Distribuidos 1. Introduction

Distributed system: Interconected processing units seen as a single computer by users. Multiprocessors with shared memory are included

Advantages

- High computer at low cost
- Data and equipments shared
- Adaptions for banks, trading...
- Reliabity, one component fails not a crash, just a degrade of performance
- Scalability, adding new components

Disadvantages

- Complex (concurrency)
- No global clock, 2 separated computers cannot have synchronized clocks
- Need of middleware (heterogeneous)
- + Elements = + Passible failures
- Vulnerability, need of security

A → C → B

Transparency and types

Management of distributed resources must be hidden to users

- **Access**: Local and remote resources should be accessed in the same way
- **Location**: Users don't need to know resources location
- **Migration**: Resources can move without user intervention
- **Replication**: Can be several copies, but user only sees one
- **Concurrency**: User shouldnt know it
- **Parallelism**: Automatic, better exploit of architecture
- **Failures**: Recovery should be automatic, secret for user
- **Security**: Secure access, simple and transparent way

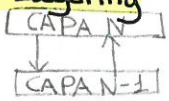
Architectural models

Connectors are used to communication and synchronization between components: JMS, RPC...

Component is specified by a contract that has: interface (implementation) and dependencies

Techniques and models:

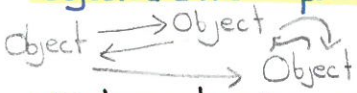
- **Layering**



Just communication from i to i-1, if not, problems

- **Object based**

Objects are the components

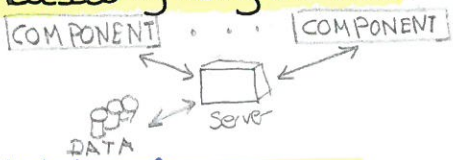


Ex: client/server ⇒

Management in server-side, integration, distribution...
Easier interrelation, + security, + concurrency
Intercept interfaces, do things right

- **Data centered model**
  - Direct access by components

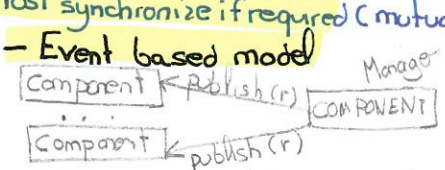
  - Accessed by using a server

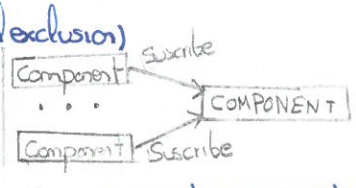


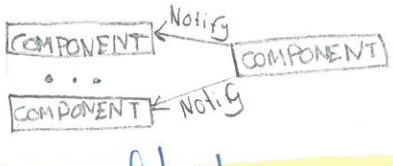
Distributed file servers (DFS) → MicroSoft

Must synchronize if required (mutual exclusion)

- **Event based model**



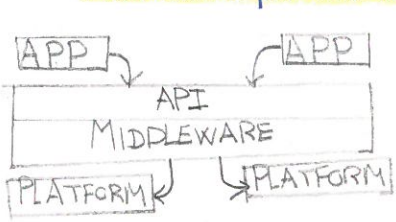


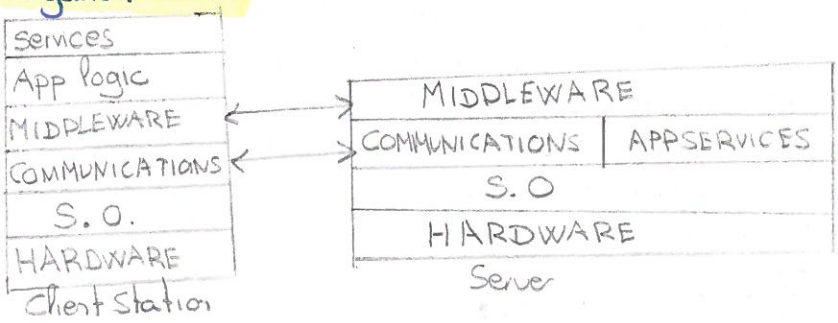


  - **Content-based notification**: Suscribers indicate a query related to resource contents (state), notified when this predicate become true
  - **Subject-based notification**: Clients joins group of interest, all notified of the event related to that group

Middleware

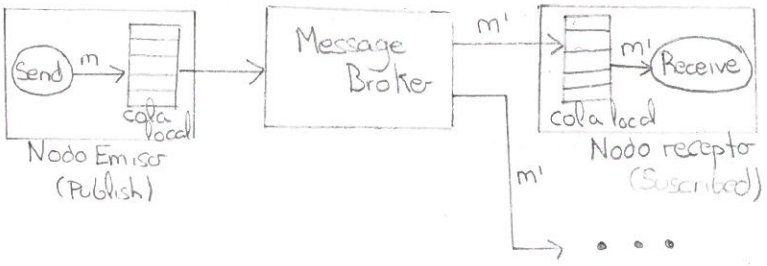
Design of distributed applications is complex, especially in heterogeneous environments
↳ Solution: Provide a standarized way of interoperability
Uniform access to the system resources, independtly of the supporting platform. Middleware ≠ OperativeSys
Middleware provides transparency in the management resource





Distributed system is dynamic, code, new devices, failures... So addaption cannot stop the system.
An addaptative middleware would fix all those problems, + replaceable components. Middware has mechanisms to dynamically load/unload components
Application server would help with those tasks, handling scalability, ladd-balancing and recovery
Ex: glassfish, Weblogic...
RPC (Remote Procedure Call): Protocols to invoke remote code
MOM (Message-oriented middleware): To send and receive messages. JMS Java
ORB (Object Requested Broker): Provide interoperability between a collection of distributed objects (Code)
Database Middleware: Support for database interoperability. Transactions service
Virtual File Systems: Abstract layer to provide uniform access to diverse file system, local or remote.
Message Broker: Connector that transforms message format for interoperability. Automatic conversor
They use syntactic rules that establish how the message contents must be transformed
Where are they placed? Either in sender, receiver or different node
Can perform more complex operations, such as implement the topic-based publish/suscribe/notify

Message-Oriented Middleware (MOM)



UNIT 1

Distributed System

Advantages
- High comput. Low Cost
- Data and equip shared
- Reliability (Solo falla 1 parte)
- Scalability

Disadvantages
- Complex (concurrency)
- No global clock
- Need of middleware
- + Elements = + Possible failures
- Vulnerability

Transparency Types

- **Access**: Local = Remote
- **Location**: No need to know where
- **Migration**: Resource may no user known
- **Failures**: Automatic and secret
- **Replication**: Several copies, user only sees one
- **Concurrency**
- **Security**: Simple and transperent
- **Parallelism**: Automatic, ↑ Exploit

Architectural Models

- **Layering**: Communication frami to i-1 layers
- **Object Based**: Objects are the components. Ex: Client/server
- **Data Centered**
  - Direct Access by components
  - By Using a Server
- **Event Based** (*)
  - Conted-based Notify
  - Subject-based Notify

(*)
- Publish resource
- Suscribe to possible event
- Notify

Container

1. Integrates components of an App
2. Facilitates interrelation automating
3. Intercept interfaces of components, to run correctly

Middleware

Standarized way to interoperate. Transparence in distribution. Unified access to resources. Middleware ≠ S.O.

- **RPC** Remote Procedure Call: invoke code
- **MOM** Message-Oriented Middleware: Recive and send messages. JMS Java
- **ORB** Object Requested Broker: Interoperability between distributeds (code)
- **Database Middleware**: Database interopeability
- **Virtual File System**: Abstract layer to provide uniform access to local or remote
- **Message Broker**: Transforms message format for interoperabili. Automatic. Placed in receiver/sender/Node. Can implement complex op. Topic-based PISI (*)

Unit 2

Send/receive ⇒ Client (dest, Mg) / Server (msg)

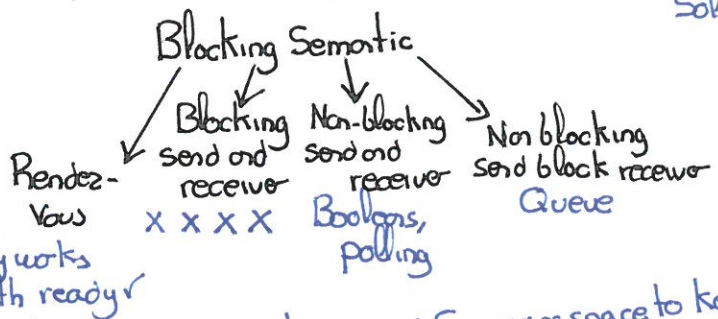
Addressing ⇒ Proces and device must be known. PID!! ⇒ Changes... Then? Ports!

Ports... How to know who? List of services supported by X port ≈⇒ Superservers (Many Ports!)

Problems with process identification!!

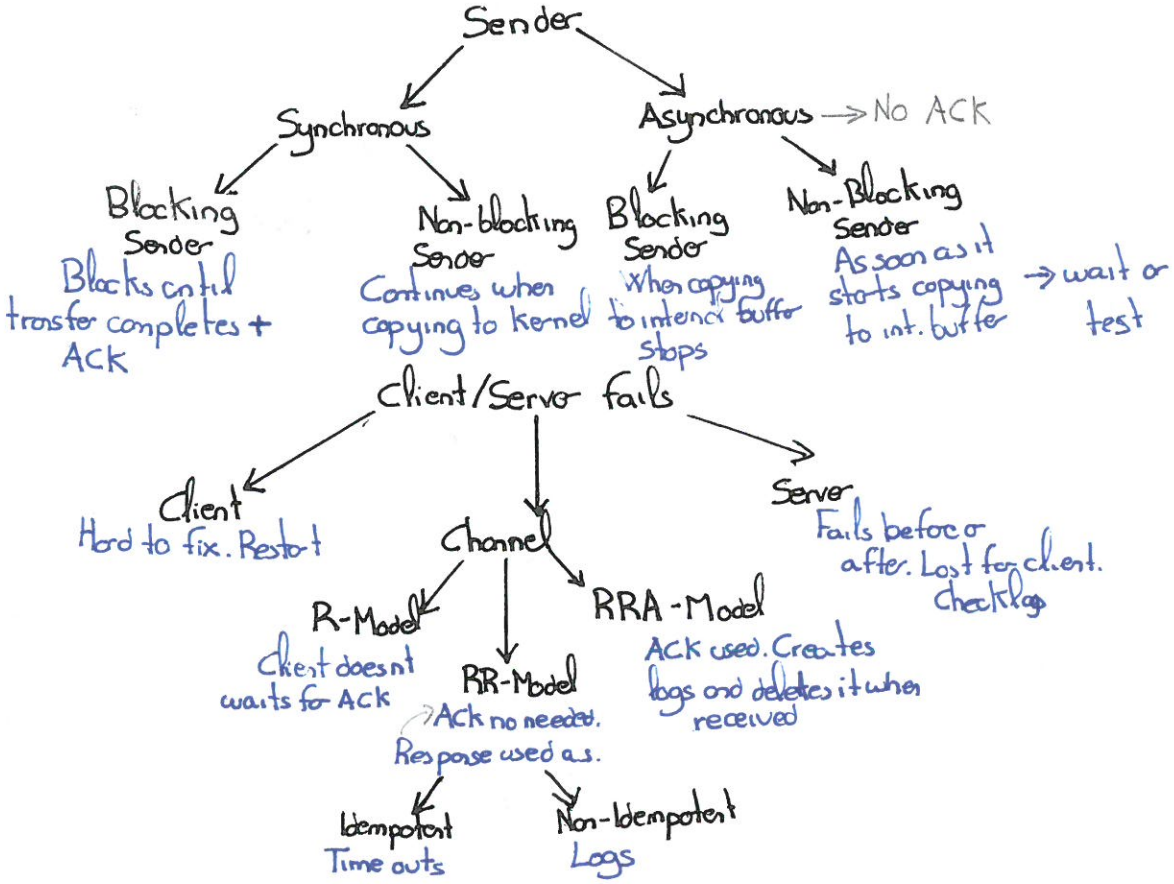
↳ Need unique mechanism. IP, port, data creation guide. Problem? Location dependant...
Solution? Initial node with ubi

UBI



Send/Receive with buffering: Msg data copied from user space to kernel (send), and viceversa (receive

Advanced Blocking Semantic (Congestion)



Socket: Intermediate element in distributed systems, in order to stablish a standarized process in some kinds of communication.

UDP in Java: Non-reliable, small size communications. DatagramSocket(int port); DatagramPacket(buff, lenght, dire port)

TCP in Java: Reliable, big size, congestion control. SeverSocket(int port); Socket(int host, int port)

Marshalling: Is a process to transform information into canonical form, so it can be easier to understand and transform. Receiver unmarshalls. Java implements it thanks to Serialization

Reflection in Java: Code modification in real time. Can invoke methods or objects that are not already able to be used. Glassfish takes advantage of it

XML: Used for transmission of documents via internet. docx

JSON: Lightweight data transfo. Legible for human

Implementing a server in Java RMI

UnicastRemoteObject → Class to implement Java RMI Servers. TCP Communication. Steps:

1. Código de server + interface
2. Código cliente, usa la interface
3. Compilar todo
4. Run rmiregistry command (binder)
5. Run Server, then client

Implementación Servidor RMI

- Implemento interfaz del servicio (Hello)
- Implemented by UnicastRemoteObject
- Must send exception RemoteException
- Install securityManager

Implementación Cliente RMI

- Sin callbacks no tiene pq ser objeto remoto
- Instala tmb SecurityManager
- Find reference to server
- Invoke server

Java RMI Activable class VS UnicastRemoteObject

- **Activable class:** Pueden estar en modo pasivo. Reactivados por un monitor de activación
- **UnicastRemoteObject:** Crear y exportar servidores easily. Cuando exportados, el cliente puede enviar peticiones

Facilita comunicación TCP. Transitorio, cuando acaba el proceso creador, también su ejecución, pero no muere y atiende peticiones entrantes

Entrepise java beans (EJB): Java component to develop server side. Deployed with Glashfish help

Advantage

- Easy application deployment
- Simple access to external resources
- Easy implementation of server-side with lightweight clients.

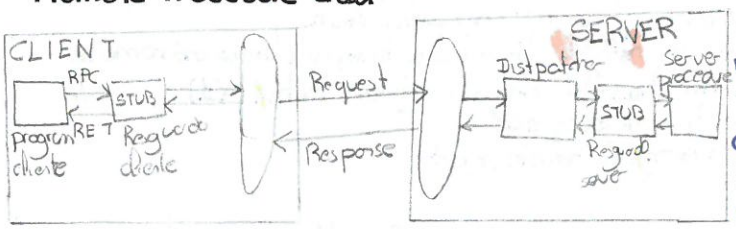
Disadvantages

- Specific technologies, less flexible
- Servers are heavyweight, stations highly loaded

- Session Beans: Implement particular task. Assigned from a pool to attend client's request
  - Stateless: Not maintained between two clients interaction
  - Stateful: They maintain a conversational state
  - Singleton: Stateless, but with a single instance shared by all clients
- Message-Driven Beans: Receive messages in JMS (Queue). No state maintained. Use onMessage

Method to receive msg

Remote Procedure Call



Client's stub: One per each procedure. Marshals the request msg and sends it through the communic. module. Unmarshals respons

Dispatcher: When a msg arrives. Takes server ID, invokes corresponding stub, passing the msg as argument

Server stub One per each procedure. Unmarshals and invokes t procedure, then marshals it and sends response through distpatch

Service interfaces are defined using specific text files. Interface compiler must be used to produce the Structural elements (stubs, communication modules)

No objects, cannot use them

Semantic models: Maybe, at least one, at most one → SUN RPC

Distribuidos 3. Distributed objects and remote invocation

Remote invocation (two models)
- RPC (Remote Procedure Call): Extension of procedure call
- RMI (Remote Method Invocation): Extension of method invocation

→ RPC, C based (tech based in Java and objects). Through server
→ RMI, not C based. More powerful

IDLs (Interface Definition Languages): Provide language standardizations for the source interfaces: methods/procedures
→ No variables declared (No shared memory)
→ In a node a variable is accessed via get/set
→ Parameter passing semantics: input, output, input-input

Examples: Corba IDL → similar to C. Support of languages // Java RMI → Interfaces of remote objects in Java

Distributed Object Communication ≃ RMI

Advantages:
- Reuse of code through inheritance, abstract classes, polymorphism
- Encapsulation and information hiding
- Object interaction via methods
- Objects can be used as parameters, and can be returned as result of a method
- Middleware allows use of different programming languages to implement objects

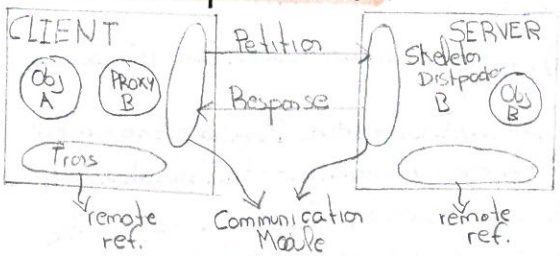
Remote invocation semantics:

	Resend Request	Seq Number
Maybe	X	X
At least once	✓	X
At most once	✓	✓

- Maybe: Happens once or never.
  If fails, stops. Client doesn't know if the petition was attend
- At least once: Resends until exit or runs out of tries (controlled by time outs)
  Same one not idempotent, so cannot be repeated
- At most once: If fails, resends, but has sequence numbers and detects duplicated
  Logs response to not be repeated processes

(Java RMI / Corba IDL)

RMI Implementation



~ Java RMI

- Communication module: R, RR or RRA semantics.
  Client: Resend manager if no ACK reply
  Server: Invokes distpatcher and sends ACKs
- Remote reference module: Translates between local and remote object references. This has the Remote object table: (1)
  (1) - 1 entry per each remote object
  - 1 entry per each proxy for remote object

- RMI Software (Proxy, Distpatcher, Skeleton)

  Proxy: Client sees remote object as locals. (Local → Remote). It performs the marshalling, homogenea los datos. Objeto remoto tiene un proxy local

  Remote objects (servers): Remote interface, must be serializable. With Remote Exception

```
import java.rmi.*;
public interface Eleccion extends Remote{
  void votar(String candidato, int dniVotante) throws RemoteException;{
```

~ We can use local objects in remote invocations (must be serializable)

```
eleccion.votar("Javi", 123);
```

RMI Registry (Binder) Servicio de registro de objetos remotos, se actua con rmi registry. Metodos de la clase Naming → Bind, registra los nombres de los objetos. Rebind, asocia un nombre a uno ya asocia Lookup, devuelve el proxy correspondiente a ese objeto remoto

RMI Implementation Servers that invoke occasionally. Can pass to passive state. Activator in charge of react

Callbacks: Los clientes se pueden comportar como servers, ofreciendo métodos al server. El server pide como si fuese cliente. La respuesta viene en el clients reply. El binder busca el lookup method. Control de concurrencia necesario

Process communication

Socket: Intermediate element of communication provided by the local O.S. between processes to set an homogeneous set of routines

UDP in JAVA → Datagrams, small messages. Non-reliable transmission → No fiable

Class `DatagramSocket(int port)` () → port libre

`DatagramPacket(byte buff[], int lenght, Int Address, port`

TCP in JAVA → Connection-oriented. Reliable transmission. No size limit and congestion control

Class `ServerSocket(int port)`

`Socket(String host, int port)`

Marshalling

Each computer represents the information in different ways, so marshalling transmits the info in a canonical form, understood by everybody. The sender transforms and sends it to the receiver, who unmarshalles to its local format

Java supports it thanks to serializable interface → transforms it in a sequence of bytes

No methods, int, boolean... are serializable, static variables NOX or locals

Classes `ObjectOutputStream` `ObjectInputStream`

Reflection in Java

Code that changes its behaviar in real-time, we can determine class content in real time. We can instantiate an object from a class unknown at compile time, calling their methods (Used by glassfish)

Package java.long.reflect needed.

`Object newInstance();` Methods → `String getName()`, `Object invoke(Object obj, Object[] args)`

XML: Lenguage to establish the structure of documents for their transmision on the Internet. .docx, has CSS, used for validation, extensible, text based

JSON: Lightweight for data exchange, easier for human reading, based on javaScript but independt With objects and arrays

Groups

Collection of application related processes with the goal of an efficient multicast communication Can be:

- **Closed**: Only group members send messages
- **Open**: Other processes can send messages

- **Static**: Members stablished when group created
- **Dynamic**: Members can come in and go out

`Msend(Group, msg)`: Msg to all members of grap

`Receive(msg)`: Receive a msg

Used for:

- Fault tolerance based on replicated services: Client sends request, everybody performs service
- Discovering services in spontaneous networking: Sends of multicast by server or user
- Data replication: Multicast messages for updates
- Event notification: In P/S/N(r) when an event occurs

Group identification:

- At APP level: Authority can assign group IDs, managing registrations and deregist
- At Physical level: Class D IPs for multicast

Ordered multicast

- No order: Messages can arrive in different order to grap members
- FIFO: If P sends m_1 and m_2, all receive m_1 and m_2
- Causal: If P_1 sends m_1 and P_2 sends m_2, all receives P_1 and then P_2
- Total ordering: All members receive the messages in exactly same order

{ High cost to implement

Dinamic groups (problems): Messages must be sent to all members, could also be messages in transit → Sol: Ordering

Atomicity is not guaranteed: all or nothing.

UDP Multicast communication in Java

`MulticastSocket` (extends `DatagramSocket`)

Constructor `MulticastSocket(int port)`

`void joingroup/leavegroup(InetAddress maastadrrs)` IP from D

`void setTimeToLive(int ttl)` Rutas por los que pasará

Reliable multicast

Msgs with IDs and Ack send by receivers. If order receive = incorrect, no reply of Ack.

Resends if not Ack back. Chanel problems are eventually

Distribuidos 2. Inter-object/Process communication

Send/Receive and semantic models

Basic routines:
Send(dest, Message)
receive(Message)
} Message passing

Addressing

Need to identify machine and process. Problem? PID can change in executions → Solution? Use ports
How to know services provided in a node? → A registry stores references attached to a port. Just 1 process to 1 port
↳ A service, offered by using a known port for clients.

Superserver: Server that listens to the ports. Only works when a request comes

Remote object/process identification

Need a unique mechanism to identify (and locate) distributed objects. Diff instances of same program can be identific
(Even if connected to the same port). IPs, port, date creation or PID for identification. Problem? Location dependent
Solution: Store initial node info about actual ubi

Blocking Semantic (4)

- Rendez-Vouz: Send and receive block when partner not ready for communication
- Non-blocking send and blocking receive. Message store in queue
- Non-blocking send and receive: Polling. Return boolean if correct or not
- Blocking send and receive: Not used

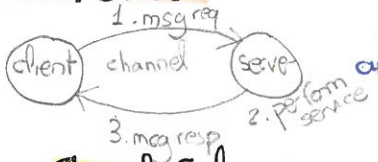
Send/receive with buffering: Message data is copied from user space to kernel space (send), and viceversa (receive)

Advanced semantic models

Blocks when the buffer is full (congestion). Time between copying from memory areas cannot be neglible (streaming)
Send models: Blocking send, waiting until transfer is complete/data copied in buffer OR sender's synchrony (ACK)
Used MPI (Message Passing Interface)
1-Blocking-Synchronous send: Sender blocking until the transfer is complete (receiver sends Ack)
2-Non-blocking-synchronous send: Send routine returns control when starting copy from user space to buffer.
Receiver sends Ack when finished
3-Blocking-Asynchronous send: Send blocked until transfer to buffer in sender is completed. Transmission can still
4-Non-blocking-Asynchronous send: As soon as copy to internal buffer starts the send routine returns control.
We can use "wait" and "test". No Ack expected

Receive models
1-Blocking receive: Wait until transfer is completed and data is copied from buffer to the user space
2-Non-blocking receive: Returns control inmediately. Completition must be checked seperetly

Client/Server



Client request some service, server does it and responds with the service done

- Channel failures
  R-Model: Client sends message but doesn't waits for Ack
  RR-Model: Client sends msg and response is used as Ack. Improved by Time-outs and logs
At least once: Improvement for Idempotent operations → After timeout, client resends msg. After N times it gets failure
At most once: Improvement for non-Idempotent operations → We can't repeat operation, check logs
  RRA-Model: ACKs included. Server keeps in log responses by order and deletes it when client receives
- Server failures: May fail before or after request, for the client is like the response is lost. Rechecked by logs
- Client failures: Can happen orphan computations and is hard to revert. Can be restarted

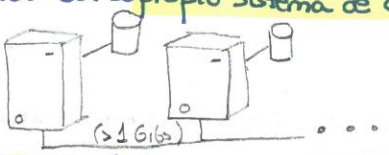
Distribuidos 4. Modern distributed technologies

Clustering

Cluster: Conjunto de computadores interconectados que realizan juntos un conjunto de procesamientos. Son homogéneos, escalables con coste reducido. Fácil migración de procesos y datos. Alta velocidad (>1 Gib/s)

Configuraciones posibles:

a) Computador con su propio sistema de almacenamiento secundario, sin acceso remoto



b) Almacenamiento secundario único con RAID

c) Almacenamiento propio pero con acceso remoto

Cluster functions:
- Gestión de fallos: Tolerancia
- Compartición de carga: Todas con un trabajo similar
- Procesamiento paralelo
- Seen as one: Gracias a un middleware

Migración de procesos: paso de una estación a otra para continuar su ejecución. Control: equilibrio de cargas para igualar nivel de trabajo

Aspectos problemáticos:
a) Ficheros abiertos en nodo origen, su acceso posterior será más complejo
b) Bloqueos sobre réplicas, se debe mantener la coherencia con los ficheros
c) Coherencia de las cachés de disco
d) Grupos de procesos muy interrelacionados: Genera un alto tráfico en la red

Grid and cloud

Grid computation: Conjunto de sistemas heterogéneos para realizar una labor. Con diferentes recursos, están alejados y se comunican por internet. Requirements:
- R1. Must be remote access to the resources
- R2. Each node provide computational service
- R3. A servant on each node is in charge of attending to the incoming requests
- R4. Support Metadata for clients
- R5. Directory system to locate resources
- R6. Software for community uses

Example: SETI@Home, vida alrededor

CLOUD computing: Sistema distribuido, colección de computadores interconectados que ofertan y gestionan de forma dinámica a gran cantidad de usuarios SLA (Service-Level Agreement). Un servidor almacena y los clientes descargan.

a) Características:
- i Autoservicio bajo demanda
- ii Acceso ubico
- iii Agrupación de recursos
- iv Dinamismo en la gestión
- v Pago por uso

b) Service model:
- i. Software as a Service (SaaS): Access to software and databases in the Cloud (Office365)
- ii. Platform as a Service (PaaS): Virtualize apps (Microsoft Azure)
- iii. Infrastructure as a Service (IaaS): Computing and storage (Dropbox)

c) Cloud deployment models:
- i. Private Cloud: For a company
- ii. Community Cloud: Share by many organizations
- iii Public Cloud: Offered by Internet, usually paying
- iv. Hybrid Cloud: Mix of them

d) Key elements:
- i. Massive scalability
- ii. Virtualization
- iii. Autonomous computation (IoT or Tesla)
- iv. Multitenancy: Shared and possibly managed by end users
- v. Geographical distribution: Seen in several places
- vi. Advanced security technologies

Virtualization

Advantages

Las MV dan mayor privacidad
+ Flexibilidad al integrar aplicaciones
Menor consumo energético
Reducción de coste en Software y Hardware

Disadvantages

Disponibilidad condicionada al internet
La info del cliente queda en servidores (Lack of security)
Problemas de escalabilidad

Grid vs Cloud

a) Similitudes
Programación paralela y distribuida, escalable y con load balancing and sharing

b) Diferencias
Cloud cada uno paga, en Grid se reparte
Cloud → IaaS ; Grid → SaaS
Cloud → Seguridad garantizada por VM ; Grid → Mecanismos de autentificación

Fog Computing: Intermedio entre Cloud y centralizada. Reduce latencia. Soporte masivo de datos (IoT)

Edge Computing: Procesamiento cerca de donde se necesitan los datos. Busca reducir latencia y tiempos de respuesta. No necesita la nube

Web Services

Estandarizado por dos organizaciones: W3C (WWW Consortium) and OASIS. Infraestructura más potente que un servidor web simple. Interacción C/S pueden ser autónomas. características

a) Interoperabilidad: Con interfaces estandarizados
b) Patrones de comunicación:
→ Intercambio asíncrono de documentos XML
→ Interacciones petición/respuesta
→ Acoplamiento débil (para reducir dependencias): Interfaces
→ Activación de servicios: un Web Service puede estar continuamente en ejecución o bajo demanda
→ Transparencia: Marshalling oculto al usuario

c) Arquitectura
WSDL: Orientada a servicios, uso de XML. Indica tipo de datos, formato, protocolo de comunicación... SOAP: Capa intermedia que estandariza los mensajes y cómo deben ser procesados. Soporte RPC y puede usarse solo para la transmisión de documentos
UDDI: Registro para la localización de servicios. Poco uso, cada empresa tiene el suyo
WS-BPEL: Estandarización de OASIS. Lenguaje muy completo, que impone la lógica de negocio. Implementación de servicios descritos en WSDL, soporte a los tipos de datos y lógica de control potente
WS-CDL: Describe la lógica de intercambio de mensajes. Jerarquía, intercambio de msg estructurados. Tratamiento de excepciones y finalizadores

d) Implementación en Java
JAX-WS (Servidor): API para implementar servicios web. Genera automáticamente WSDL y SOAP. Se ejecuta como un servlet dentro de un contenedor de servlets (Glassfish)
JAVA SE (Cliente): El cliente usa un proxy para acceder al servidor

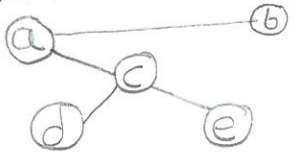
Blockchain

Sistema P2P. Conjunto de nodos que acuerdan el estado del sistema. Se parte de un estado inicial y se construye una cadena de transacciones. El código MDC (Modification Detection Code) garantiza la validez de toda la cadena.

Bitcoin: Tecnología de bloques para una moneda virtual de forma descentralizada. Cada tío con clave pública y privada. Límite de bitcoins: 21 millones

Uso del nonce: Gran esfuerzo computacional para un nuevo bloque en la cadena
Al recibir una transacción, el nodo receptor la valida y retransmite al resto de nodos de la red. Se comprueba el nodo con la clave pública y los bitcoins actuales.
Consume MUCHA energía, ya que en la mayoría de casos no hay éxito en el proceso

Application level multicast



MESH

TREES

Objective: minimize the aggregated cost, ideal to find a minimal spanning tree, better with iterative algorithms

Algorithms based on epidemic behaviours are like a flood, with no initial or central node

Best case: tree → N-1 arcs

Worst case: Fully connected graph → $\binom{n}{2} = \frac{N!}{2 \cdot (N-2)!} = \frac{N \cdot (N-1)}{2}$ arcs

Mobility

Data and code can move to improve performance. Could be problems while moving, process P is stopped while moving to node N...

Mobile code in client/server interactions to reduce network traffic, server workloads is reduced, quality improves

Security problems: clients execute code, we may limit access to remote code

Mobile agents: Running program that travels from one computer to another carrying out a task. Can replicate themselves and work in parallel

Advantages: Avoid transfers of large amounts of data, replacing local invocations

Drawbacks: Constitute a security threat (access to local) & they are vulnerable