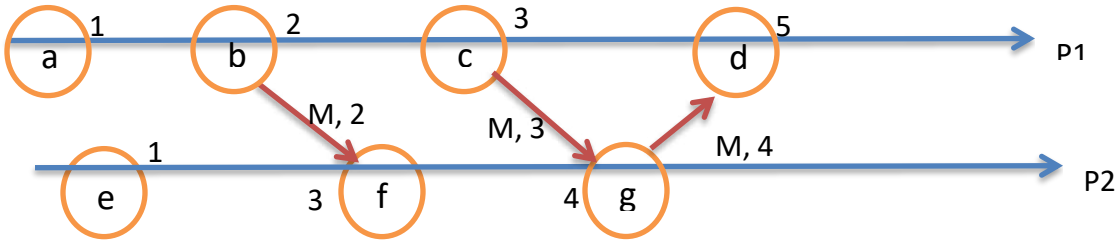
```
(message, time_stamp) = receive();
time = max(time_stamp, time)+1;
```


Follow us on facebook to get real-time updates from RGPV

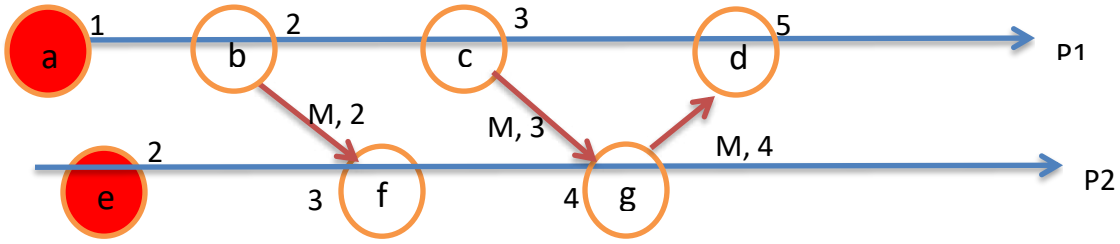
Downloaded from be.rgpvnotes.in

RGPVNOTES.IN

Example:



Limitation of lamport's clock:



Shown in space-time diagram we can see that c(a)<c(e) or time_stamp(a) < time_stamp(e) , but we cannot tell which one are executed first , because there is no relation between event a nad e.

Vector clock:

Vector Clocks are used in distributed systems to determine whether pairs of events are causally related. Using Vector Clocks, timestamps are generated for each event in the system, and their causal relationship is determined by comparing those timestamps.

The timestamp for an event is an n-tuple of integers, where n is the number of processes.

Each process assigns a timestamp to each event. The timestamp is composed of that processes logical time and the last known time of every other process.

Example:

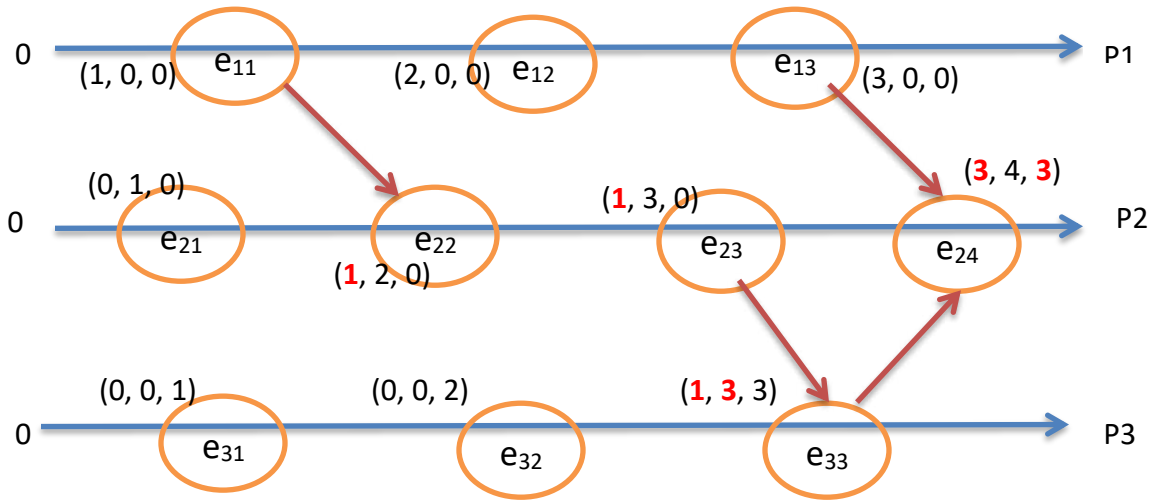
Timestamp for P2:

(P1_Last_Known_Time, P2_Logical_Time, P3_Last_Known_Time)

How Vector Clock Timestamps Are Assigned

1. **Events:** Every time an event is generated, a process increments its clock and assigns a timestamp to the event based on its knowledge of all the clocks in the system.
2. **Sending messages:** When a message is sent the timestamp of the sending event is given to the message.
3. **Receiving messages:** When a message is received, the process updates its knowledge of the system clock states by taking the maximum of each component of the message timestamp and its current knowledge of the system clock states. The receiving event gets this new timestamp.

Example:



Follow us on facebook to get real-time updates from RGPV

Downloaded from be.rgpvnotes.in

RGPVNOTES.IN

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 must deal with unpredictable message delays and incomplete knowledge of the system state.

Classification of Mutual Exclusion Algorithm:

The problem of mutual exclusion has received considerable attention and several algorithms to achieve mutual exclusion in distributed system have been proposed.

They tend to differ in their communication topology and in the amount of information maintained by each site about other site these algorithms can be grouped into two classes:

1. **Non-token based:**
   These algorithm require two or more successive rounds of message exchanges among the site. These algorithms are assertion based because a site can enter in critical section when an assertion defined on its local variable becomes true.
2. **Token based:**
   In these algorithms a unique token (also known as the PRIVILEGE message) is share among the site. A site is allowed to hold the token until the execution of the CS is over

Requirements of Mutual exclusion algorithm:

The primary objective of the mutual exclusion algorithm is to maintain mutual exclusion. That is, only on process can access the critical section at a time. In addition the following characteristics are considered important in a mutual exclusion algorithm.

a) **Freedom from Deadlock:** Two or more site should be forced to wait indefinitely to execute CS while other sites are repeatedly executing CS. That is very requesting sites should get an opportunity to execute CS in a finite time.

b) **Freedom from starvation:** A site should not be forced to wait indefinitely to execute CS while other sites are repeatedly executing CS. That is even requesting sites should get an opportunity to execute CS in a finite time.

c) **Fairness:** Fairness dictates that request must be executed in the order they are made. Since a physical clock does not exist, time is determined by logical clock. Note that fairness implies freedom from the starvation but not vice-versa.

d) **Fault Tolerance:** A mutual exclusion algorithm is fault-tolerant if in the wake up the failure it can be recognize itself so that it continues to function without any interruption.

1. NON-TOKEN BASED ALGORITHM:

a) Center coordinator Algorithm:

The central server algorithm simulates a single processor system. One process in the distributed system is elected as the coordinator (shown in Fig 1.8). When a process wants to enter a critical section, it sends a request message (identifying the critical section, if there is more than one) to the coordinator. If nobody is currently in the section, the coordinator sends back a grant message and marks that process as using the critical section. If, however, another process has previously claimed the critical section, the server simply does not reply, so the requesting process is blocked. When a process is done with its critical section, it sends a release message to the coordinator. The coordinator then can send a grant message to the next process in its queue of processes requesting a critical section (if any). This algorithm is easy to implement and verify.

Follow us on facebook to get real-time updates from RGPV

Downloaded from be.rgpvnotes.in

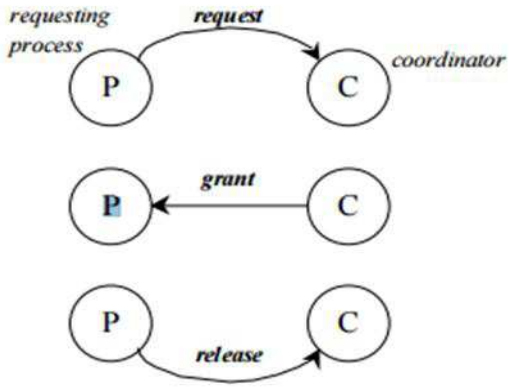


Fig 1.8 Center coordinator Algorithm

It's fair in that all requests are processed in order. Unfortunately, it suffers from having a single point of failure. A process cannot distinguish between being blocked (not receiving a grant because someone else is in the critical section) and not getting a response because the coordinator is down. Moreover, a centralized server can be a bottleneck in large systems.

b) Ricart & Agrawala Algorithm:

Ricart & Agrawala came up with a distributed mutual exclusion algorithm in 1981. It requires the following:

- Total ordering of all events in a system (e.g. Lamport's algorithm or others).
- Messages are reliable (every message is acknowledged).

When a process wants to enter a critical section, it:

1) Composes a message containing {its identifier(machine ,proc#), name of critical section, current time).
2) Sends a request message to all other processes in the group (may use reliable group communication).
3) Wait until everyone in the group has given permission.
4) Enter the critical section.

When a process receives a request message, it may be in one of three states:

Case 1: The receiver is not interested in the critical section, send reply (OK) to sender.

Case 2: The receiver is in the critical section; do not reply and add the request to a local queue of requests.

Case 3: The receiver also wants to enter the critical section and has sent its request. In this case, the receiver compares the timestamp in the received message with the one that it has sent out. The smallest (earliest) timestamp wins. If the receiver is the loser, it sends a reply (OK) to sender. If the receiver has the earlier timestamp, then it is the winner and does not reply. Instead, it adds the request to its queue

When the process is done with its critical section, it sends a reply (OK) to everyone on its queue and deletes the processes from the queue.

As an example of dealing with contention, consider Fig 1.9. Here, two processes. 0 and 2, requested to access the same resource (critical section). Process 0 sends its request with a timestamp of 8 and process 2 sends its request with a timestamp of 12 (fig 1.9 a). Since process 1 is not interested in the critical section, it immediately sends back permission to both 0 and 2. Process 0 is interested in the critical section. It sees that process' 2 timestamp was later than its own, so process 0 wins. It queues a request from 2 (fig 1.9 b). Process 2 is also interested in the critical section. When it compares its message's timestamp with that of the message it received from process 0, it sees that it lost, so it replies with permission to process 0 and continues to wait for everyone to give it permission to enter the critical section (fig 1.9 c). As soon as process 2 send process 0 permission, process 0 received permission from the entire group and can enter the critical section. When process 0 is done with the critical section, it examines its queue of pending permissions, finds process 2 in that queue, and sends it permission to enter the critical section (fig 1.9 d). Now process 2 has received permission from everyone and can enter the critical section.

Follow us on facebook to get real-time updates from RGPV

Downloaded from be.rgpvnotes.in

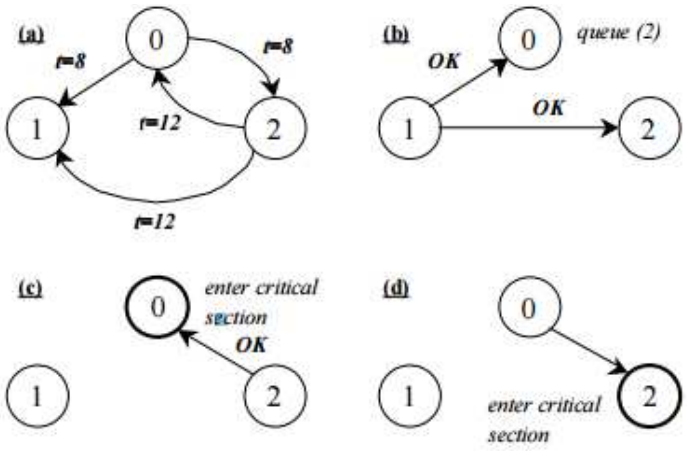


Fig 1.9 Ricart & Agrawala Algorithm

Issues and solution: One problem with this algorithm is that a single point of failure has now been replaced with n points of failure. A poor algorithm has been replaced with one that is essentially n times worse. All is not lost. We can patch this omission up by having the sender always send a reply to a message either an OK or a NO. When the request or the reply is lost, the sender will time out and retry. Still, it is not a great algorithm and involves quite a bit of message traffic but it demonstrates that a distributed algorithm is at least possible.

2. Token- Based Algorithm:

a) Token Ring Algorithm:

For this algorithm, we assume that there is a group of processes with no inherent ordering of processes, but that some ordering can be imposed on the group. For example, we can identify each process by its machine address and process ID to obtain an ordering. Using this imposed ordering, a logical ring is constructed in software. Each process is assigned a position in the ring and each process must know who is next to it in the ring (shown in fig 1.10).

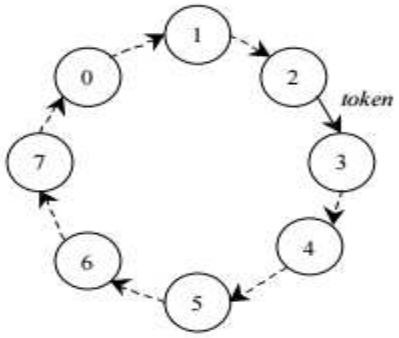


Fig 1.10 Token Ring algo

- The ring is initialized by giving a token to process 0. The token circulates around the ring (process n passes it to (n+1) mod ring size.
- When a process acquires the token, it checks to see if it is attempting to enter the critical section. If so, it enters and does its work. On exit, it passes the token to its neighbor.
- If a process isn't interested in entering a critical section, it simply passes the token along.

Only one process has the token at a time and it must have the token to work on a critical section, so mutual exclusion is guaranteed. Order is also well-defined, so starvation cannot occur.

The biggest drawback of this algorithm is that if a token is lost, it will have to be generated. Determining that a token is lost can be difficult.

Follow us on facebook to get real-time updates from RGPV

Downloaded from be.rgpvnotes.in

RGPVNOTES.IN

b) The Ring Based Algorithm:

The ring algorithm uses the same ring arrangement as in the token ring mutual exclusion algorithm, but does not employ a token. Processes are physically or logically ordered so that each knows its successor.

- If any process detects failure, it constructs an election message with its process I.D. (e.g. network address and local process I.D.) and sends it to its successor.
- If the successor is down, it skips over it and sends the message to the next party. This process is repeated until a running process is located.
- At each step, the process adds its own process I.D. to the list in the message.

Eventually, the message comes back to the process that started it:

- The process sees its ID in the list.
- It changes the message type to coordinator.
- The list is circulated again, with each process selecting the highest numbered ID in the list to act as coordinator.
- When the coordinator message has circulated fully, it is deleted.

Multiple messages may circulate if multiple processes detected failure. This creates a bit of overhead but produces the same results.

RGPVNOTES.IN

Follow us on facebook to get real-time updates from RGPV

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in