Sistemi distribuiti

Un **sistema centralizzato** è formato da un **singolo nodo**, su cui sono contenuti gli *applicativi* e le basi dati necessarie.

Un **sistema distribuito** è formato da più nodi. Ci sono 2 tipi:

- **Elaborazione distribuita**, cioè che gli applicativi sono in più nodi.
- **Basi dati distribuite**, quando gli archivi sono in più nodi.

I *sistemi distribuiti* hanno vari **Vantaggi**:

- **Affidabilità / Tolleranza ai guasti**, nel caso di un nodo guasto il traffico può venir re-indirizzato ad un'altro nodo.
- **Eterogeneità**, i nodi possono usare più *hardware* e *software* diversi.
  - **Apertura**, i nodi possono usare hardware e software di produttori diversi.
- **Trasparenza**, il sistema, anche se formato da più nodi, appare come un unico sistema all'utente.
- **Scalabilità**, un sistema distribuito è più facile da ingrandire di un sistema centralizzato. (In un sistema centralizzato servirebbe modificare l'hardware/software, invece in uno distribuito basta aggiungere altri nodi)
  - **Economicità**, è più economico aggiungere nodi che cambiare l'hardware e il software di un sistema centralizzato.
    Le CPU non si possono rimpicciolire molto, a causa di problemi di surriscaldamento.
    e **Svantaggi**:
- **Complessità**, i sistemi sono molto complessi, date le differenze *hardware* e *software*. Rende quindi più difficile la **Produzione software**.
- **Sicurezza**, bisogna *proteggere* gli *accessi*, i *nodi* e le *intercettazioni*.

  Tolleranza ai guasti, eterogeneità, trasparenza, scalabilità, economicità, apertura, connettività e collaborazione.
  Sviluppo software, sistemi complessi, sicurezza, comunicazione

Architetture distribuite

Classificazione Flynn

Secondo Flynn, le **architetture hardware si dividono in 4 gruppi**: (in base a come si combinano dati e istruzioni)

- **SISD** (*Single Instruction, Single Data*)
  Sistema centralizzato, *nessun parallelismo*.

Il sistema contiene una sola *CPU*, che evolve un unico programma, cioè un solo flusso di istruzioni, su dati provenienti da un unico flusso.

- **SIMD** (*Single Instruction, Multiple Data*)
  parallelismo a livello dati.
  Abbiamo più CPU, che evolvono un unico programma, dove ognuno elabora dati provenienti da flussi diversi.
- **MISD** (*Multiple Instruction, Single Data*)
  parallelismo a livello di istruzioni.
  Ogni processore evolve *programmi diversi*, cioè ognuno ha un proprio flusso di istruzioni, che elaborano dati provenienti da un singolo flusso.
- **MIMD** (*Multiple Instruction, Multiple Data*)
  Abbiamo più processori, dove ognuno evolve processi diversi, lavorando con un proprio flusso di dati.
  Abbiamo 2 tipi:
- **Multiprocessor**, a ***memoria fisica condivisa***,
  dove tutti i processori condividono lo stesso spazio di indirizzamento in memoria.
  Condividono lo spazio. ma non collaborano.
- **Multicomputer**, a **memoria privata**,
  Dove ogni processore ha una propria memoria privata.

Hanno 3 caratteristiche principali questi 2 tipi:
costo, **distanza dalle memorie** e **gestione degli accessi**

Il primo tipo è più *economico*, ma c'è maggiore *distanza*, e bisogna gestire gli *accessi*. (es. memoria ram)
Il secondo tipo è più *costoso*, con minore *distanza*, e non bisogna gestire gli *accessi*. (es. memoria cache)

Altre Architetture Distribuite

- **Cluster Computing**
  Insieme di **nodi MIMD omogenei**, *a memoria privata*. Le caratteristiche principali sono:
  - Connessi in rete locale, con velocità sui GB/s
  - Hardware potente
  - Potenza di calcolo uguale alla somma dei singoli nodi
  - Sistemi centralizzati, collocati in un *rack* (armadi/scaffali)
    Usati per eseguire **calcoli lunghi o impossibili**.
    Il *controllo*/*monitorazione* dei sistemi può essere fatto da un nodo centralizzato oppure un'applicazione, da cui si può anche assegnare i processi da evolvere.
- **Grid Computing**
  Insieme di **nodi MIMD eterogenei**, *a memoria privata*.
  Il problema è la **divisione del lavoro**, bisogna coordinare le risorse hardware e

software.
Vengono usati per risolvere *problemi scientifici, ingegneristici...*

Esempio: Servizi di ricerca medica. Ce ne sono alcuni dove puoi collegarti con qualunque dispositivo, per offrire potenza di calcolo nella ricerca medica.
Ci sono anche servizi simili per il clima e gli asteroidi.

- **SD Pervasivi**
  Insieme di **nodi piccoli e mobili**, **connessi via rete wireless**.
  Alcuni esempi sono:
  - *Rete domestica*
  - *Domotica*
    Esempio: Le case smart, che possono essere controllate in modo automatico o anche da remoto, con dispositivi come *Alexa*.
    Svantaggio, **privacy**: i dati vengono collezionati e inviati a un sistema centralizzato.
  - *Sistemi sanitari*
  - *Reti di sensori*
    Comunicano i dati a sistemi centralizzati, o li memorizzano in proprie memorie.
    Usati ad esempio per collezionare i dati sulle stime dell'elettricità.
    Esempio: sistemi per il tracciamento della temperatura in una città

Architetture Software

- **Architettura a terminali remoti**
  Tutte le operazioni sono effettuate da un'unità centrale, i terminali vengono usati solamente per chiedere e interfacciare i dati agli utenti.
- **Architettura Client-Server**
  Tutte le macchine hanno capacità di elaborazione,
  i server forniscono servizi aggiuntivi.
- **Architettura Web-Centric**
  Simile alla struttura *Client-Server*, i servizi offerti si trovano sul Web, e i client vengono usati solo per l'interfaccia all'utente.

Middleware

Software che si interpone tra il sistema operativo e le applicazioni, crea un'**architettura multilivello**, usata per poter far comunicare sistemi eterogenei.
Esempio: Wine.

Tecnologie Web

Si dividono in 2 tipi:

- **Client-side**, dove l'elaborazione avviene sul client, tramite software come il browser. Qui vengono interpretati i codici HTML e Java-Script.

- **Server-Side**, dove l'elaborazione avviene sul server, tipicamente un web server come apache. Vengono generalmente interpretati codici PHP, Java Servlet...
  Per visualizzare i contenuti di una pagina server side bisogna generalmente collegarsi al server. O si possono usare server virtuali come XAMPP.

Abbiamo 2 tipi di linguaggi nell'ambiente web:

- **Markup**, usati per strutturare o formattare dati e pagine. HTML (*HyperText Markup Language*), XML (*eXtensible Markup Language*)
  La verifica della struttura dei file XML può essere automatizzata tramite file DTD (*Document Type Definition*) o XMD (*XML Schema*)
- **Programmazione**, come PHP e Java-Script, per creare siti dinamici.

Il web è basato sul modello client-server, di cui ci sono alcune caratteristiche:

- Gli host non sono server o client, ma lo sono i processi che si stanno evolvendo su di loro. L'Host è un actor.
- Il processo server è in ascolto alle richieste tramite il **socket**, una porta accoppiata all'indirizzo IP, usato per individuare quel servizio su quell'indirizzo/host.
  FTP 20, 21
  SSH 22 (+ SFTP)
  HTTP 80
  HTTP 443

Ci sono 2 tipi di comunicazione:

- **Unicast**, il server comunica con un client alla volta, accettando la connessione solo quando non sono già connessi altri dispositivi
- **Multicast**, al server si possono connettere più client contemporaneamente, assegnando a ogni client connesso un numero di porta diverso, per lasciare libero il socket per altre richieste di connessione

Le applicazioni hanno 2 architetture principali:

- **Modello Three-Tier**, diviso in:
  - **Presentation Layer**, dove vengono eseguite le procedure di *acquisizione e presentazione* dei dati all'utente
  - **Business/Application Logic Layer**, dove viene *elaborata la logica* (l'algoritmo che esegue le operazioni per completare la richiesta)
  - **Resource Management Layer**, dove vengono eseguite le procedure per *memorizzare e recuperare informazioni persistenti* dagli archivi dati
    Offre il vantaggio che i 3 layer possono essere modificati/sostituiti indipendentemente dagli altri, offrendo *scalabilità e manutenzione*.
    I diversi layer possono essere *distribuiti su diversi nodi, di una rete eterogenea*.
- **Architettura a 2 livelli**, che si divide in:

- **Thin-client**, dove il server è *responsabile per la logica applicativa e la gestione dei dati*. Il client è *responsabile solo per la presentazione*
- **Thick-client**, dove il client è *responsabile per la presentazione e la logica applicativa*. Il server *gestisce solo gli archivi dati*

Applicazioni di rete

Socket

Coppia **indirizzo IP: numero di porta**.
I socket vengono usati dai vari servizi, come HTTP, SMTP, FTP...
Ogni servizio usa numeri di porta diversi.

I **numeri di porta** si dividono in 3 parti:

- **well-known port numbers**
  Sono i numeri da **0** a **1023**, riservati ad applicazioni particolari come HTTP, SSH...
  Usati solo dai server in apertura passiva

I numeri da **1024** a **49151** sono registrati e usati da alcuni servizi, e possono anche essere usati dai client.
Generalmente usati dai client una volta che viene stabilita la connessione.

I numeri da **49152** a **65535** sono invece delle porte libere, dinamicamente assegnate dai processi applicativi.

Alcune porte note:

Porta logica	Protocollo di rete
7	ECHO
21/tcp	FTP
22/tcp	SSH
23/tcp	Telnet
25/tcp	SMTP
42	WINS
53	DNS
80/tcp	HTTP
110/tcp	POP3
143/tcp	IMAP
161	SNMP
443/tcp	HTTPS

TCP - UDP

TCP	UDP
orientato alla connessione, prima di trasmettere i dati, negozia la connessione tra mittente e destinatario. (*3-way handshake*)	**connectionless** (senza connessione), i dati vengono inviati senza verificare che la connessione sia stata stabilita
affidabile, garantisce la consegna dei segmenti. I dati vengono consegnati ordinati	**non affidabile**, non viene neanche verificato che i dati arrivino in ordine
Controllo di flusso e controllo di congestione	**Nessuna funzionalità sul controllo di flusso**

Invece, hanno in comune i 2 protocolli:

- **Funzionalità di controllo di errore sui pacchetti**, tramite il campo *checksum*
- **Multiplexing** delle connessioni, grazie ai *numeri di porta*

TCP viene generalmente usato quando è necessario avere garanzie sulla consegna dei dati (come i trasferimenti di file).
UDP viene usato quando ci sono vincoli sulla velocità. (streaming, video-chiamate, gioco online...)

Comunicazione Web - HTTP

Il Web è basato sul modello **Client-Server**.

- Il *Client* è un **elemento attivo**, che richiede le risorse tramite gli **URL**
- Il *Server* è un **elemento passivo**, che rimane in ascolto, attendendo richieste dai *Client*
  Comunicano tramite il **protocollo HTTP**.

Con lo sviluppo di versioni successive, in HTTP viene introdotta la **connessione permanente**.
Le connessioni possono essere di 2 tipi:

- **Incanalata**, cioè che il Client può inviare richieste consecutive, che fanno parte di una pipeline e le risposte rispettano l'ordine di arrivo
- **Non incanalata**, il Client invierà nuove richieste solo se ha ricevuto la risposta della precedente

Formato richiesta HTTP:

- **Riga di richiesta**
  - **Metodo** (*GET, POST*...)
  - **URI**, identificatore della risorsa

    - **Versione** protocollo
- **Intestazione**
  Formata da diverse righe, con formato *nomeHeader : valore*, informazioni usate per identificare il messaggio
- **Corpo del messaggio**

Alcuni esempi di parametri usati nell'intestazione della richiesta sono:

- **Transfer-Encoding**, che indica la compressione dei dati trasmessi
- **Content-Type**, tipo di contenuto (testo, immagine...)
- **User-Agent**, che contiene informazioni sul browser ed il sistema operativo del Client
- **Host**, nome di dominio del server, e la porta TCP sulla quale il server è in ascolto

Metodi HTTP:

METODO	DESCRIZIONE
GET	Richiesta di un file dal server
HEAD	Richiesta dell'header di una risorsa
POST	Invio di informazioni all'URL specifico, per essere poi elaborate dal server
PUT	Upload di un file sul server
DELETE	Eliminazione di un file sul server
OPTIONS	Richiede l'elenco dei metodi concessi dal server
TRACE	Tracciamento di una richiesta

Formato riposta HTTP:

- **Riga di stato**
    - Versione HTTP
    - Codice di stato
- **Intestazione** (header)
    - Stesse informazioni degli header di richiesta
- **Corpo messaggio**
    - I dati trasporti dal server al client

Codici di stato:

Codice	Tipo	Significato
100-199	*Information*	Informazioni temporanee sulla richiesta, durante il suo svolgimento (codici in disuso)

Codice	Tipo	Significato
200-299	*Successful*	Completamento di una richiesta con successo
300-399	*Redirection*	Stati che non indicano un errore, ma che richiedono un trattamento particolare da parte del browser
400-499	*Client error*	Condizioni di errore provocate dal Client
500-599	*Server error*	Condizioni di errore verificate sul Server