# Sistemi distrubuiti

Monday, 20 March 2023 08:13

-> Un insieme di computer autonomi (hardware, software) comunicano tra di loro
e si coordinano Attraverso un passaggio di informazioni

- Questo insieme di computer all'utente finale compare come 1 singolo computer
- Singoli computer = Nodi
- Questo gruppo può essere aperto/chiuso sulla base se i pc si possono rimuovere/aggiungere liberamente
- Devono essere programmati affinchè riescano a collaborare
- Se 1 computer cade, il sistema non deve cadere
- I vari nodi devono essere indipendenti, ed essi possono essere:
  - Logicamente indipendenti, aka tutti indipendenti e tutti hanno un servizio che fanno da soli
  - Composition, dove ogni componenti performa delle tasks collaborando con altri

Caratteristiche:

- Gestione della memoria
  Siccome ogni pc è autonomo, bisogna avere una memoria virtuale condivisa
  Che si può ottenere con, ad esempio, uno scambio di messaggi
- Gestione dell'esecuzione
  Ogni pc è autonomo, quindi c'è un esecuzione concorrente
  Questo comporta dei problemi in sincronizzazione delle attività
- Gestione del tempo
  Essendo ogni pc diverso, non abbiamo un clock globale
  Questo si può coordinare solo attraverso lo scambio di messaggi
- Fallimenti
  Come detto prima, se 1 nodo crasha, il sistema non deve crashare

Per far avvenire una coerenza ed una comunicazione, si è creata l'architettura del software:

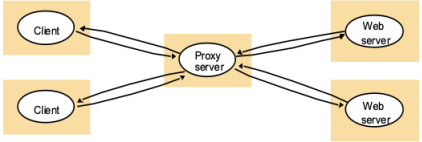
- Definisce la struttura del sistema, le interfacce ed i componenti
- Le varie tipologie sono:
  - Tier
  - Basata su oggetti
    Aka noi mettiamo degli oggetti nelle macchine e tra di loro riescono a comunicare
  - Centrata sui dati
    Aka il web come idea di Explorer dei file
  - Eventi
    Aka il web come chiamate dinamiche
  - Strati
    - Ogni strato comunica con gli altri, e questa comunicazione varia a seconda della struttura.
    - Ogni strato è costruito sopra ad un altro
      Più uno strato è alto, e più è specificializzato

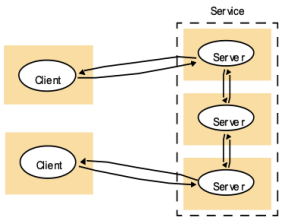
    Ed esistono varie tipologieL
    - Pure layer, qui il layer di sopra possono comunicare con quelli di sotto
    - Mixed layer, i layer di sopra possono comunicare con ognuno dei layer sotto
    - Mixed downcalls and uppercalls, qui i layer fanno ciò che vogliono

Tipologie di sistemi distribuiti:

- DOS
  - Funzione di riuscire a nascondere all'utente l'esistenza di più macchine
    E di gestire le risorse hardware
  - Per passare i dati, si possono utilizzare 3 tecnologie:
    - Data migration, trasferiamo data transferendo file interi
    - Computer migration, noi trasferiamo delle computazioni
    - Process migration, dove trasferiamo inteeri processi
- NOS
  - Funzione di offrire servizi locali a clienti remoti
  - Il client sa dell'esistenza di più macchine
    Ex. TeamViewer, remote desktop
  - C'è una connessione diretta tra i processi
    Ex. Socket
- Middleware
  - Funzione di implementare dei servizi per farli utilizzare alle applicazioni
  - Memoria condivisa
  - E' tipo un canale di comunicazione fra processi
    Quindi le macchine devono avere lo stesso software
    Stesso software $\neq$ Sistema operativo

- Client-Server
  - Client fa una richiesta
  - Client aspetta per una risposta
  - Serve fa una risposta
  - Può essere utilizzato per implementare un sistema distribuito
    Attraverso un cluster che può essere:
    - Server multipli, dove i nostri server sono interconnessi
    - Proxy, dove 1 serve funge da proxy, e poi smista nei vari server




Nota: Meccanismi $\neq$ Politiche

- Meccanismi = Capacità di determinati componenti (nodi)

- Politiche = Come viene implementato

Possono esistere diverse politiche per 1 meccanismo.
Per questo si comprende che:

- Più una politica è uguale dal meccanismo, e più il tutto diventa generico e non facilmente comprensibile
- Più una politica è diversa dal meccanismo, più è complicata e difficile da maneggiare

Bisogna trovare un giusto bilancio il file configurazione è abbastanza custom però non troppo complicato
Ex.

- Context switch = Meccanismo
- Round robin = Politica

I problemi del sistema distribuito

- Il client per comunicare con il server deve:
  - Sapere il nome del server
  - L'indirizzo IP -> Come raggiungerlo
  - Il protocollo che verrà utilizzato
    - Capire le richieste, che definisce il formato, ordine, tipo dei dati ed azioni dei messaggi.
    - Questo protocollo è spesso definito con una libreria comune
    - Ex.
      TCP/IP
      Noi abbiamo un meccanismo che ci scambiamo stream di byte
      E viene messo in pratica con la politica dei messaggi
  - Devono essere capaci di comprendere ciò che verrà ricevuto
- Transparenza, il server deve riuscire a nascondere dei procedimenti che stanno venendo internamente:
  - Naming, il nome con cui si utilizza per identificare le risorse che fanno parte del sistema
  - Access, quando la tipologia cambia da pubblica a privato ex. Proxy
  - Location, dove le risorse sono
  - Relocation or mobile transparency, mettiamo caso siamo in treno, il server non serve sapere che noi stiamo continuando a cambiare pozione
  - Migration, qui invece, quando siamo in treno cambiamo continuamente l'antenna di accesso, ed il server non importa saperlo
  - Replication, quando tipo un server cambia ed il proxy prima ci porta da A, e poi da B
  - Concurrency, che tipo il server è fatto da un cluster
  - Failure, se un nodo cade il sistema non deve crollare
  - Persistance, nascondere che i dati sono volatili/permanenti

  Raggiungere una transparenza totale è impossibile, direttamente opprure indirettamente:
  - Una transparenza completa costa in performance
  - Se un nodo fallisce, c'è più carico in meno computer e quindi si và più lenti
  - A volte è utile che non esiste una trasparenza totale

  Tutto questo processo viene chiamato information hiding, e possiamo notare che esso si divide in:
  - Chi nascondiamo, che è definito da delle interfacce che vengono definite con le API
  - Come nascondiamo, che è definito con dei tool (middleware, framework)