UNIT-1

CHARACTERIZATION OF DISTRIBUTED SYSTEM

INTRODUCTION : A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages.

Following characteristics of distributed system :

- Concurrency of components
- Lack of a global clock.
- independent failures of components.

Examples of Distributed System :

1. The Internet
2. Intranet
3. Mobile and ubiquitous computing.

RESOURCE SHARING AND THE WEB :

1. World Wide Web : is an evolving system for publishing and accessing resources and services across the Internet.

- Web is an open system.
- First its operation is based on communication standards and document standards that are freely published and widely implemented.
- Second, the Web is open with respect to the types of resources that can be published and shared on it. At its simplest, a resource on the web is a web page or some other types of content that can be stored in a file and presented to the user, such as program files, media files and documents in PostScript or Portable Document Format.
- The Web is based on three main standard technological components:
  * The HyperText MarkUp Language (HTML) is a language for specifying the contents and layout of pages as they are displayed by web browsers.

* Uniform Resource Locators (URLs), which identify documents and other resources stored as part of the Web.

* A client-server system architecture, with standard rules for interaction (HyperText Transfer Protocol - HTTP) by which browser and other client fetch document.

CHALLENGES :

- Heterogeneity : The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Heterogeneity applies to all of the following :
  - network
  - computer hardware.
  - operating system
  - programming languages.
  - implementations by different developers.

Solution to Heterogeneity

(1) Middleware : Middleware provides a uniform computational model for use by the programmers of servers and distributed applications. Possible models include remote object invocation, remote event notification, remote SQL access and distributed transaction processing. For example CORBA (Common Object Request Broker Architecture)

CORBA : It provides remote object invocation, which allows an object in a program running on one computer to invoke a method of an object in a program running on another computer.

(2) Mobile code : It is used to refer to code that can be sent from one computer to another and run at the destination — Java applets are an example. Code suitable for running on one computer is not necessarily suitable for running on,

another because executable programs are normally specific both to the instruction set and to the operating system of host.

The Virtual Machine approach provides a way of making code executable on any hardware : the compiler for a particular language generates code for a virtual machine instead of a particular machine.

- Openness : It determines whether the system can be extended and re-implemented in various ways. The openness of distributed system is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs. Openness cannot be acheived unless the ~~applicati~~ specification and documentation of the key software interface of the components of a

system are made available to software developers. In a word, the key interfaces are published.

The publication of interfaces is only the starting point for adding and extending services in the distributed system.

The challenges to designers is to tackle the complexity of distributed system consisting of many components engineered by different people. The designers of the Internet protocols introduced a series of documents called 'Requests For Comments' or RFC.

Systems that are designed to support resource sharing are termed open distributed systems.

- Security : Security for information resources has three components : confidentiality (protection against disclosure to unauthorized individual); integrity (protection against alternation or corruption); and availability (protection against interference with the means to access the resources)

In a distributed system, clients send requests to access data managed by servers, which involves sending information in messages over a network. Security is not just a matter of concealing the contents of messages - it also involves knowing for sure the identity of the user or other agent on whose behalf a message was sent. These challenges can be met by the use of encryption techniques. However, the following two security challenges have not yet been fully met :

* Denial of service attacks : User may wish to disrupt a service for some reason. This can be acheived by bombarding the service with such a large number of pointless requests that the serious users are unable to use it.

* Security of Mobile Code :

- Scalability : Distributed systems operates effectively and efficiently at many different scales, ranging from a small intranet to the Internet. A system is described as scalable it it will remain effective when there is a significant increase in the number of resources and the number of users.

The design of scalable distributed system presents the following challenges :

- Controlling the cost of physical resources.
- Controlling the performance loss.
- Preventing software resources running out.
- Avoiding performance bottlenecks.

- Failure handling :

Failures in a distributed system are partial — that is some components fail while others continue to function. The following techniques for dealing with failures are :

* Detecting failures : Some failures can be detected. For example, checksums can be used to detect corrupted data in a message or a file.

* Masking failures : Some failures that have been detected can be hidden or made less severe. For eg. ① messages can be retransmitted when they fail to arrive. ② file data can be written to a pair of disks so that it one is corrupted, the other may still be correct.

* Tolerating failures : It would not be practical to detect and hide all of the failures. Their clients can be designed to tolerate failures, which generally involves the users tolerating them as well.

- Recovery from failures : Recovery involves the design of software so that the state of permanent data can be recovered or rolled back.

* Redundancy : Services can be made to tolerate failures by the use of redundant components. Consider the following examples :

1. There should always be at least two different routes between any two routers in the Internet.
2. In the Domain Name System, every name table is replicated in at least two ~~router~~ different servers.

* Concurrency : For an object to be safe in a concurrent environment, its operation must be synchronized in such a way that its data remains consistent. This can be achieved by standard techniques such as semaphores.

- Transparency : Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independant components.

* Access transparency enables local and remote resources to be accessed using identical operations.

* Location transparency enables resources to be accessed without knowledge of their physical or network location.

* Concurrency transparency enables several processes to operate concurrently using shared resources without interference between them.

* Replication transparency enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

* Failure transparency enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.

* Mobility transparency allows the movement of resources and clients within a system without affecting the operation of users or programs.

* Performance transparency allows the system to be reconfigured to improve performance as loads vary.

* Scaling transparency allows the system and applications to ~~extend~~ expand in scale without change to the system structure or the application algorithms.

SYSTEM MODELS

"An architectual model of a Distributed system is concerned with the placement of its parts and the relationships between them."

For eg. client-server model and the peer to peer model.

There is no global time in a distributed system, so the clocks on different computers do not necessarily give the same time as one another. All communication between processes is acheived by means of messages. Message communication over a computer network can be affected by delays, can suffer from a variety of failures and is vulnerables to security attacks. These issues are addressed by three models:

- The interaction models with performance and with difficulty of setting time limits in a distributed system, for example for message delivery.

- The failure model attempts to give a precise specification of the faults that can be exhibited by processes and communication channels. It defines reliable communication and correct processes.
- The security model discusses the possible threats to processes and communication channels. It introduces the concept of a secure channel, which is secure against those threats.

Architectual Models :

The architecture of a system is its structure in terms of separately specified components.

An architectural model of a distributed system first simplifies and abstracts the functions of the individual components of a distributed system and then it considers:

- the placement of the components across a network of computers — seeking to define useful patterns

for the distribution of data and workload;

- the interrelationships between the components — that is, their functional roles and the patterns of communication between them.

An initial simplification is achieved by classifying processes as server processes, client processes and peer processes.

— Software layers :

Applications, services

Middleware

Platform [Operating System

Computer and Network H/W

S/W and H/W service layers in distributed systems.

- **Platform :** The lowest-level h/w and s/w layers are often referred to as a platform for distributed systems and applications. These low-level layers provide services to the layers above them, which are implemented independently in each computer, bringing the system's programming interface up to a level that facilitates communication and co-ordination between processes.

- **Middleware :** It is a layer of software whose purpose is to mask heterogeneity and to provide a convenient programming model to application programmers.
  It is represented by processes or objects in a set of computers that interact with each other to implement communication and resource sharing support for distributed applications.

It is concerned with prouding useful building blocks for the construction of software components that can work with one another in a distributed system.

It raises the level of the communicat activities of application programs through the support of abstractions such as remote method invocation, communication between a group of processes, notification of events, the partitioning, placements and retrieval of shared data objects amongst cooperating computers, the replication of shared data objects ~~amongst~~ and the transmission of multimedia data in real time.

List of Middleware :

1. SUN RPC
2. Isis
3. CORBA
4. Java RMI
5. Microsoft's Distributed component Object Model (DCOM)
6. Reference Model for Open Distributed Processing (RM-ODP)

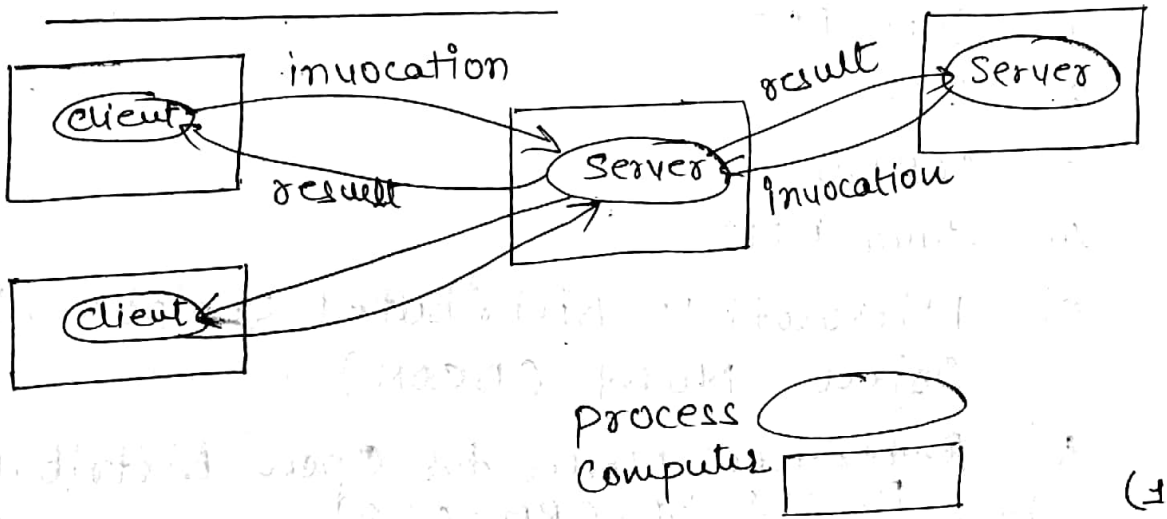
limitations of middleware :

Many distributed applications rely entirely on the services provided by the available middleware to support their needs for communication and data sharing. For example, an application that is suited to the client-server model such as a database of names and addresses can rely on middleware that provides only remote method invocation.

Much has been achieved in simplifying the programming of distributed systems through the development of middleware support, but some aspects of the dependability of systems require support at the application level.

- System Architectures :

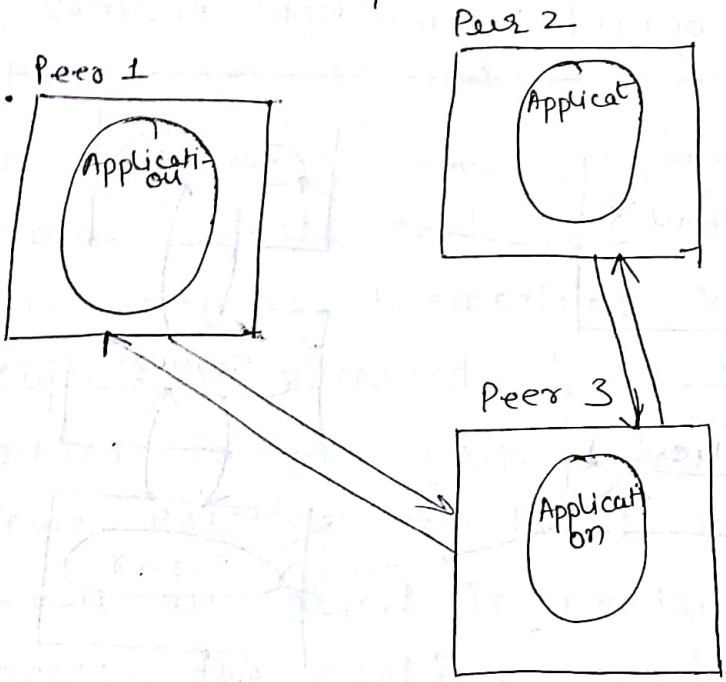
• Client - Server :



engine.

8

- **Peer to Peer :** All of the processes involved in a task or activity play similar roles, interacting cooperatively as peers without any distinction between client and server processes or the computers that they run on.



- Variations :

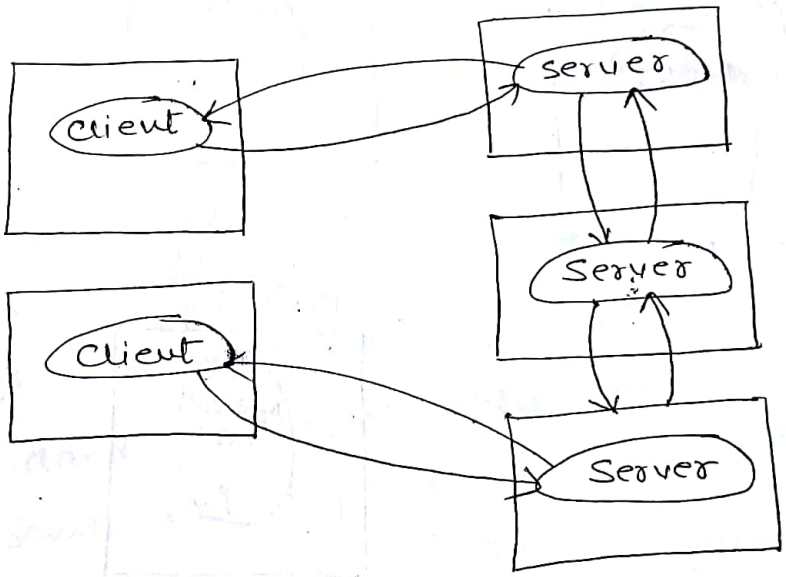
Several variations on the above models can be derived from the consideration of the following factors :

- the use of multiple servers and caches to increase performance and resilience
- the use of mobile code and mobile engine.

- Mobile Agents : A mobile agent is a

- Users need for low-cost computers with limited hardware resources that are simple to manage.
- the requirement to add and remove mobile devices in a convenient manner.

Services provided by multiple servers :



- Services may be implemented as several server processes in separate host computers interacting as necessary to provide a service to client processes.
- The servers may partition the set of objects on which the service is based and distribute them between themselves, or they may maintain

that can

replicated copies of them on several hosts.

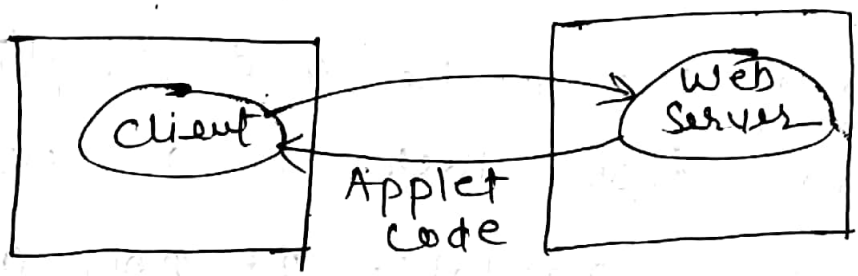
- The Web provides a common example of partioned data in which each web server manages its own set of resources. A user can employ a browser to access a resources at any one of the servers.

- Proxy servers and caches :

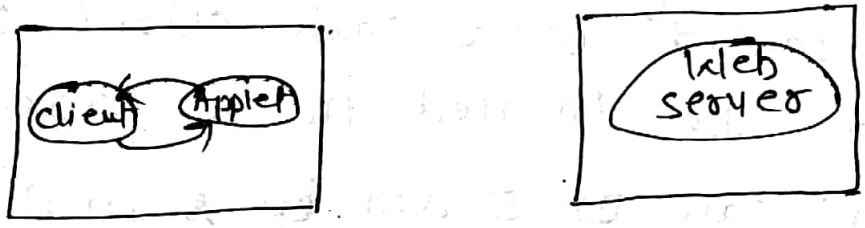
A cache is a store of recently used data objects that is closer than the objects themselves. When a new objects is received at a computer it is added to the cache store, replacing some existing objects it necessary. When an object is needed by a client process the caching service first checks the cache and supplies the object from there it an up-to-date copy is available. If not, an up-to-date copy is fetched. Caches may be collocated with each client or they may be located in a proxy server that can be shared by several clients.

- Mobile Code : Applets are a well known and widely used example of mobile code - the user running a browser selects a link to an applet whose code is stored on a web server; the code is downloaded to the browser and runs there.
An advantage of running the downloaded code locally is that it can give good interactive response since it does not suffer from delays or variability of bandwidth associated with network communication.

(a) Client request results in the downloading of applet code



(b) Client interact with the applet



- Mobile Agents : A mobile agent is a running program (including both code and data) that travels from one computer to another in a network carrying out a task on someone's behalf, such as collecting information, eventually returning with the results. A mobile agent may make many invocations to local resources at each site it visits - for example, accessing individual database entries.

Mobile agents might be used to install and maintain software on the computers within an organization or to compare the price of products from a number of vendors by visiting the site of each vendor and performing a series of database operations.

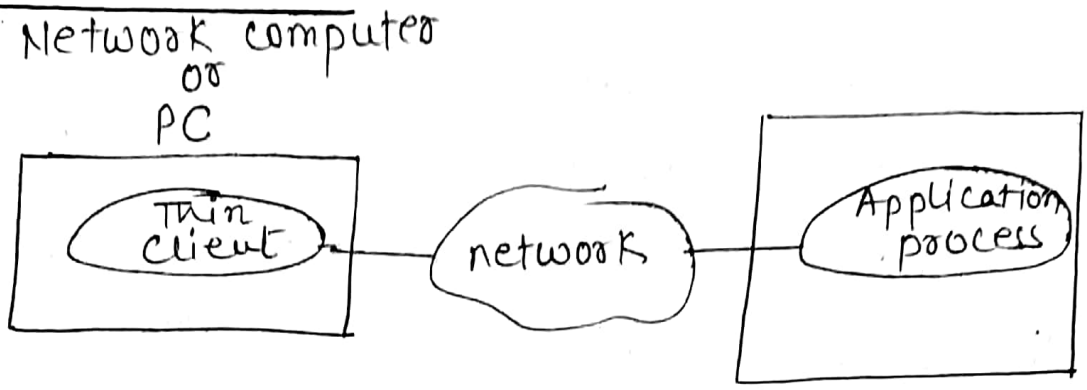
Mobile agents are a potential security threat to the resources in computers that they visit. The environment receiving a mobile agent should decide on which of the local resources it should be allowed to use, based on the identity of the users on whose

behalf the agent is acting.
Mobile agents can themselves be vulnerable. they may not be able to complete their task if they are refused access to the information they need.
The task of mobile agents can be performed by other means. For example, web crawlers.

- Network Computers : The network computer downloads its operating system and any application software needed by the user from a remote file server. Applications are run locally but the files are managed by a remote file server. Network applications such as a web browser can also be run. Since all the applications data and code is stored by a file server, the users may migrate from one network computer to another.

The processor and memory capacities of a network computer can be constrained in order to reduced its cost.

- Thin Clients :



The term thin client refers to a software layer that supports a window-based user interface on a computer that is local to the user while executing application programs on a remote computer. This architecture has the same low management and hardware costs as the network computer scheme, but instead of downloading the code of applications into the user's computer, it runs them on a computer server.

Drawback : Highly interactive graphical activities such as CAD and image processing, where the delays experienced by users are increased by the need to transfer image and vector informat between the thin client and the application process.

2. Balancing computational

- Mobile devices and spontaneous interoperation :

Mobile devices are hardware computing components that move between physical locations and thus networks, carrying software components with them. Mobility leads to spontaneous interoperation a variation on the client-server model in which associations between devices are routinely created and destroyed.

- Design requirements for distributed architectures :

- Performance issues :

1. Responsiveness : Users of interactive applications requires a fast and consistent response to interaction; but client programs often need to ~~proces~~ access shared resources.
2. Throughput : the rate at which computational work is done. It is affected by processing speeds at client and servers and by data transfer rates.

... tem systems that

3. Balancing computational loads :

Distributed system is to enable applications and services processes to proceed concurrently without competing for the same resources and to exploit the available computational resources.

- Quality of service :

The main non-functional properties of systems that affect the quality of the service experienced by clients and users are reliability, security and performance. Adaptability is another important aspect.

- Use of caching and replication :

Various issues of design of distributed system have overcome by the use of caching and replication.

- Dependability issues :

Correctness
Security
Fault tolerance.

- Fundamental Models :

A model contains only the essential ingredients that we need to consider in order to understand and reason about some aspects of a system's behaviour.

A system model has to address the following questions :

- What are the main entities in the system ?
- How do they interact ?
- What are the characteristics that effects their individual and collective behaviour ?

The purpose of a model is :

- To make explicit all the relevant ~~information~~ assumptions about the system we are modelling.
- To make generalizations concerning what is possible or impossible, given those assumptions. The generalization may take the form of general-purpose algorithms or desirable properties that are guaranteed.

The aspects of distributed systems that we wish to capture in our fundamental models are intended to help us to discuss and reason about:

- Interaction: Computation occurs within processes; the processes interact by passing messages, resulting in communication and co-ordination between processes.
- Failure: The correct operation of a distributed system is threatened whenever a fault occurs in any of the computers on which it runs or in the network that connects them. Our models defines and classify them.
- Security: Security models defines and classify the forms that such attacks may take, providing a basis for the analysis of threats to a system and for the design of systems that are able to resist them.

Interacting processes performs all of the activity in a distributed system. Each process has its own state, consisting of the set of data that it can ~~occur~~ access and update, including the variables in its program. Two significant factors affecting interacting processes in a distributed system:

- communication performance is often a limiting characteristic;
- it is impossible to maintain a single global notion of time.

- performance of communication channels:
  - The delay between the starts of the a message's transmission from one process and the begining of its receipt by another is refered as latency. The Latency includes:
    - The time taken for the first of a string of bits transmitted through

a network to reach its destination.

- The delay in accessing the network, which increases significantly when the network is heavily loaded.
- The time taken by the operating system communication services at both the sending and the receiving processes.

- The **bandwidth** of a computer network is the total amount of information that can be transmitted over it in a given time.
- Jitter is the variation in the time taken to deliver a series of messages. Jitter is relevant to multimedia data.

Computer clocks and timing events :
Each computer has its own internal clock, which can be used by local processes to obtain the value of the current time. Therefore, two processes running on different computers can associate timestamps with their

events. However, even two process read their clocks at the same time, their local clocks may supply different time values. This may happen because of clocks drift.

The term clock drift rate refers to the relative amount that a computer clock differ from one ~~another~~. a perfect reference clock.

Variants of the interaction model :

① Synchronous distributed system :

- Time to execute each step of a process has known lower and upper bounds.
- Each message terminated over a channel is received within a known bounded time.
- Each process has a local clock whose drift rate from perfect time has a known bound.

It is possible to suggest likely upper and lower bounds for process execution time, message delay and clock drift rates in a distributed system. But it is difficult to arrive at realistic values and to provide guarantees of the chosen values.

(2) Asynchronous distributed systems :

- A system in which there are no bounds on :
  - > Process execution times.
  - > Message delivery times.
  - > clock drift rate.
- Allows no assumptions about the time intervals involved in any execution.
- Exactly models the internet.
  - > Browsers are designed to allow users to do other things while they are waiting.
- More abstract and general:
  - > A distributed algorithm executing on one system is likely to also work on another one.

2. Failure Model : The failure model defines the ways in which failure may occur in order to provide an understanding of the effects of failures. There are 3 types of failures :

(i) Omission failure :

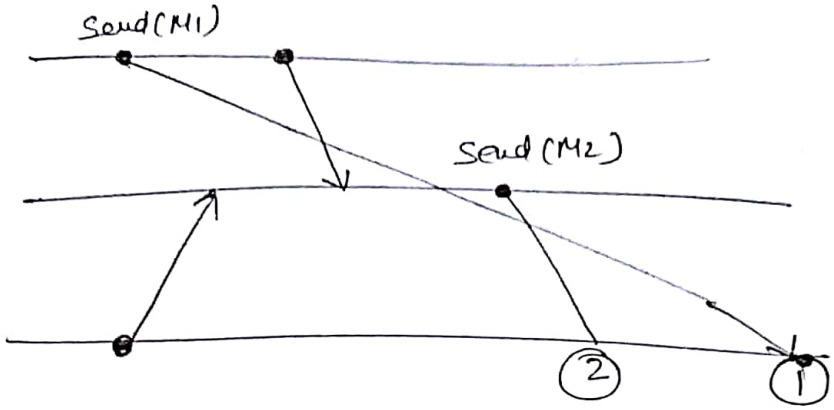
It refers to cases when a process or communication channel fails to perform action that is supposed to do. There are two types of omission failures.

They are :

(a) Process omission failures : The design of services that can survive in the presence of faults can be simplified if it can be assumed that the service on which it depends crash cleanly, i.e. the processes either function correctly or else stop. Other processes may be able to detect such a crash by the fact that the process repeatedly fails to respond to invocation messages. However, this method of crash detection relies on the use of timeouts i.e. a method in which one process allows a fixed period of time for something to occur.

A process crash is called fail-stop if other processes can detect certainly that the process has crashed.

(b) Communication omission failures : The communication channel produces an omission failure if it does not transport a message from p's outgoing message butter to q's incoming message butter. This known as 'dropping messages.' The loss of messages



between the incoming message buffer and the receiving process is called receiver omission failure and the loss of messages in between is called channel omission failure.

(ii) Arbitary Failures :

An arbitary failures of a process is one in which it ~~arbitary~~ arbitrarily omits intended processing steps or takes unintended processing steps.

(iii) Timing failures :

Timing failures are applicable in synchronous distributed systems where time limits are set ~~to~~ for all operations.

Timing failures are listed below :

Clock — Process
Performance — Process
Performance — Channel

(iv) Masking failures :

It is possible to construct reliable servics from components that exhibit failures. For example, multiple servers that holds replicas of data can

continue to provide a service when one of them crashes. A knowledge of the failure characteristic of a component can enable a new service to be designed to mask the failure of the components on which it depends. A service masks a failure, either by hiding it altogether or by converting it into a more acceptable type of failure.

(V) Reliability of one to one Communication

The term reliable communication is defined in terms of validity and integrity as follows :

Validity : Any message in the outgoing message buffer is eventually delivered to the incoming message buffer.

Integrity : The message received is identical to one sent, and no messages are delivered twice.

3. Security Model :

The security of a distributed system can be achieved by securing the processes and the channels used for their interactions and by protecting the objects

possible threats :
that they encapsulate against unauthorized access.

Protecting objects : Protection is described in terms of objects. Objects are intended to be used in different ways by different users. For example, some objects may hold a user's private data and other objects may hold shared data such as web pages. To support this, access rights specify who is allowed to perform the operations (read / write) of an object.

Securing Processes and their Interaction : Processes interact by sending messages. To secure the messages passed over the network, cryptographic algorithms (encryption) is applied to the message, to provide confidentiality, authentication and integrity. The threats from a potential enemy are as follows :

- Threats to process : The lack of reliable knowledge of the source of a message is a threat to the correct functioning of both servers and clients. e.g. spoofing.
- Threats to communication channels :

An enemy can copy, alter or inject

messages as they travel across the network and its intervening gateways. Such attacks present a threat to the privacy and integrity of information as it travels over the network and to the integrity of the system.

All these threats can be defeated by the use of secure channels, which is described below :

Secure Channels

- It ensures the privacy and integrity of the data transmitted across it. Encryption and authentication are used to build secure channels as a service layer on top of existing communication services.
- A secure channel is a communication channel connecting a pair of processes with the following properties :
  - > Each of the processes know reliably the identity of other processes.
  - > It ensure the privacy and integrity of data transmitted across it.
  - > Each message includes a physical or logical time stamp to prevent messages from being replayed.

Other possible threats :

- Denial of service : This is a form of attack in which the enemy interfaces with the activities of authorized users by making excessive and pointless invocations on services or message transmissions in a network, resulting in overloading of physical resources.
  Such attacks are usually made with the intention of delaying or preventing actions by other users.
- Mobile code : Any process that receives and executes program code from elsewhere, such as the e-mail attachements. Such code may easily play a Trojan horse role, purporting to fulfill an innocent purpose but in fact including code that accesses or modifies resources that are legitimately available to the host process but not to the originator of the code.

Drawbacks of Security Techniques :

The use of security techniques such as encryption and access control incurs substantial processing overhead

Theoretical Foundation for

and management costs. (1·40)

Theoretical Foundation for Distributed System

A distributed system refer to a system that consists of several computers that do not share a memory or a clock and communicate with each other by exchanging messages over a communication network.

Motivations

1. Improved price/performance.
2. Resource sharing.
3. Enhanced performance.
4. Improved reliability and availability.
5. Modular expandability.

Issues in ~~Dsi~~ Distributed operating Systems

1. Global knowledge
2. Naming
3. Scalability
4. Compatibility : 3 level → binary, execution, protocol.
5. Process synchronization
6. Resource Management
7. Security
8. Structuring
   (i) Monolithic Kernel

(ii) Collective kernel structure

(iii) Object-oriented operating system.

Inherent limitations of a Distributed System

- Common time does not exist in a distributed system; different processes may have different notions of time.
- Since processes in a distributed system do not share common memory, it is not possible for an individual process to obtain an up-to-date state of the entire system.

① Absence of a Global Clock:

There exists no systemwide common clock (global clock).

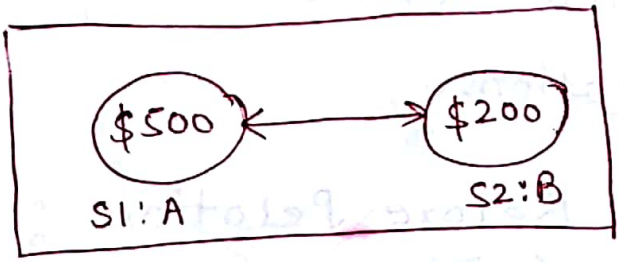
If we are having synchronized clock for all processes in the system. In this case, two different processes can observe a global clock value at different instants due to unpredictable message transmission delays. Therefore, two different processes may falsely perceive two different instants in physical time to be a single instant in physical time.

On the other hand, if we provide each computer in the system with a physical

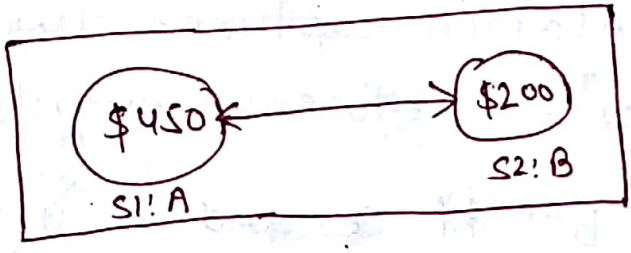
clock and try to synchronize them, these physical clock can drift from the physical time and the drift rate may vary from clock to clock due to technological limitations. Therefore, this approach can also have two different processes running at different computers that perceive two different instant in physical time as a single instant.

(2) Absence of Shared Memory :
Since the computers 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.

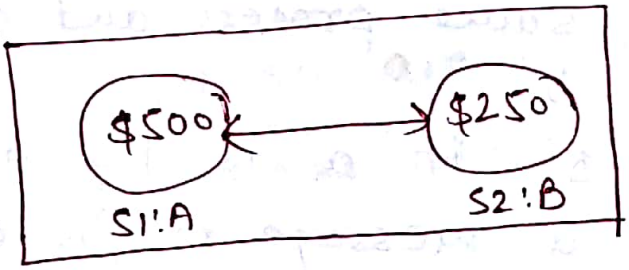
(a)



(b)



(c)



LAMPORT'S LOGICAL CLOCKS :

- The execution of processes is characterized by a sequence of events.

Definitions

Due to the absence of perfectly synchronized clocks and global time in distributed systems, the order in which two events occur at two different computers cannot be determined based on the local time at which they occur. However, under certain conditions, it is possible to ascertain the order in which two event occur based solely on the behaviour exhibited by the underlying computations.

Happened Before Relation ($\rightarrow$) : The happened before relation captures the casual dependencies between events.

- $a \rightarrow b$, if a and b are events in the same process and a occured before b.
- $a \rightarrow b$, if a is the event of sending a message m in a process and b is the event of receipt of the same message m by another

process.

- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$ i.e. "$\rightarrow$" relation is transitive.

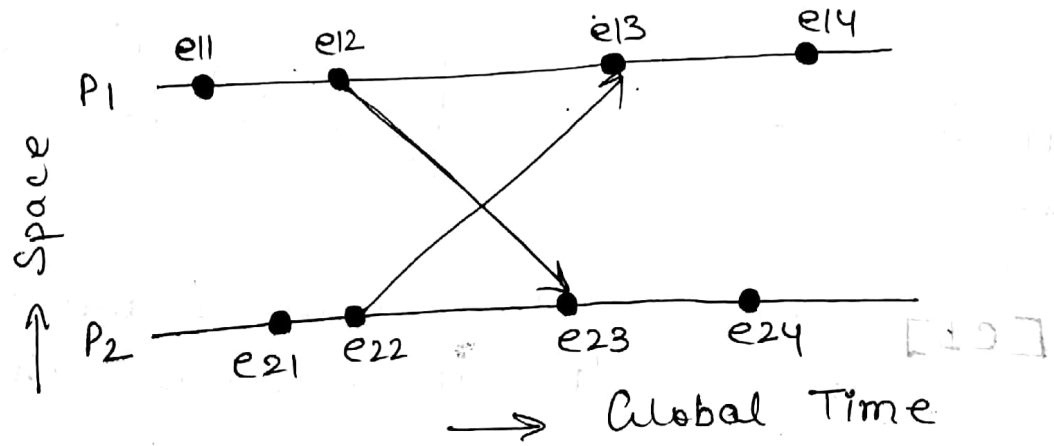
Casually Related Events :

Event a ~~usually~~ causally affects event b if $a \rightarrow b$.

Concurrent Events :

Two distinct events a and b are said to be concurrent (denoted by $a \| b$) if $a \not\rightarrow b$ and $b \not\rightarrow a$. In other words, concurrent events do not casually affect each other.

Example :



e11, e12, e13 and e14 are events in process P1 and e21, e22, e23 and e24 are events in process P2.

$e12 \rightarrow e23$, $e23 \rightarrow e24$ i.e. $e12 \rightarrow e24$

$C_i(a) < C_j(b)$

$e_{22} \rightarrow e_{13}$, $e_{13} \rightarrow e_{14}$ i.e. $e_{22} \rightarrow e_{14}$

Logical Clocks : The clock C_i can be thought of as a function that assigns a number $C_i(a)$ to any event a, called the time-stamp of event a, at P_i.

- The numbers assigned by the system of clocks have no relation to physical time, and hence the name logical clocks.
- These clocks can be implemented by counters.

Conditions satisfied by the system of clocks For any events a and b :

if $a \rightarrow b$, then $C(a) < C(b)$

The happened before relation '$\rightarrow$' can now be realized by using the logical clocks if the following two conditions are met :

[C1] For any two events a and b in a process P_i, if a occurs before b, then

$$C_i(a) < C_i(b)$$

[C2] If a is the event of sending a message m in process P_i and b is the event of receiving the same message m at process P_j, then

$C_i(a) < C_j(b)$

The following implementation rules (IR) for the clocks guarantee that the clocks satisfy the correctness conditions C1 and C2:

[IR1] Clock C_i is incremented between any two successive events in process P_i:

$$C_i := C_i + d \quad (d > 0)$$

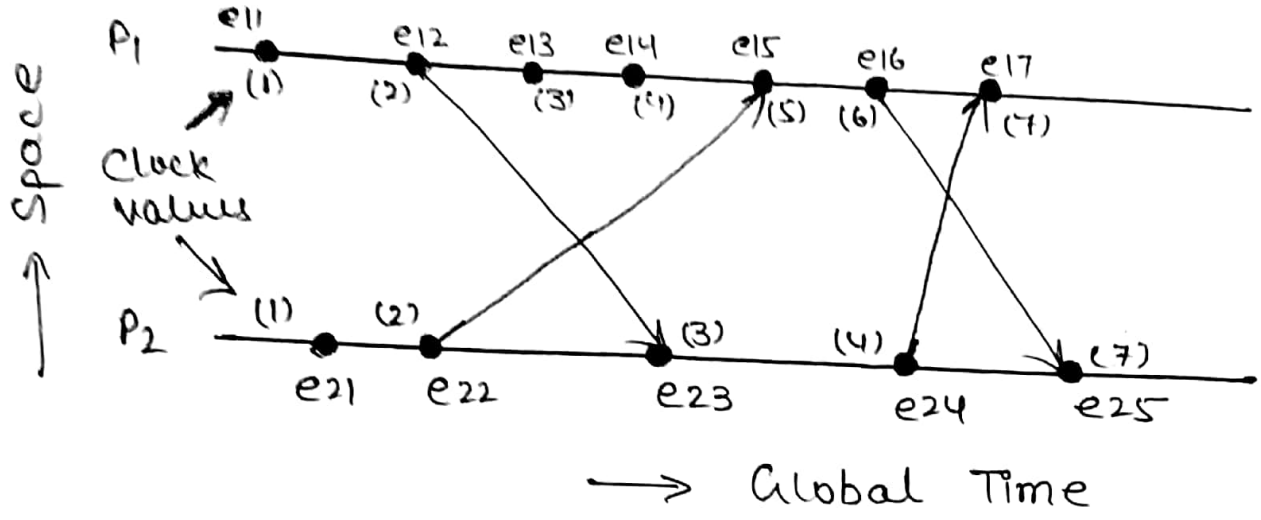
If a and b are two successive events in P_i and $a \rightarrow b$, then $C_i(b) = C_i(a) + d$.

[IR2] If event a is the sending of message m by process P_i, then message m is assigned a timestamp $t_m = C_i(a)$. On receiving the same message m by process P_j, C_j is set to a value greater than or equal to its present value and greater than t_m.

$$C_j := \max(C_j, t_m + d) \quad (d > 0)$$

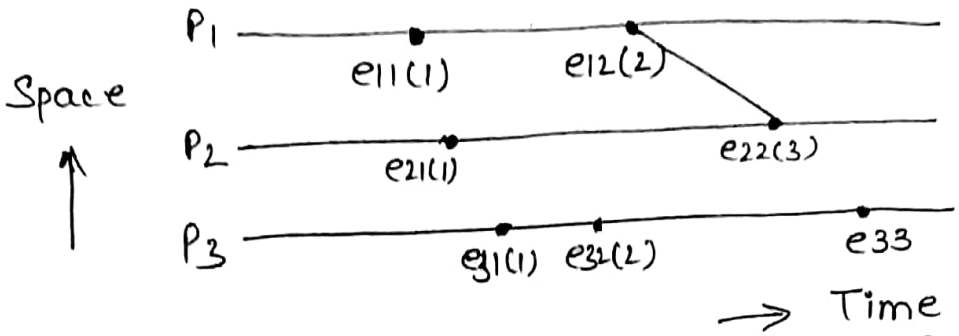
(1147)

(1.51)



Both the clock values CP_1 and CP_2 are assumed to be zero initially and d is assumed to be 1. e_{11} is an internal event in process P_1 which causes CP_1 to be incremented to 1 due to IR1. Similarly, e_{21} and e_{22} are two events in P_2 resulting in $CP_2 = 2$ due to IR1. The message is assigned a timestamp = 6. The event e_{25}, corresponding to the receive event of the above message, increments the clock CP_2 to 7 (max(4+1, 6+1)) due to rules IR1 and IR2. Similarly, e_{24} is a send event in P_2. The message is assigned a timestamp = 4. The event e_{17} corresponding to the receive event of the above message increments the clock CP_1 to 7 (max(6+1, 4+1)) due to rules IR1 and IR2.

limitation of Lamport's clocks



It shows $C(e_{11}) < C(e_{22})$ and $C(e_{11}) < C(e_{32})$

Lamport's system of clocks, we can guarantee that if $c(a) < c(b)$ then $b \not\to a$, but we cannot say whether events a and b are casually related or not, by just looking at the time-stamps of the events.

Vector Clocks :

The system of vector clocks was independently proposed by Fidge and Mattern.

Let 'n' be the number of processes. Each process pi is equipped with a clock ci, which is an integer vector of length n.

The clock ci can be thought of as a function that assigns a vector ci(a) to any event a. ci(a) is referred to as the time stamp of event a at pi.

(1.49)

The implementation rules for the vector clocks are as follows :

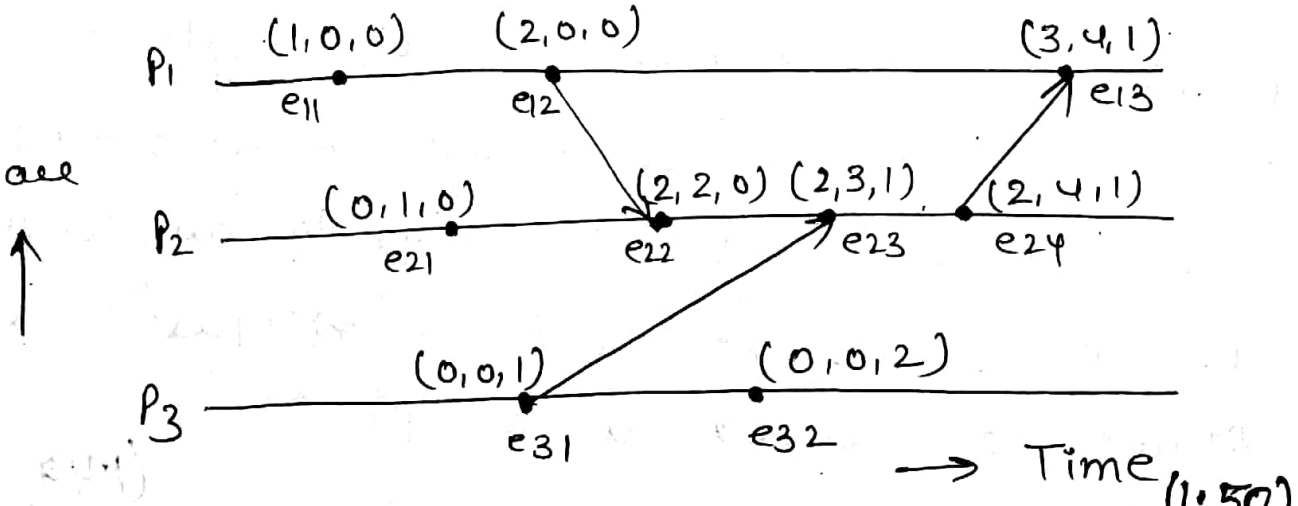
1. Clock c_i is incremented between any two successive events in process p_i as

$$c_i[i] := c_i[i] + d \quad (d > 0)$$

2. If event 'a' is sending of message 'm' by process p_i then message 'm' is vector timestamp $t_m = c_i(a)$; on receiving the same message 'm' by process p_j, c_j is updated as follows :

$$\forall k, \quad c_j[k] := \max(c_j[k], t_m[k])$$

In rule (1), we treat message send and message receive by a process as events, In rule (2), a message is assigned a timestamp after the sender process has incremented its clock due to (1).



(1.50)

Casually related events : Events 'a' and 'b' are casually related if $t^a < t^b$ or $t^b < t^a$. Otherwise these events are concurrent.

Thus, the system of vector clocks allows us to order events and decide whether two events are casually related or not by simply looking at the timestamps of the event.

- Casual Ordering of Messages :

It deals with the notion of maintaing the same casual relationship that holds among "message send" events with the corresponding "message receive" events. Techniques for the casual ordering of messages are useful in developing distributed algorithms and may simplify the algorithm themselves. For example, for applications such as replicated database system, it is important that every process in charge of updating a replica receives

the updates in the same order to maintain the consistency of the database In the absence of casual ordering of messages, each and every update must be checked to ensure that it does not violat the consistency constraints.

(1.52)

(1) Birman - Schiper - Stephenson Protocol :

1. Before broadcasting a message m, a process pi increments the vector time $VT_{pi}[i]$ and timestamps m. Note that $(VT_{pi}[i]-1)$ indicates how many messages from pi precede m.

2. A process $pj \neq pi$, upon receiving message 'm' timestamped VT_m from pi, delays its delivery untill both the following conditions are satisfied :

(a) $VT_{pj}[i] = VT_m[i] - 1$

(b) $VT_{pj}[K] \geq VT_m[K] \quad \forall K \in \{1, 2, \ldots n\} - \{i\}$

where n is the total number of processes

Delayed messages are queued at each process in a queue that is sorted by vector time of the messages. Concurrent messages are ordered by time of their receipt.

3. When a message is delivered at a process pj, VT_{pj} is updated according to the vector clock rule (i.e. implementation Rule 2)

Step 2 is the key to the protocol. Step 2 (a) ensure that pj has received all those messages received by pi before sending m.

Step 2(b) ensures that pj has received all those messages received by pi before sending 'm'.

2. Schiper-EGGLI-Sandoz Protocol :

Each process p maintains a vector denoted by V-p of size (N-1), where N is the number of processes in the system. An element of V-p is an ordered pair (p', t) where p' is the ID of the destination process of a message and t is the vector timestamp. The processes in the system are assumed to use vector clocks. The following notations are used in describing the protocol.

- tm = logical time at the sending of message M.
- tpi = present/ current logical time at process pi.

The Protocol

Sending of a message M from Process p1 to process p2.

- Send messge M (timestamped tm) along with V-p1 to process p2.
- Insert pair (p2, tm) into V-p1. If V-p1 contains a pair of (p2, t), it simply gets over written by the new

pair (p_2, tm). Any future message carrying the pair (p_2, tm) cannot be delivered to p_2 untill $tm < tp_2$.

Arrival of a message M at process p_2:

If v_M (the vector accompanying message M) does not contain any pair (p_2, t)
then the message can be delivered else.
if $t \neq tp_2$ then
the message cannot be delivered else.
the message can be delivered.

If message M can be delivered at process p_2 then the following three actions are taken:

1. Merge v_M accompanying M with $v\text{-}p_2$ in the following manner:
   - If $(\exists p, t) \in v\_M$ such that $p \neq p_2)$ and $(\forall (p', t) \in v\text{-}p_2, p' \neq p)$, then insert (p, t) into $v\text{-}p_2$.

This means if there is no entry for process p in $v\text{-}p_2$ and v_M contains an entry for process p, insert that entry into $v\text{-}p_2$.

- $\forall p_3, p \neq p_2$, if $((p, t) \in V\_M) \wedge ((p, t') \in V\_p_2)$,

then the algorithm takes the following actions :

$(p, t) \in V\_p_2$ can be substituted by the pair $(p, tsup)$ where $tsup$ is such that $\forall i$, $tsup[i] = max(t[i], t'[i])$.

Due to the above two actions, the algorithm satisfies the following two conditions :

(a) No messages can be delivered to p as long as $t' < tp$ is not true.

(b) No message can be delivered to p as long as $t < tp$ is not true.

2. Update site p_2's logical clock.

3. Check for the buffered messages that can now be delivered since local clock has been updated.

A pair (p, t) can be deleted from the vector maintained at a site after ensuring that the pair (p, t) has become obsolete.

Global State :

A recorded global state may be inconsistent if $n < n'$ where n is the number of messages sent by A along the channel before A's state was recorded and n' is the number of messages sent by A along the channel before the channel's state was recorded.

Global state may be inconsistent if $n > n'$.

Hence, a consistent global state requires

$$\boxed{n = n'}$$

or

$$\boxed{m = m'}$$

where m' = number of messages received along the channel before account B's state was recorded and m = number of messages received along the channel by B before the channel's state was recorded.

Since there is no system in which the number of messages sent along the channel can be less than the number of messages received along the channel. i.e.

$$\boxed{n' \geq m}$$

So, $$\boxed{n \geq m}$$

- Chandy - Lamport's Global State Recording

Consistent Global State :

For every received message a corresponding send event is recorded in the global state.

Inconsistent global state :

There is at least one message whose receive event is recorded but its send event is not recorded in the global state.

Transitless global state :

All communication channels are empty in a transitless global state.

Strongly consistent global state :

A global state is strongly consistent if it is consistent and transitless. In this, not only the send events of all the recorded received events are recorded, but the receive events of all the recorded send events are also recorded. Thus, a strongly consistent state corresponds to a consistent global state in which all channels are empty.

- Chandy - Lamport's Global State Recording Algorithm :

The algorithm uses a marker (a special message) to initiate the algorithm and the marker has no effect on the underlying computation. The communication channels are assumed to be FIFO. The recorded global state is also referred to as a snapshot of the system state.

- Marker Sending Rule for a Process P.
  - P records its state.
  - For each outgoing channel C from P on which a marker has not been already sent, P sends a marker along C before P sends further messages along C.
- Marker Receiving Rule for a Process Q:

  On receipt of a marker along a channel C.
  If Q has not recorded its state then
  begin
  Record the state of C as an empty sequence.

Follow the "Marker sending Rule."
end.
else.
Record the state of C as the sequence of messages received.
along C after Q's state was recorded and before Q received.
the marker along C.

Termination Detection :

- A computation is said to have terminated if and if all the processes are idle and there are no messages in transit.
- The messages sent by the termination detection algorithm are referred to as control messges.
- One of the co-operating processes monitors the computation and is called the controlling agent.

Huang's Termination Detection Algorithm

Rule 1 : The controlling agent or an active process having weight W may send a computation message to a process P by doing :
Drive W_1 and W_2 such that

$w_1 + w_2 = w, \quad w_1 > 0, \quad w_2 > 0$

$w := w_1$;

send $B(w_2)$ to P;

Rule 2 : On receipt of $B(\Delta w)$, a process P having weight w does.

$w := w + \Delta w$;

If P is idle, P become active.

Rule 3 : An active process having weight w may become idle at any time by doing :

send $C(w)$ to the controlling agent ;

$w := 0$;

(The process become idle) ;

Rule 4 : On receiving $C(\Delta w)$, the controlling agent having weight w takes the following actions

$w := w + \Delta w$;

If $w = 1$, conclude that the computations has terminated.

* —— *