Sistema Distribuito

Definizione di Sistema Distribuito

Un sistema distribuito è una collezione di elementi di computazione autonomi che appare ai suoi utenti come un unico sistema coeso.

1.1 Caratteristiche degli SD

- **Elementi di computazione autonomi**, chiamati nodi, sono dispositivi HW o processi SW.
- **Unico sistema coeso**: utenti o applicazioni lo percepiscono come un unico sistema, quindi i nodi hanno bisogno di collaborare tra di loro.
- **Comportamento indipendente**: ogni nodo è autonomo e per questo hanno la propria nozione di tempo. Non esiste un clock globale, quindi porta ad avere una necessità di sincronizzazione e problemi di coordinazione.
- **Collezione di nodi**: necessità di gestire un sottoinsieme (coeso) di nodi e tutti i problemi che ne consegue *(qualsiasi nodo può partecipare? Oppure solo alcuni nodi specifici possono? Come si fa a capire se sto comunicando con un membro autorizzato del sottoinsieme?)*. La collezione di nodi opera nello stesso modo, non importa dove, quando o come avvengono le interazioni tra gli utenti e il sistema.

Riassunto

- Gestione della memoria
  - No memoria condivisa
  - Comunicazione via scambio messaggi
  - Non esiste uno stato globale. Ogni componente conosce solo il proprio stato.
- Gestione dell'esecuzione
  - Ogni componente è autonomo: esecuzione concorrente
  - È importante il coordinamento delle attività
- Gestione del tempo (temporizzazione)
  - Non c'è clock globale
  - Non c'è possibilità di controllo/scheduling globale
  - Solo coordinamento via messaggio
- Tipi di fallimenti
  - Fallimenti indipendenti dei singoli nodi
  - Non c'è fallimento globale

Distribution transparency

Abilità del sistema nel mettere a disposizione accesso uniforme a risorse distribuite e servizi, nascondendo il dettaglio su dove e come tali risorse sono fisicamente collocate o accessibili.
In altre parole, la distribution transparency permette agli utenti, sviluppatori e applicazioni di interagire con un sistema distribuito come se fosse una singola entità centralizzata, senza la necessità di essere a conoscenza della sua complessità.

Problema: fallimenti indipendenti dei singoli nodi.
È inevitabile che in ogni momento una parte del sistema distribuito può fallire. Il problema sta nel fatto che nasconderli o la loro recovery stesso è molto difficile e in generale impossibile da nascondere. Non c'è un fallimento globale.

1.1.1 Architettura del software

> **Architettura del software**
>
> Una architettura del software definisce la struttura del sistema, le interfacce tra i componenti e i pattern di interazione (i protocolli). I sistemi distribuiti possono essere organizzati secondo diversi stili architettonici.

1.1.1.1 Tipologie di architettura

- Architettura a strati (layered)
- Sistemi operativi
- Middleware
- Architetture a livelli (tier)
- Le applicazioni client-server (2-tier, 3-tier)
- Architetture basate sugli oggetti
- Java-Remote Method Invocation (RMI)
- Architetture centrate sui dati
- Il Web come file sistema condiviso
- Architetture basate su eventi
- Applicazioni Web dinamiche basate su callback (AJAX)

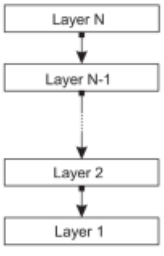
Architettura a strati

Una architettura a strati è una architettura che organizza i software su strati. Ogni **strato** è può essere definito come un insieme di sottosistemi degli stessi tipi dello strato sottostante (che è di un tipo più generico).
In sintesi:

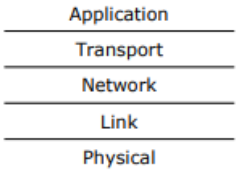
- *strati superiori -> applicazioni più specifiche*
- *strati inferiori -> applicazioni più generali*

Tipologie di architettura a strati

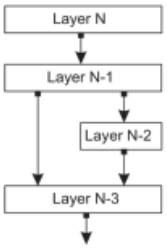
- **Pure layered organization**
  Lo strato superiore può comunicare solo con lo strato inferiore



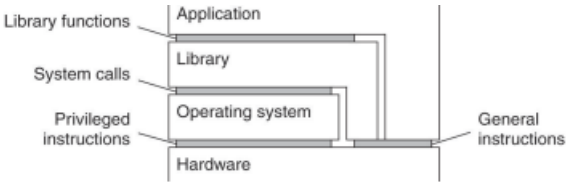
  Esempio: ISO/OSI



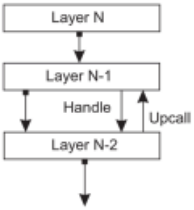
- **Mixed layered organization**
  Lo strato superiore può comunicare con più strati inferiori



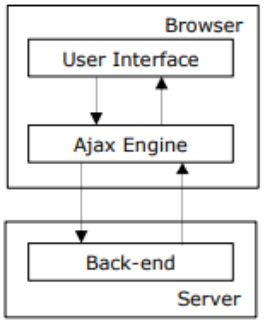
*Esempio: OS



- **Mixed downcalls and upcalls**
  Uno strato può comunicare con strati sia superiori che inferiori



Esempio: Web app



1.1.2 Sistemi Operativi dei Sistemi Distribuiti

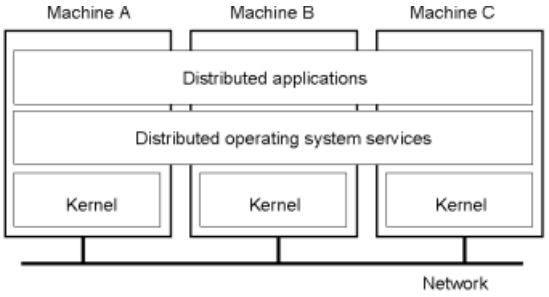
Sistema	Descrizione	Main Goal	Esempio
DOS	Sistema operativo che gestisce le risorse di più computer omogenei interconnessi (devono far andare lo stesso software)	Nasconde e gestisce le risorse hardware. Offre servizi trasparenti alle applicazioni (Data Storage, Esecuzione di processi etc.).	
NOS	Sistema operativo che gestisce le risorse di più computer eterogenei interconnessi (LAN e WAN). I computer possono eseguire software diversi	Offre servizi locali esplicitamente gestiti dalle applicazioni (Data storage, esecuzione di processi etc.) a client remoti	MacOSX, Windows, Linux etc.
Middleware	Uno strato aggiuntivo sopra il NOS che implementa servizi general-purpose	Implementa servizi (uno o più) per renderli trasparenti alle applicazioni (Distribution transparency).	Java, RMI

Distribuited Operating System (DOS)

> **DOS**

Un sistema operativo distribuito è un tipo di SO che gestisce le risorse di più computer omogenei interconnessi e li coordina per lavorare insieme come un'unica entità. Nasconde e gestisce le risorse hardware.

Contrariamente ai sistemi operativi tradizionali che operano su un singolo nodo o computer, un DOS distribuisce il carico di lavoro e le risorse tra più nodi all'interno di una rete consentendo una maggiore scalabilità, affidabilità e prestazioni. È progettato per gestire le risorse di calcolo e di stage su una rete di computer interconnessi.



Caratteristiche dei DOS

- Gli utenti non sono al corrente della molteplicità di macchine;
  - Si può accedere alle risorse remote come a quelle locali.
  - Un DOS nasconde la complessità della rete e dei nodi individuali agli utenti e ai programmi, fornendo l'illusione di un sistema operativo centralizzato.
- **Data Migration**
  È possibile trasferire un intero file oppure solo porzioni del file utili per la corrente task.
- **Computation Migration**
  Trasferimento della computazione tra i sistemi (invece che di dati).
- **Process Migration**
  - Esecuzione di un intero processo (o parte di esso) in posti differenti.
    - *Load balancing* - Distribuzione di processi nella rete per pareggiare la quantità di lavoro;
    - *Computation speedup* - i sottoprocessi possono eseguire concorrentemente in locazioni diverse;
    - *Hardware preference* - le esecuzioni di processi possono aver bisogno di processori specializzati;
    - *Software preference* - i software necessari potrebbero essere disponibili solo in locazioni particolari;
    - *Data access* - esecuzione di processi in remoto invece di un trasferimento di tutti i dati localmente.

Riassunto

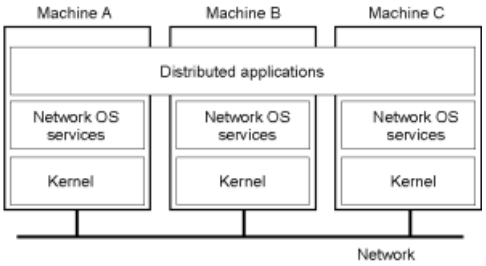
Tipo di sistema operativo progettato per distribuire il carico su **più server hardware** del computer.
Offre **prestazioni e disponibilità** migliori perchè è distribuito su più componenti. In altre parole, è un sistema operativo che viene utilizzato quando più calcolatori sono collegati tra loro e si vuole ottimizzare l'utilizzo delle risorse, evitando che esse risiedano in una singola macchina.

In un DOS è necessario garantire uno **spostamento efficace dei dati, computazioni e processi tra sistemi** in modo tale da poter proseguire nella loro esecuzione.
Essendo mirato a distribuire il carico su più server hardware, è necessaria anche una **distribuzione dei processi nella rete** e garantire che **i sottoprocessi possano eseguire concorrentemente in posti diversi**, oltre che permettere l'esecuzione dei processi anche da remoto invece che tutte localmente.
Infine, è importante anche la **condivisione delle risorse** in quanto alcuni processi possono aver bisogno di processori specializzati situazioni in determinati posti oppure di un software disponibile solo in locazioni particolari.

Network-Operating System (NOS)

NOS

Un sistema operativo network è un tipo di sistema operativo che gestisce più computer (eterogenei=lontani). Offre servizi locali a client remoti.



A differenza di SO tradizionali che gestiscono principalmente le risorse di un computer singolo, un NOS gestisce le risorse di rete come file, stampanti, server e dispositivi di archiviazione, oltre a fornire funzionalità di comunicazione tra i computer all'interno della rete.

Caratteristiche dei NOS

- Gli utenti sono a conoscenza della molteplicità di macchine.
- NOS mette a disposizione delle features di comunicazione:
  - Diretta comunicazione tra processi (socket);
  - Esecuzione concorrente (indipendente) di processi da applicazioni distribuite;
  - I servizi come process migration sono gestiti dalle applicazioni.
- Accesso a risorse di molteplici macchine è effettuato da:
  - Log remoto alla corretta macchina remota (Telnet, ssh);
  - Desktop remoto (Microsoft Windows);
  - Trasferimento dati da macchine remote a macchine locali attraverso ai meccanismi di File Transfer Protocol (FTP).

Riassunto

Un **sistema operativo di rete** è un software o uno strumento che consente di creare un'**interfaccia utente per una rete locale** (LAN), connettendo tra loro vari dispositivi all'interno della rete senza la necessità di accesso a Internet. In altre parole, un NOS è progettato per gestire le **operazioni di rete**, consentendo ai computer e ai dispositivi di comunicazione, condividere risorse come file, programmi e stampanti, e collaborare all'interno di una rete locali.

Gli utenti sono a conoscenza delle macchine interconnesse nel sistema. Il NOS deve **gestire le comunicazione** tra i dispositivi nella rete, consentendo loro di **scambiarsi dati e informazioni** (features di comunicazione) e deve **permettere l'accesso alle risorse di tutte le macchine connesse e condividerlo tra gli utenti all'interno della rete**, controllando che solo gli utenti autorizzati possano accedere alle risorse di rete.

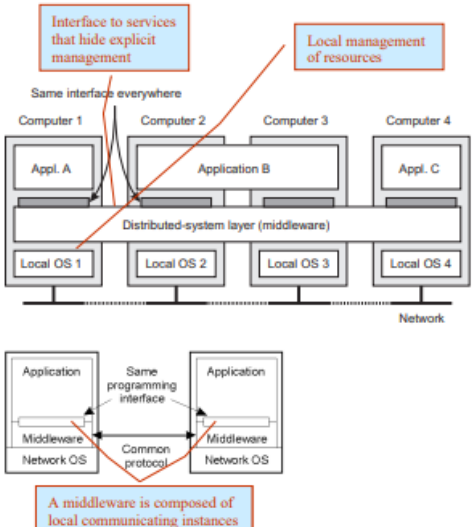
Middleware

È uno strato aggiuntivo sopra il NOS che implementa servizi general-purpose. Il suo obiettivo è la "distribution transparency".

I servizi possono essere di diversi tipi, dai più generici ai più specifici del dominio.

Caratteristiche del Middleware

- **Naming**
  - I nomi simbolici sono usati per identificare entità parte di un sistema distribuito.
  - Possono essere usati dai registri come i reali indirizzi (DNS, registri RMI) o implicitamente dal Middleware.
- **Access transparency**
  Definisce e offre un modello di comunicazione che nasconde i dettagli di un trasferimento di messaggi.
- **Persistence**
  Definisce e offre un servizio automatico per data storage (in un File System o in un DB).
- **Distribuite transactions**
  Definisce e offre un modello persistente per garantire consistenza per operazioni di lettura/scrittura (solitamente in un DB).
- **Security**
  Definisce e offre modelli per la protezione dei dati da accessi indiscreti, servizi (con diversi livelli di permessi) e integrità delle computazioni.



Comparazione tra sistemi

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

1.2 Considerazioni sugli SD

Ogni sistema distribuito deve affrontare *quattro problemi*:

1. **Naming**: come vengono assegnati i nomi ai processi o alle risorse? Abbiamo bisogno di un nome.
2. **Access point**: come posso raggiungere un processo o una risorsa remota? Abbiamo bisogno di una reference.
3. **Protocol**: come fanno i partecipanti a scambiarsi i messaggi? Si deve concordare un formato standard.
4. Si deve concordare una sintassi e una semantica dei dati. È ancora un problema irrisolto.

1.2.1 Distribution transparency

Distribution transparency

La distribuzione trasparente si riferisce a un'implementazione di sistemi distribuiti in cui l'utente o l'applicazione non è consapevole della complessità della distribuzione stessa. In altre parole, l'utente o l'applicazione non deve preoccuparsi di dove si trovano le risorse o come vengono gestite.

- **Naming**: ad ogni risorsa del sistema distribuito viene assegnato un nome per identificarlo;
- **Access transparency**: nasconde le differenze tra le rappresentazioni di dati e come si accede ad una risorsa locale o remota;
- **Location transparency**: nasconde dove una risorsa è situata nella rete;
- **Relocation or mobility transparency**: nasconde il fatto che una risorsa possa essere spostata in un'altra posizione mentre è in uso;
- **Migration transparency**: nasconde il fatto che una risorsa possa essere spostata in un'altra posizione;
- **Replication transparency**: nasconde il fatto che una risorsa viene duplicata;
- **Concurrency transparency**: nasconde il fatto che una risorsa possa essere condivisa da tanti utenti indipendenti (si assicura che lo stato sia consistente);
- **Failure transparency**: nasconde il fallimento e il recovery di una risorsa;
- **Persistence transparency**: nasconde il fatto che una risorsa sia volatile o salvato permanentemente.

È efficace una trasparenza totale?

Potrebbe essere eccessivo puntare ad una totale Distribution transparency:
• C'è **latenza nella comunicazione**;
• È **impossibile nascondere completamente i fallimenti della rete**:
◦ Non riesci a distinguere un computer lento da uno che sta fallendo;
◦ Non puoi essere mai sicuro se un server stesse effettuando una operazione prima di un crash.
• Una trasparenza totale **costa molto a livello della performance**
• Può essere utile esporsi per alcuni servizi basati sulla posizione oppure quando si ha a che fare con utenti situati in time-zone diversi.

1.2.2 Information Hiding

Information Hiding

Concetto di progettazione software che prevede la restrizione dell'accesso diretto alle informazioni interne di un modulo, di una classe o di un oggetto, mantenendo invece solo l'accesso attraverso un'interfaccia ben definita. L'obiettivo principale è isolare i dettagli implementativi e proteggere le informazioni sensibili, consentendo agli altri moduli di interagire con l'oggetto solo attraverso operazioni consentite dall'interfaccia pubblica.

Information Hiding è una separazione tra "*cosa*" deve mettere a disposizione un componente o un sistema e "*come*" un servizio è stato implementato e può essere utilizzato.

- "Cosa" è definito da un Interface Definition Languages (IDL) per definire l'Application Programming Interface (API) dei componenti o sistema. Deve seguire una determinata semantica;
- "Come" è implementato da un tool (es: framework, middleware) adatto per quel specifico problema o ambiente. Deve essere implementato con degli algoritmi e tecnologie specifiche ed efficaci.

Le **interfacce** devono essere:

- Progettati secondo dei *principi comuni* (es: meccanismi di comunicazione);
- *Completi* (offrono tutto ciò che serve);

- *Neutrali* (indipendentemente dall’implementazione o rilascio). Il supporto deve essere interoperabile, portabile, estendibile.