

COSE DA FARE:

- mettere a posto appunti su sistemi distribuiti in generale
 - finire di controllare slides, prox slide asw410-sistemi-distribuiti di luca cabibbo - architetture dei sistemi distribuiti
 - continuare slide introduzione con proprietà/vantaggi e svantaggi legati alla distribuzione
- vedere sezione articoli/siti web per selezionare materiale per lavoro di gruppo
- fare appunti su client/server n-tier (evoluzione architettura client server). prendere da Elena
 - trovare modo di far capire bene concetto di business layer perché.. che ne sanno loro? difficile da immaginare
 - vedere queste slide che sembrano carine Modulo3a.ppt

articolo:

<https://www.freecodecamp.org/news/a-thorough-introduction-to-distributed-systems-3b91562c9b3c/>

<https://www.freecodecamp.org/news/what-is-cloud-computing/>

<https://www.freecodecamp.org/news/beginners-guide-to-cloud-computing-with-aws/>

<https://aws.amazon.com/it/what-is/distributed-computing/>

<https://vitolavecchia.altervista.org/differenza-tra-grid-computing-e-cluster-computing-in-informatica/>

https://it.wikipedia.org/wiki/Grid_computing

<https://csfrontiers.org/distributed-computing.html>

slides:

<http://www.ce.uniroma2.it/courses/sdcc2324/#slides>

<https://webpages.charlotte.edu/abw/GridComputingBook/Slides.html>

siti web:

<https://wbigger.github.io/book-tpsi-5y/>

<https://sistemicorsoc.altervista.org/TPSIT/architetture%20di%20rete.pdf>

<https://www.whymatematica.com/2017/10/10/tpsit-aspetti-evolutivi-delle-reti/>

http://new345.altervista.org/Dispense/Sistemi_distribuiti_caratteristiche.pdf

<http://lucalaurinoblog.blogspot.com/2017/06/tpsit-app-inventor-e-le-applicazioni.html>

Prof Luca Cabibbo <http://cabibbo.inf.uniroma3.it/asw/lezioni.html> + repository git con esempi

<https://github.com/aswroma3/asw/tree/main/environments>

video

<https://www.youtube.com/watch?v=hDxIJalhBNw> - cloud computing

Maturità 2024 ripasso TPSI sistemi distribuiti: <https://www.youtube.com/playlist?list=PLzlagn-0PRQz4jWvXJvWqrax196iGxzqi>

Appunti

Riferimenti Libro di testo

- 1.1 I sistemi distribuiti (pp. 2-9)
 - I sistemi distribuiti
 - Classificazione dei sistemi distribuiti
 - Benefici legati alla distribuzione
 - Svantaggi legati alla distribuzione
- 1.2 Evoluzione dei sistemi distribuiti e dei modelli architetturali (pp. 10-21)

Video interessanti

1. <https://www.youtube.com/watch?v=CESKgdNiKJw> 12 MINUTI INGLESE => buono per introdurre concetto sistemi distribuiti
 1. A distributed system is a collection of independent computers that appear to its users as one computer (A. Tanenbaum)
 2. I computer operate concurrently
 3. Computers fail independently => tolleranza al guasto, la presenza di un componente guasto non deve pregiudicare il funzionamento del sistema

4. Computers do no share a global clock (Network Time Protocol, Precision Time Protocol)
5. Concetto di Publish/Consumer
2. <https://www.youtube.com/watch?v=MwGOjSl5iE> 9 MINUTI INGLESE => NOIOSSETTO
 1. Amazon netflix google microsoft facebook: non esiste un computer magico in grado di gestire tutto il traffico in arrivo, questo traffico va distribuito tra computer diversi che agiscono come un solo computer
 2. complessità nascosta all'utente, sembra di interagire con un singolo computer
 1. Caratteristiche
 1. No shared clocks (ogni server ha il proprio "tempo"): problema sincronizzazione, ritardi (latency) non accettabili in certi contesti (es. mercato azionario) (google risolve con atomic clock e gps receiver)
 2. No shared memory (ogni server (o nodo) ha la sua memoria RAM, il suo storage). Se un nodo ha bisogno di dati che non ha, deve chiederli
 3. Shared resources: hardware, software e data condivisi
 4. Concurrency and consistency
 3. Serve un modo per comunicare: protocollo di comunicazione
 3. <https://www.youtube.com/watch?v=wEsUZkt67VI> 3:34 ITALIANO Prof. Pollini => molto base, ni... parla del problema di sincronizzazione con clock diversi ma lo accenna a malapena
 4. <https://www.youtube.com/watch?v=iyzcztESG30> PESSIMO
 5. https://www.youtube.com/watch?v=Sa_gJwB55jY 6:32 INGLESE =>
 1. scalability e performance, 24/7 servizi accessibili da tutto il mondo ha portato a sistemi distribuiti, fault tolerance, problema sicurezza e latenza
 2. definizione sistemi distribuiti
 3. esempio app per prestito libri
 1. rindondanza
 2. communication challenges
 3. coordination challenges
 4. poi sponsorizza tool ma va beh
- 6.

UDA

introduzione con video gelato <https://www.youtube.com/watch?v=CESKgdNiKJw>

continuare con esempio gelati per definire un sistema distribuito e le sue caratteristiche e riportare a esempi informatici (esempio sito che offre servizio a cui accedono tanti utenti vedi IG TikTok Facebook etc.)

- replicazione risorse (rindondanza)
- tolleranza al guasto
- apertura
- scalabilità

Presentare articoli su grid computing o altro

grid-computing - fare presentazione => slides definizione, esempio, slide su differenza con cluster e cloud

<https://www.geeksforgeeks.org/grid-computing/>

<https://aws.amazon.com/it/what-is/grid-computing/>

<https://www.zerounoweb.it/big-data/grid-computing-cose-come-funziona-ed-esempi/>

generale su sistemi distribuiti

<https://learningdaily.dev/what-are-distributed-systems-a-guide-for-beginners-b8f43ad67c17>

RPC call e tutorial in python

<https://www.youtube.com/watch?v=Oua3cMmdHsg>

<https://aws.amazon.com/it/compare/the-difference-between-rpc-and-rest/> differenza REST e RPC presentazione con docker e piccola implementazione

tutorial python <https://medium.com/@taraszhore/coding-remote-procedure-call-rpc-with-python-3b14a7d00ac8>

Sistema Distribuito

Libro di testo:

Unità 1 Lezione 1 - concetto sistema distribuito e proprietà

Unità 1 Lezione 2 - pag. 14-fine capitolo per arch, software

Unità 1 Lezione 2 - pag 10-13 per arch. hardware distribuite

Materiale usato come riferimento

1. Walter Cazzola Dipartimento di Informatica e Comunicazi 01.ds.8.pdf
-

Introduzione ai Sistemi Distribuiti

2.

- **Reti Locali ad Alta Velocità:** consentono connessioni rapide tra dispositivi all'interno di un'area limitata (LAN).
- **Reti Geografiche:** connettono milioni di computer su vasta scala, spesso attraverso continenti (WAN).

Questa evoluzione permette di utilizzare **sistemi di calcolo distribuito** basati su più computer connessi tramite rete, per aumentare la capacità di calcolo e migliorare l'efficienza operativa.

Un **sistema distribuito** è composto da componenti e strumenti interconnessi tramite una rete di computer.

- Questi componenti comunicano tra loro tramite **messaggi**.

- I messaggi contengono informazioni che possono essere:
 - **Controllo** (istruzioni per il funzionamento del sistema)
 - **Dati** (informazioni da trasmettere)
 - Oppure una combinazione di entrambi, controllo e dati.

I sistemi distribuiti sono tipicamente eterogenei, presentando diversità in:

- **Sistema operativo**
- **Interfaccia di comunicazione**
- **Potenza e capacità di calcolo (CPU)**
- **Protocolli di rete**
- **Posizione geografica e velocità di connessione**

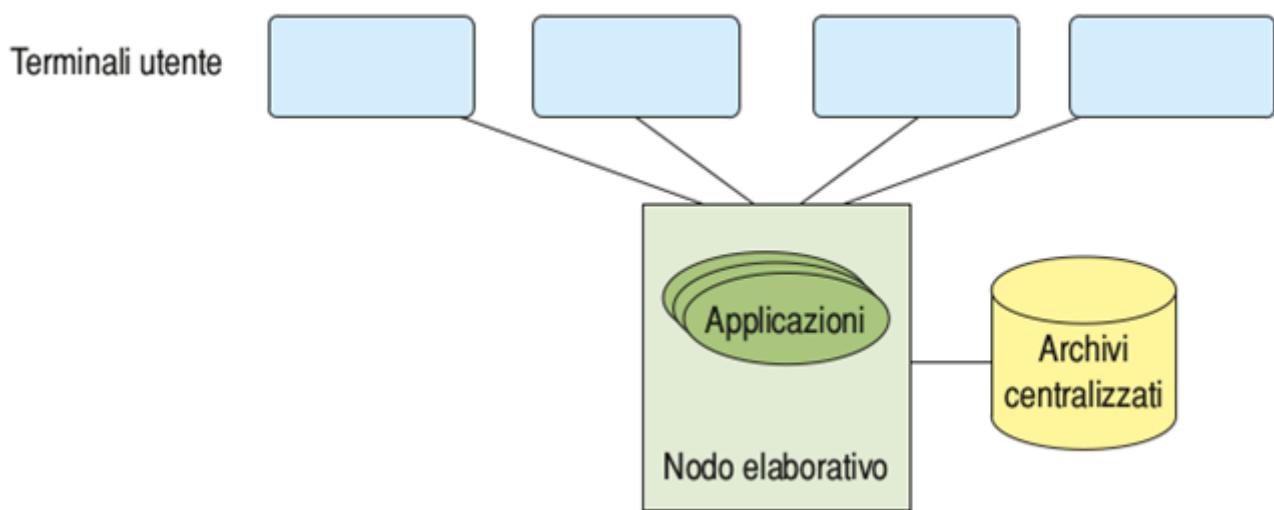
Queste diversità richiedono soluzioni scalabili e adattabili per garantire la coesione e la funzionalità del sistema distribuito.

Sistema Centralizzato

Si parla di **sistema informatico centralizzato** quando i dati e le applicazioni risiedono in un unico nodo elaborativo.

- Le **applicazioni** girano in un singolo processo o su un solo host, che rappresenta l'unico componente autonomo del sistema.
- Questo **componente** è condiviso tra vari utenti.
- **Tutte le risorse** del componente sono sempre accessibili.
- Il componente rappresenta un **single point of control** e un **single point of failure** del sistema.

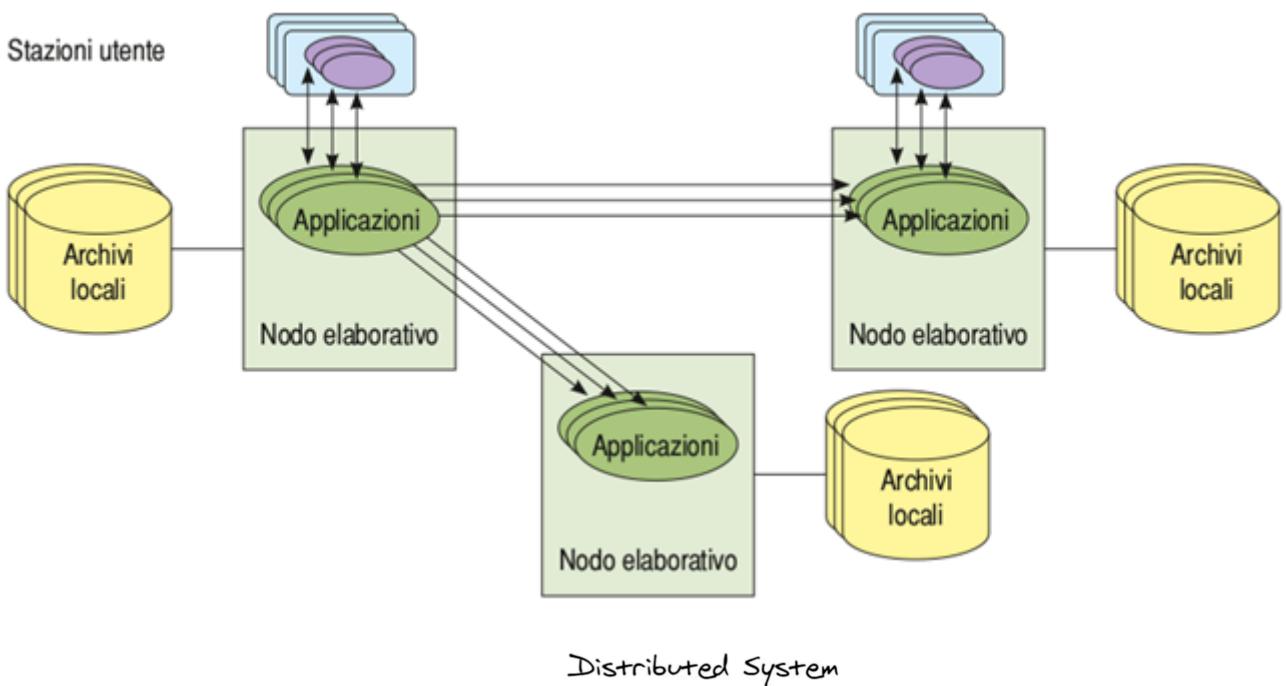
Un'architettura client-server è un classico esempio di sistema centralizzato, come riportato in figura



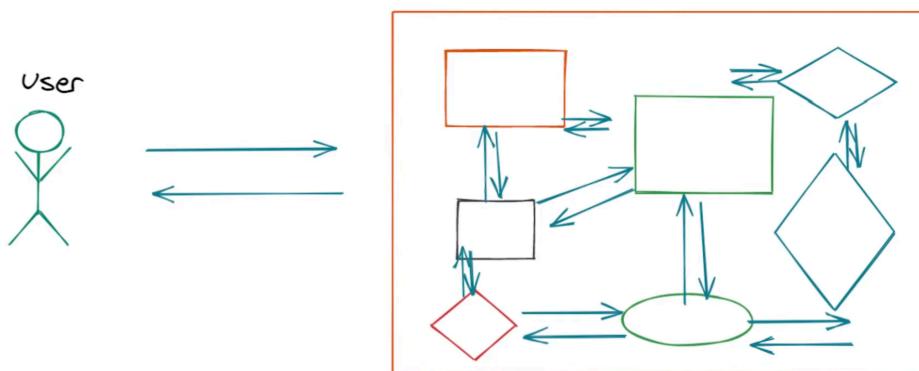
Sistema distribuito

Un **sistema informatico distribuito** invece è costituito da un insieme di applicazioni logicamente indipendenti che collaborano per il perseguimento di obiettivi comuni attraverso una infrastruttura di comunicazione hardware e software.

Definizione: Un sistema distribuito È una collezione di computer indipendenti che appare all'utente come un singolo sistema (A. Tanenbaum)



Distributed System



- Le **applicazioni** sono costituite da più processi cooperanti, eseguiti in parallelo su un insieme di unità di elaborazione (CPU) autonome.
- Un'applicazione si definisce **concorrente** se non viene rispettato il requisito di autonomia delle CPU.

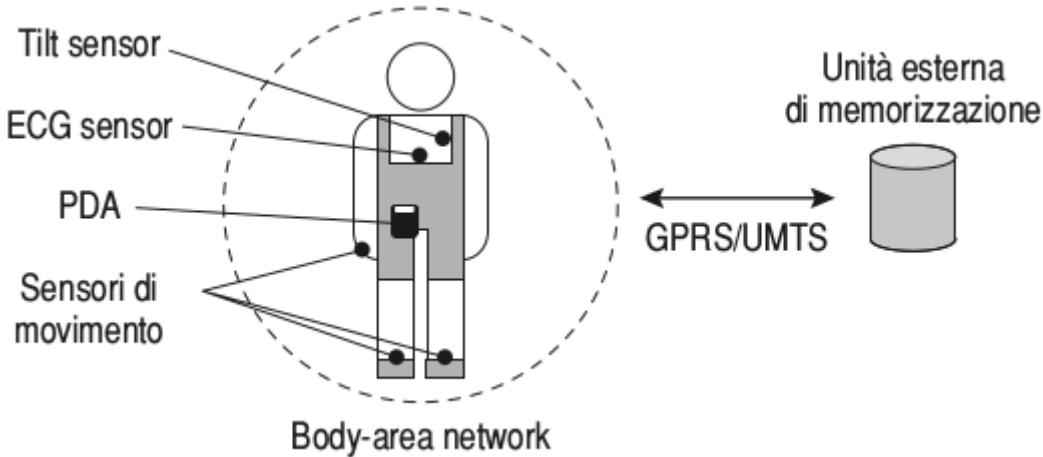
Le applicazioni che costituiscono un sistema distribuito hanno ruoli diversi all'interno del sistema stesso, e sono identificate nel seguente modo:

- **Client**: un'applicazione assume il ruolo di client quando è utilizzatore di servizi messi a disposizione da altre applicazioni;
- **Server**: un'applicazione assume il ruolo di server quando è fornitore di servizi usati da altre applicazioni;
- **Actor**: un'applicazione assume il ruolo di actor quando svolge sia il ruolo di client, sia quello di server in diverse situazioni nel contesto del sistema.

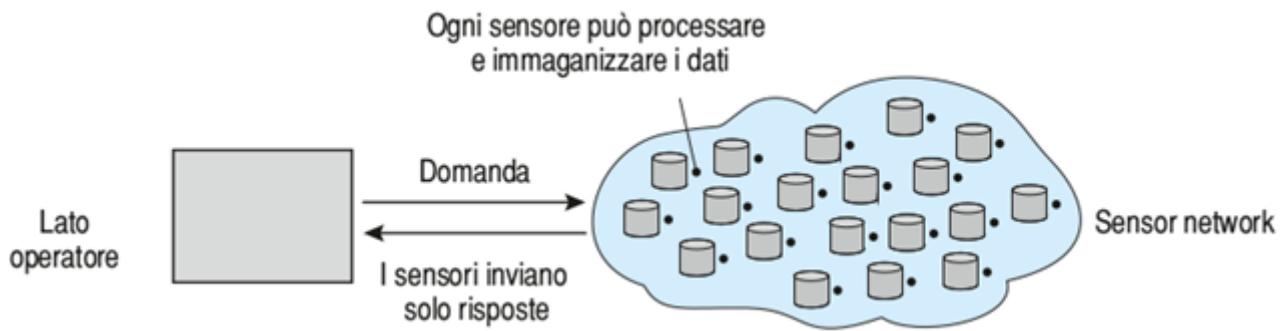
Un esempio comune di sistema distribuito è il Web o Internet. Tuttavia, non tutti i sistemi distribuiti sono confinati a Internet o al Web. Alcuni esempi includono:

- **Intranet aziendale**: una rete privata interna che può essere considerata un sistema distribuito.
- **Server farm**: un insieme di server che operano insieme come un sistema distribuito.
- **Sistema di calcolo parallelo**: un'architettura in cui più unità di calcolo lavorano insieme per risolvere problemi complessi è anch'essa un sistema distribuito.

A livello macroscopico, esempi di sistemi distribuiti includono Internet, mentre a livello individuale possiamo parlare delle PAN (Personal Area Network), spesso associate al wearable computing.



Un altro esempio è una **rete di sensori**.



Vantaggi dei Sistemi Distribuiti

- I **sistemi con grandi capacità di calcolo** sono molto costosi, mentre il costo delle CPU e dell'infrastruttura di comunicazione si è ridotto notevolmente.
- **Facile scalabilità**: è possibile gestire la crescita del sistema (nodi o numero di utenti) in modo incrementale.
- **Distribuzione del carico di lavoro**: il carico può essere suddiviso in modo più efficiente tra le varie macchine.
- **Condivisione di dati e risorse computazionali** tra utenti e processi.
- Alcune applicazioni sono **naturalmente distribuite**, ad esempio i servizi di prenotazione aerea o alberghiera.
- **Maggiore tolleranza ai guasti** grazie alla distribuzione delle risorse e dei processi.

Svantaggi dei Sistemi Distribuiti

- latenza invocazione dei servizi
- "Un sistema distribuito è un sistema in cui il mio programma smette di funzionare a causa di una macchina di cui non ho mai sentito parlare." Leslie Lamport
- Differenze di prestazioni: l'invocazione remota di un servizio ha una latenza superiore di 4-5 ordini di grandezza rispetto a quella locale.
- Fallimenti parziali e concorrenza:
 - Una computazione locale può essere concorrente; una distribuita lo è sempre.
 - Un sistema locale fallisce completamente; un sistema distribuito può subire fallimenti parziali.
 - Nei sistemi locali esiste un unico punto di controllo per allocazione risorse, sincronizzazione e gestione malfunzionamenti. Questo non vale nei sistemi distribuiti.
- Accesso alle risorse: il metodo di accesso alle risorse, ad esempio per i puntatori, cambia.
- Sicurezza: i sistemi distribuiti introducono maggiori vulnerabilità legate alla sicurezza.

L'obiettivo di un sistema distribuito è offrire una visione unica del sistema che in realtà è composto da computer e reti eterogenei. Un sistema distribuito presenta un'organizzazione a strati

1. livello superiore: utenti e applicazioni
2. livello di integrazione: middleware, strato software (vedere dopo integrazione)

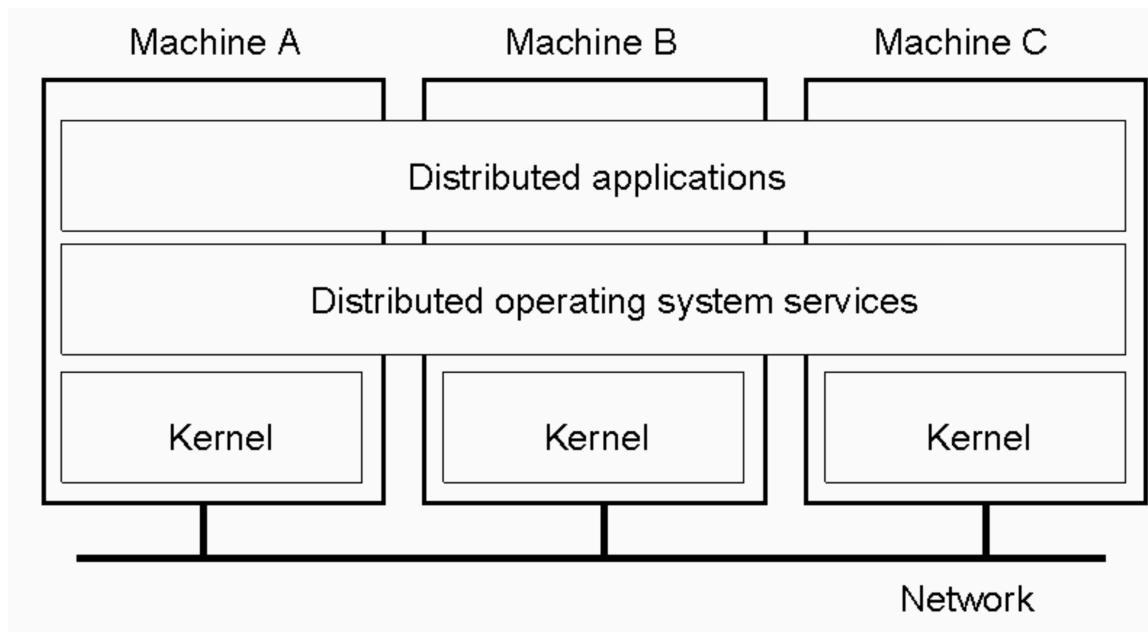
Concetti Software

| Sistema | Descrizione | Obiettivo |
|-------------------|--|---|
| DOS | Sistemi operativi integrati per multi-processori e multi-computer omogenei | <i>Nascondere e gestire le risorse hardware</i> |
| NOS | Sistemi operativi lasciamente accoppiati per multi-computer eterogenei (LAN e WAN) | <i>Offrire servizi locali a clienti remoti</i> |
| Middleware | Livello addizionale sul NOS per implementare servizi general-purpose | <i>Fornire trasparenza di distribuzione</i> |

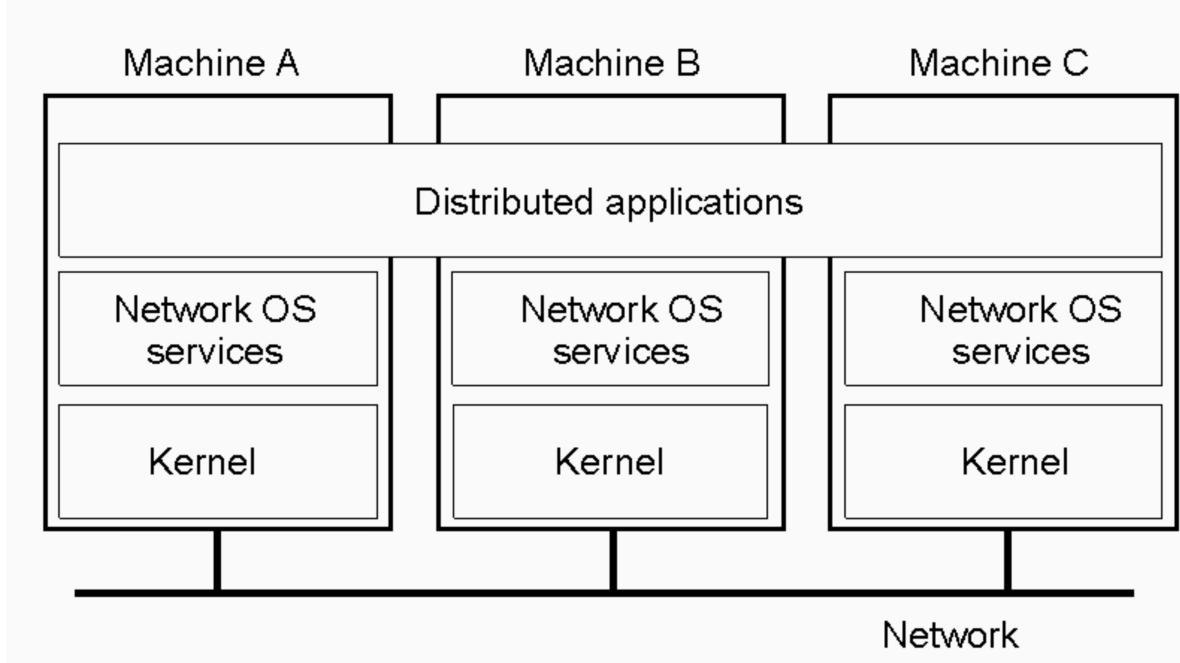
Tre approcci differenti:

- DOS (Distributed Operating Systems)
- NOS (Network Operating Systems)
- Middleware

Struttura generale di un sistema operativo per multi-computer

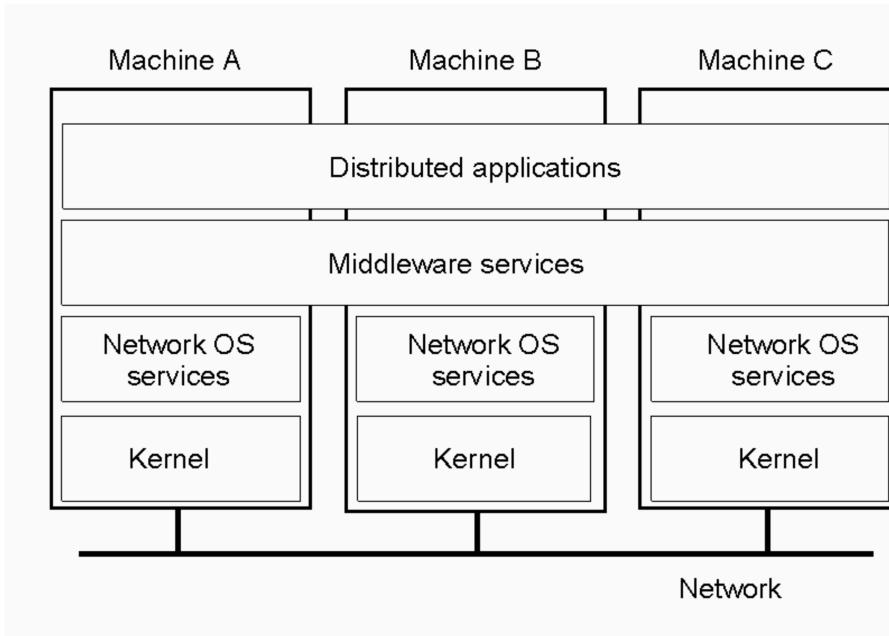


Struttura generale di un NOS e di un NOS + middleware



Struttura generale di un NOS

Posizionamento del Middleware



Struttura generale di un Sistema Distribuito come Middleware

Proprietà Sistemi Distribuiti

Un sistema distribuito deve essere
aperto
scalabile

flessibile
trasparente (ovvero nascondere che le risorse sono distribuite)

Un sistema distribuito è aperto se offre servizi secondo regole standard per descrivere la sintassi e la semantica del servizio. Le regole sono specificate secondo interfacce, che specificano i nomi delle funzioni disponibili e la loro intestazione.

Recap vantaggi e svantaggi

Vantaggi

Benefici della Distribuzione

- **Affidabilità:** Grazie alla distribuzione su più nodi, il sistema è meno vulnerabile a guasti singoli. Un singolo punto di guasto non compromette l'intero sistema.
- **Apertura:** I sistemi distribuiti sono spesso progettati per essere interoperabili con altri sistemi, facilitando l'integrazione di nuove tecnologie e servizi.
- **Connettività:** La natura distribuita consente una maggiore connettività e accesso a risorse globali, come internet.
- **Prestazioni:** La distribuzione del carico di lavoro su più macchine può migliorare le prestazioni complessive del sistema, soprattutto per applicazioni che richiedono un alto livello di parallelizzabilità.
- **Tolleranza ai guasti:** La capacità di continuare a funzionare anche in presenza di guasti a singoli componenti è una caratteristica fondamentale dei sistemi distribuiti.
- **Integrazione:** I sistemi distribuiti possono integrare facilmente nuove risorse e servizi, aumentando la loro flessibilità.
- **Trasparenza:** L'accesso alle risorse distribuite è spesso reso trasparente all'utente finale, che non deve preoccuparsi della loro localizzazione fisica.
- **Economicità:** L'utilizzo di risorse distribuite può ridurre i costi, ad esempio sfruttando risorse inutilizzate o distribuendo il costo dell'hardware su più utenti.

Svantaggi legati alla Distribuzione

- **Complessità:** La gestione di sistemi distribuiti è più complessa rispetto a sistemi centralizzati, richiedendo protocolli di comunicazione, sincronizzazione e gestione delle risorse distribuite.
- **Sicurezza:** La maggiore superficie di attacco e la necessità di comunicare attraverso reti aperte rendono i sistemi distribuiti più vulnerabili ad attacchi informatici.
- **Comunicazione:** La comunicazione tra i nodi distribuiti può essere soggetta a latenze, errori e congestione, influenzando le prestazioni del sistema.
- **Produzione di software:** Lo sviluppo di software per sistemi distribuiti richiede competenze specifiche e l'utilizzo di strumenti e framework adatti.

Spiegazione nel Contesto dei Sistemi Informatici Distribuiti

Un sistema informatico distribuito è un insieme di computer indipendenti, spesso geograficamente dispersi, che collaborano per eseguire un compito comune. Questa distribuzione offre numerosi vantaggi, come una maggiore affidabilità, scalabilità e flessibilità. Tuttavia, introduce anche nuove sfide, come la gestione della complessità, la garanzia della sicurezza e l'ottimizzazione delle comunicazioni.

Esempi di Sistemi Informatici Distribuiti

- **Cloud computing:** L'infrastruttura cloud è un esempio tipico di sistema distribuito, dove le risorse computazionali e di storage sono distribuite su numerosi server.
- **Reti peer-to-peer:** In una rete peer-to-peer, ogni computer agisce sia come client che come server, condividendo risorse e comunicando direttamente con gli altri peer.
- **Sistemi di file distribuiti:** Questi sistemi consentono di accedere a file su computer remoti come se fossero locali, offrendo una maggiore disponibilità e scalabilità.

Affidabilità

Uno dei principali vantaggi dei sistemi distribuiti è l'**affidabilità**, garantita dalla **ridondanza** hardware e software. Quando questa è progettata e implementata in modo sistematico, il sistema può continuare a funzionare anche in caso di guasto di un componente, sebbene con un possibile aumento dei tempi di risposta.

Le risorse possono essere replicate, migliorando:

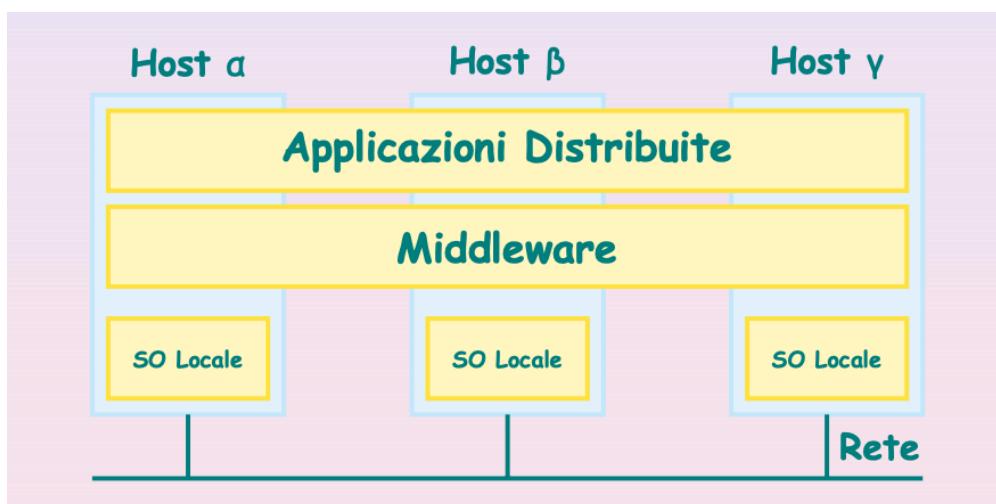
- **Velocità di accesso** (es. tramite cache).
- **Disponibilità** (es. attraverso il load balancing).
- **Affidabilità**, mascherando i fallimenti grazie alla replicazione.

Questa affidabilità è raggiungibile solo predisponendo strumenti hardware e software in grado di intervenire automaticamente in caso di problemi. Ad esempio, l'uso di algoritmi che permettono alle entità funzionanti di sostituire quelle guaste consente di mantenere operativo il sistema, attivando sistemi di backup al momento opportuno.

Tuttavia, è evidente che **realizzare un sistema altamente affidabile comporta costi e complessità crescenti**. Di conseguenza, non sempre è possibile implementare una soluzione completamente affidabile.

Integrazione

Un'altra importante caratteristica dei sistemi distribuiti è la capacità di **integrare componenti eterogenei tra loro**, sia per tipologia hardware (dai PC ai mainframe, dagli smartphone ai tablet), sia per sistema operativo. Infatti, i sistemi distribuiti sono spesso organizzati con uno strato di software (**middleware**) che gestisce in modo trasparente un insieme eterogeneo di computer e reti. Il middleware è uno strato software che si colloca tra l'applicazione e il sistema operativo, con l'obiettivo di astrarre e arricchire le funzionalità del sistema operativo, rendendole accessibili alle applicazioni. In un contesto distribuito, il middleware semplifica la progettazione separando i livelli di sistema e applicazione, facilitando lo sviluppo e la gestione di entrambi. I middleware distribuiti astraggono i dettagli della rete, fornendo API standard per la comunicazione. Consentono agli sviluppatori di interagire con sistemi distribuiti senza preoccuparsi dei dettagli della rete, migliorando la produttività e riducendo i costi di sviluppo.



Esempi di middleware: CORBA, DCE, .NET, PVM, e JVM.

Il middleware:

- Permette di collegare sistemi con diverse architetture hardware e software. (integrazione)
- Supporta formati di dati interoperabili, come **JSON** (snello e moderno) e **XML** (strutturato e robusto). (integrazione)
- Nasconde la complessità della comunicazione e sincronizzazione tra nodi distribuiti. (astrazione)
- Espone funzionalità di alto livello alle applicazioni. (astrazione)

Un middleware è composto da diversi strati. Ogni strato del middleware offre servizi distinti e si occupa di compiti differenti, rendendo più semplice la gestione e l'interazione tra applicazioni distribuite. Noi ci concentreremo principalmente sui primi due strati, ovvero l'infrastruttura dell'host e il middleware di distribuzione, che sono fondamentali per la comunicazione tra i componenti distribuiti.

Il middleware è suddiviso in vari strati, ognuno dei quali fornisce funzionalità specifiche per garantire il corretto funzionamento di un sistema distribuito:

1. Host Infrastructure Middleware

- Posizione: Più vicino al sistema operativo.
- Funzione: Gestisce le risorse hardware di base e incapsula i servizi locali per la distribuzione.
- Caratteristiche: Facilita la comunicazione tra i componenti locali e fornisce un supporto unificato per vari sistemi

attraverso API.

- Esempi: JVM, .NET, e altri modelli locali.
- Utilità: Fornisce l'infrastruttura di base necessaria per l'esecuzione delle applicazioni distribuite.

2. Distribution Middleware

- Posizione: Si trova sopra l'Host Infrastructure Middleware e si occupa della gestione della comunicazione e sincronizzazione tra i nodi distribuiti.
- Funzione: Fornisce modelli di programmazione distribuita e facilita la gestione delle risorse distribuite.
- Caratteristiche: Supporta una comunicazione e un coordinamento efficaci tra i nodi, con funzionalità come:
 - Un modello delle risorse
 - API di comunicazione basate su un modello concettuale
 - Funzionalità accessorie per la comunicazione, supporto ai nomi, e scoperta delle risorse (discovery)
 - Esempi: RMI, CORBA, DCOM, SOAP.
 - Utilità: Facilita la comunicazione tra i nodi, semplificando la programmazione di sistemi distribuiti.
 - Concetto di RPC: REMOTE PROCEDURE CALL ... RMI Java per oggetti remoti, stub client oggetto remoto server marshal/unmarshal oggetti

3. Common Middleware Services

- Posizione: Più in alto, sopra il livello di Distribution Middleware.
- Funzione: Fornisce servizi comuni come gestione delle transazioni e sicurezza, supportando operazioni di livello più alto.
- Caratteristiche: Si concentra su funzionalità addizionali che semplificano lo sviluppo di applicazioni distribuite, come:
 - Gestione degli eventi
 - Logging
 - Streaming
 - Sicurezza
 - Fault tolerance
- Esempi: CORBAservices, J2EE, .NET Web Services.
- Utilità: Supporta una programmazione orientata ai componenti e fornisce una gestione più avanzata delle risorse.

4. Domain-Specific Middleware Services

- Posizione: Più vicino all'applicazione.
- Funzione: Fornisce funzionalità specializzate per specifici domini applicativi.
- Caratteristiche: È progettato per soddisfare esigenze particolari di settori specifici, come:
 - Commercio elettronico
 - Finanza (home banking, assicurazioni, brokeraggio, ecc.)
 - Telemedicina
- Standardizzazione: Gruppi come OMG, HL7, e Siemens stanno lavorando per definire e standardizzare queste funzionalità per diversi settori.
- Utilità: Permette di adattare il middleware alle necessità specifiche di vari domini applicativi, migliorando l'efficienza e la funzionalità.

Dati interoperabili

In un sistema distribuito, l'integrazione tra sistemi eterogenei è un aspetto fondamentale. I **dati interoperabili** giocano un ruolo cruciale in questo contesto, poiché la capacità di scambiare dati tra diverse piattaforme e tecnologie è essenziale. Formati come **JSON** e **XML** sono ampiamente utilizzati per garantire che i dati possano essere trasmessi e compresi da sistemi differenti, facilitando l'integrazione tra componenti eterogenei.

Dati Interoperabili con JSON e XML

- **JSON (JavaScript Object Notation)** e **XML (Extensible Markup Language)** sono due dei formati di scambio di dati più comuni nei sistemi distribuiti.
- **JSON** è preferito in molti ambienti moderni grazie alla sua leggerezza e alla facilità di parsing in molte lingue di programmazione. È utilizzato in ambienti web e applicazioni mobili per il trasferimento di dati tra server e client.
- **XML**, pur essendo più verboso rispetto al JSON, continua a essere utilizzato in ambiti più tradizionali e in scenari che richiedono una descrizione più strutturata dei dati. È ampiamente supportato da sistemi legacy e viene utilizzato in contesti come la comunicazione tra servizi web e sistemi enterprise.

Esempio: REMOTE PROCEDURE CALL

L'RPC è una tecnica che permette di astrarre le chiamate a procedure distribuite come se fossero locali, semplificando la comunicazione in un ambiente distribuito.

Funzionamento di RPC:

1. Client-Side Stub:

Si comporta come un proxy per la procedura remota.

Localizza il server e gestisce la **marshalling** dei parametri, cioè la conversione dei dati in un formato che può essere trasmesso sulla rete.

2. Server-Side Stub:

Riceve la chiamata, **unpacks** (smonta) i parametri, e invoca la procedura sul server.

Questa tecnica maschera la complessità della rete, permettendo agli sviluppatori di concentrarsi sulla logica applicativa.

Esempio tutorial RPC in python: <https://medium.com/@taraszhore/coding-remote-procedure-call-rpc-with-python-3b14a7d00ac8>

Esempio: WEB SERVICE REST

I servizi Web REST si inseriscono tipicamente nello strato di “Domain-Specific Middleware Services”, anche se, a seconda dell’implementazione e della sua architettura, possono essere associati anche ad altri strati.

Inserimento nei diversi strati del middleware:

1. Domain-Specific Middleware Services:

- Questo strato è il più naturale per i servizi Web REST, poiché sono progettati per fornire funzionalità specifiche a determinati domini applicativi, come il commercio elettronico, la gestione di contenuti, o l’integrazione con altre applicazioni aziendali.
- I servizi REST sono utilizzati per creare interfacce di programmazione (API) che permettono alle applicazioni di interagire tra loro in modo semplice e flessibile, sfruttando il protocollo HTTP e formati standardizzati come JSON o XML per la comunicazione dei dati.
- I servizi REST sono tipicamente utilizzati per consentire lo scambio di informazioni in ambienti distribuiti e per implementare la comunicazione tra client e server.

2. Common Middleware Services:

- Se i servizi REST sono implementati per la gestione di funzionalità più generali (ad esempio, gestione delle transazioni, logging, autenticazione, sicurezza, ecc.), allora potrebbero anche rientrare nello strato dei Common Middleware Services.
- In questo caso, i servizi REST potrebbero essere utilizzati come un modo per accedere a questi servizi comuni tramite una API HTTP, sfruttando il protocollo per rendere disponibili operazioni come il controllo degli accessi o la gestione delle transazioni in modo remoto.

Conclusioni:

- REST si inserisce principalmente nello strato di Domain-Specific Middleware Services, poiché fornisce un’interfaccia web per interagire con applicazioni e dati appartenenti a specifici domini.
- Se l’integrazione riguarda funzioni più generali o condivise, allora REST può anche essere associato allo strato Common Middleware Services.

Il middleware, e in particolare il middleware distribuito, gioca un ruolo cruciale nell’astrazione e nella gestione dei sistemi distribuiti, fornendo un’interfaccia uniforme che semplifica la comunicazione tra nodi. La sua suddivisione in strati permette di gestire le risorse e le funzionalità in modo modulare, migliorando la flessibilità e l’efficienza delle applicazioni distribuite.

SOAP e XML - livello distribuzione

SOAP (Simple Object Access Protocol) e XML si collocano principalmente nello strato di Distribution Middleware, anche se possono interagire con gli altri strati in alcuni scenari. Ecco una spiegazione del loro posizionamento:

SOAP nel Middleware

- Posizione: Strato di Distribution Middleware.
- Funzione: SOAP è un protocollo basato su XML progettato per lo scambio di messaggi strutturati tra componenti di un sistema distribuito.
- Caratteristiche:
- Consente la comunicazione tra sistemi eterogenei, grazie al formato standardizzato basato su XML.
- Supporta l'integrazione e la comunicazione tra nodi distribuiti, anche attraverso firewall o reti diverse.
- Utilizza protocolli di trasporto come HTTP o SMTP, rendendolo versatile per ambienti distribuiti.
- Esempi di utilizzo:
- Implementazione di web service in architetture distribuite.
- Comunicazione tra applicazioni enterprise o tra domini diversi.

XML nel Middleware

- Posizione: Presente in vari strati, ma principalmente usato come formato di scambio dati nello strato di Distribution Middleware.
- Funzione: XML (Extensible Markup Language) è utilizzato come linguaggio standard per descrivere e trasmettere dati in modo strutturato e leggibile sia per le macchine che per gli esseri umani.
- Caratteristiche:
- Utilizzato da protocolli come SOAP per la rappresentazione dei dati.
- Garantisce interoperabilità tra componenti distribuiti e piattaforme diverse.
- Può essere utilizzato per rappresentare configurazioni, messaggi o altri dati condivisi tra componenti distribuiti.
- Esempi di utilizzo:
- Definizione e trasporto di payload nei messaggi SOAP.
- Configurazione di sistemi distribuiti.
- Rappresentazione di strutture dati in ambienti distribuiti.

Relazione con gli altri Strati

1. Host Infrastructure Middleware:
 - XML potrebbe essere utilizzato in alcune implementazioni per configurare ambienti runtime come JVM o .NET.
2. Common Middleware Services:
 - SOAP e XML possono essere strumenti per implementare servizi comuni, ad esempio per la gestione di transazioni distribuite o l'autenticazione.
3. Domain-Specific Middleware Services:
 - SOAP e XML sono spesso usati per costruire API che supportano servizi specializzati, come e-commerce o telemedicina.

Conclusione

SOAP e XML sono essenziali nello strato di Distribution Middleware per facilitare la comunicazione e l'interoperabilità in sistemi distribuiti. Grazie al loro design standardizzato e alla capacità di gestire dati in modo flessibile, sono ideali per supportare scenari eterogenei e complessi, estendendosi anche ad altri strati dove necessario.

Legacy System e integrazione

Questo tipo di integrazione eterogenea è fondamentale soprattutto per integrare i legacy System.

Un **legacy system** (sistema ereditato) è un sistema informatico o applicativo che, pur essendo obsoleto o superato tecnologicamente, continua a essere utilizzato in un'organizzazione per svolgere funzioni critiche. Questi sistemi spesso si basano su tecnologie, linguaggi di programmazione o piattaforme che non sono più supportati o aggiornati, ma sono mantenuti in funzione per garantire la continuità operativa, poiché la loro sostituzione potrebbe comportare costi elevati o difficoltà nell'integrazione con i sistemi moderni.

Scenario d'esempio

Immagina un'azienda che utilizza un **sistema legacy** per gestire le operazioni di inventario e ordini, basato su un software antiquato che gira su un mainframe. Questo sistema è ancora fondamentale per le operazioni quotidiane, ma non può comunicare facilmente con le nuove applicazioni cloud-based o con i dispositivi mobili usati dai dipendenti sul campo.

L'**integrazione di sistemi distribuiti** può essere cruciale per consentire a questi sistemi moderni di lavorare insieme al sistema legacy. Ad esempio, attraverso un middleware o un'API, il sistema legacy di gestione degli ordini può essere integrato con un'app mobile che consente ai venditori di accedere in tempo reale alle informazioni sugli stock, aggiornando l'inventario direttamente dal campo.

In questo modo, non è necessario sostituire completamente il sistema legacy, ma si possono sfruttare i vantaggi di un'architettura distribuita (accessibilità, aggiornamenti in tempo reale, nuove funzionalità) pur mantenendo intatti i dati

storici e le funzioni critiche del sistema più vecchio.

Molte aziende scelgono di non rifare completamente un sistema legacy, ma piuttosto di **integrare** il sistema esistente con tecnologie moderne tramite strumenti come **middleware**, **API** o **microservizi**. Questo approccio consente di:

- **Preservare l'investimento** nel sistema legacy, mantenendo la sua stabilità e riducendo i rischi.
- **Integrare funzionalità moderne** senza compromettere il funzionamento del sistema esistente.
- **Evita le interruzioni** a livello operativo, in quanto la migrazione avviene gradualmente e senza interrompere i processi aziendali vitali.

In sintesi, rifare il sistema legacy è possibile, ma richiede una valutazione attenta dei costi, dei benefici e dei rischi. In molti casi, l'integrazione di sistemi distribuiti con il sistema legacy può essere una soluzione più pratica ed economica.

RMI livello middleware distributed verse Web Service REST livello domain o common

Web Services REST

- **Cos'è:**
 - Un'architettura progettata per la comunicazione tra client e server tramite il protocollo HTTP.
 - Utilizza risorse identificate da URI e supporta operazioni CRUD attraverso metodi HTTP (GET, POST, PUT, DELETE).
- **Caratteristiche:**
 - **Stateless**: Ogni richiesta è indipendente, e non conserva informazioni di stato sul server.
 - **Formati di scambio dati**: Tipicamente JSON o XML.
 - **Portabilità**: Può essere utilizzato da qualsiasi linguaggio e sistema che supporti HTTP.
 - **Applicazioni**: API web, microservizi, IoT, mobile, integrazione tra sistemi eterogenei.

RMI (Remote Method Invocation)

- **Cos'è:**
 - Una tecnologia Java-specifica che permette l'invocazione remota di metodi come se fossero locali.
 - Utilizza stubs (proxy lato client) e skeletons (server-side) per gestire la comunicazione e il passaggio di parametri.
- **Caratteristiche:**
 - **Stateful**: Può mantenere il contesto tra chiamate successive.
 - **Serializzazione**: I parametri e gli oggetti sono serializzati per il trasferimento sulla rete.
 - **Java-Centric**: Progettato per sistemi in cui client e server utilizzano Java.
 - **Applicazioni**: Sistemi distribuiti che richiedono stretta integrazione tra componenti.

Confronto tra REST e RMI

| Aspetto | REST | RMI |
|----------------------|--|---|
| Protocollo | HTTP | Protocolli personalizzati su TCP/IP |
| Stato | Stateless | Stateful |
| Interoperabilità | Alta (funziona con qualsiasi linguaggio e piattaforma) | Bassa (specifico per Java) |
| Facilità di utilizzo | Semplice, basta un browser o client HTTP | Richiede setup più complesso |
| Efficienza | Leggermente meno efficiente per overhead HTTP | Più efficiente per ambienti Java omogenei |
| Adattabilità | Adatto per sistemi eterogenei e distribuiti su larga scala | Adatto per sistemi omogenei in Java |

REST nel Middleware

- **Dominio:**
 - Spesso integrato nei **Domain-Specific Middleware Services** per:
 - Fornire API web per applicazioni distribuite.
 - Facilitare l'integrazione tra componenti eterogenei.

- Consentire la comunicazione tra client leggeri (come browser) e server.

- **Utilizzo:**

- Usato anche nei **Common Middleware Services** per:
 - Interfacce di logging, autenticazione, o gestione delle transazioni accessibili via API.

RMI nel Middleware

- **Dominio:**

- Si colloca nei **Distribution Middleware** per:
 - Gestire la comunicazione tra componenti distribuiti in ambienti Java omogenei.
 - Offrire un modello di programmazione che consente agli sviluppatori di lavorare con oggetti remoti come se fossero locali.

Integrazione e Confronto nell'Architettura Middleware

1. **Interoperabilità:**

- REST eccelle quando il sistema coinvolge piattaforme eterogenee (e.g., applicazioni web, mobile, IoT).
- RMI è limitato all'ecosistema Java, rendendolo meno flessibile in contesti eterogenei.

2. **Efficienza e Stato:**

- RMI è più efficiente nei sistemi Java, poiché elimina l'overhead HTTP.
- REST, essendo stateless, è scalabile ma può essere meno performante in ambienti che richiedono una gestione dello stato tra client e server.

3. **Usi Tipici:**

- REST è ideale per servizi distribuiti su larga scala, come API pubbliche o microservizi.
- RMI è più adatto a sistemi interni con stretta integrazione tra componenti Java.

4. **Integrazione Potenziale:**

- In alcune architetture middleware, REST e RMI possono coesistere:
 - **REST** per l'esposizione di API pubbliche e per l'integrazione con sistemi esterni.
 - **RMI** per la comunicazione interna tra componenti Java all'interno dello stesso sistema distribuito.

- **REST:**

- Flessibile e adatto a contesti eterogenei e decentralizzati.

- **RMI:**

- Offre integrazione e performance migliori in ambienti Java-centrici.

Nel middleware, **REST** è spesso preferito per la sua capacità di adattarsi a sistemi moderni, mentre **RMI** può essere vantaggioso in contesti specifici dove l'omogeneità del sistema e l'efficienza sono prioritarie.

Scalabilità

- La **scalabilità** è la capacità di un sistema (distribuito) di migliorare le proprie prestazioni all'aumentare delle risorse disponibili.
- Un sistema scalabile è quello in cui le prestazioni migliorano con l'aggiunta di nuove risorse.

1. **Quantità di risorse:**

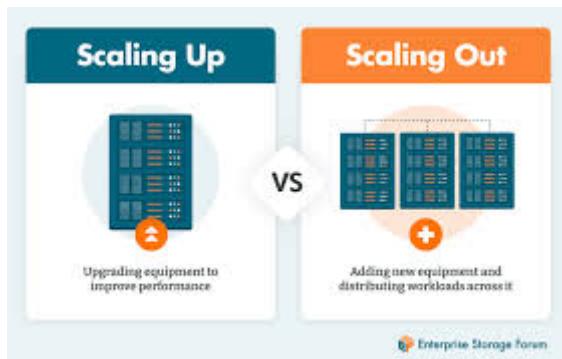
- Coinvolgimento di un numero crescente di risorse.

2. **Distanza geografica:**

- Gestione di risorse distribuite geograficamente.

3. **Amministrazione:**

- Coordinazione di risorse gestite in modo distribuito o con criteri di sicurezza differenti.



Scalabilità Verticale:

- Aggiungere risorse a una singola macchina.
- Esempio: aggiornare un server con più RAM, CPU, o spazio di archiviazione.
- **Vantaggio:** semplice da implementare.
 - **Limite:** c'è un tetto massimo alle risorse che un singolo server può gestire.

Scalabilità Orizzontale:

- Aggiungere più macchine per condividere il carico di lavoro.
- Esempio: smistare il traffico utenti su più server replicati.
- **Vantaggio:** maggiore affidabilità e flessibilità.
 - **Problemi:**
- Gestione della consistenza tra repliche.
- Maggiore complessità nell'architettura.

Difficoltà nella Scalabilità

RISORSE

- **Collo di bottiglia:**
 - Un server centralizzato può diventare il punto critico del sistema.
- **Soluzioni centralizzate obbligate:**
 - Esempio: gestione di conti correnti.
- **Vantaggi dei sistemi distribuiti:**
 - Esempio: DNS utilizza un approccio distribuito per scalare meglio.
- **Problemi nelle reti Peer-to-Peer:**

- Generano traffico significativo tra i nodi, complicando la scalabilità.

DISTANZA GEOGRAFICA

- **Latenza:**

- La comunicazione sincrona è penalizzata su WAN, dove i tempi sono molto più lunghi rispetto a LAN.

- **Affidabilità:**

- Le WAN sono meno affidabili e mancano di meccanismi di **broadcast** o **multicast**.

SICUREZZA

- La scalabilità può coinvolgere sistemi amministrati con criteri di sicurezza differenti, complicando la gestione complessiva.

MIGLIORARE LA SCALABILITÀ

Nascondere la Latenza

- 1. **Comunicazione asincrona:**

- Non attendere la risposta dal server; il client utilizza il tempo di attesa per altre operazioni.

- Esempio: il client è notificato quando la risposta arriva.

- 2. **Spostare parte della computazione al client:**

- Esempio: validazione dei dati immessi direttamente lato client.

Distribuzione del Servizio

- Scomporre una componente in parti più piccole da distribuire nel sistema.

- Esempio: il sistema DNS è progettato come distribuito.

Replicazione

- **Miglioramento delle prestazioni:**

- Le copie vicine riducono i tempi di latenza (es. cache locale).

- Le richieste vengono distribuite tra repliche per garantire tempi di risposta migliori.

- **Problema della consistenza:**

- La sincronizzazione delle copie può risultare complessa, specialmente in sistemi ad alta intensità di scrittura.

- La scalabilità è una proprietà fondamentale, ma complessa da ottenere nei sistemi distribuiti.

- **Chiavi per il successo:**

- Bilanciare tra scalabilità verticale e orizzontale.

- Ottimizzare la distribuzione del carico e minimizzare i problemi di consistenza.

- Progettare architetture che tengano conto di latenze, affidabilità e sicurezza.

Trasparenza

Con il termine **trasparenza** si intende il concetto di **considerare il sistema distribuito non come un insieme di componenti ma come un sistema di elaborazione unico**.

Lo scopo è quello di rendere trasparente all'utente la presenza di un sistema distribuito composto da molteplici entità, anche fisicamente distanti fra loro e con configurazioni mutevoli nel tempo, in modo che abbia invece la percezione di utilizzare un singolo elaboratore.

L'ANSA nell'ISO 10746, *Reference Model of Open Distributed Processing*, identifica otto forme di trasparenza, delle quali le prime due risultano fra le più importanti da implementare nell'ambito un sistema distribuito:

- **trasparenza di accesso** (*access transparency*): permette l'accesso alle risorse locali e remote usando operazioni identiche e uniformi, nascondendo le differenze nella rappresentazione dei dati e nelle modalità di accesso;
- **trasparenza di locazione** (*location transparency*): permette l'accesso alle risorse utilizzando un identificatore indipendente dalla reale disposizione geografica della risorsa stessa, ad esempio utilizzando un URL;
- • **trasparenza di concorrenza** (*concurrency transparency*): permette ai processi di operare in maniera concorrente per l'accesso ad una risorsa condivisa, lasciandola sempre in uno stato consistente implementando, per esempio, meccanismi di locking (semafori, monitor);
- **trasparenza di replicazione** (*replication transparency*): permette di aumentare l'affidabilità e le prestazioni del sistema effettuando duplicazioni delle risorse mantenendone sempre lo stesso nome, e senza che l'utente ne abbia la percezione;
- **trasparenza ai guasti** (*failure transparency*): permette di mascherare un eventuale guasto di una risorsa e l'eventuale ripristino;
- **trasparenza alla migrazione** (*mobility transparency*): nasconde l'eventuale spostamento (logico o fisico) di una risorsa senza interferire sulla sua modalità di accesso, e quindi senza influenzare le operazioni degli utenti che la riguardano;
- **trasparenza alle prestazioni** (*performance transparency*): nasconde le operazioni necessarie per riconfigurare il sistema al variare del carico, allo scopo di migliorarne le prestazioni;
- **trasparenza di scalabilità** (*scaling transparency*): permette di espandere il sistema senza interromperne o modificarne il funzionamento.
lizzazione di un sistema centralizzato di un tempo, basato su mainframe, il costo di un sistema distribuito risulta sicuramente maggiore di diversi ordini di grandezza, ma a differenza della vecchia tecnologia, il costo procapite dell'uso del sistema viene ripartito su un maggior numero di soggetti, fornendo anche un **margine di guadagno** di molto superiore.

Inoltre la possibilità di **condividere** risorse hardware e software comporta vantaggi economici dovuti alla condivisione di apparecchiature speciali di costo elevato, che diversamente potrebbero risultare sotto-utilizzate.

Infine un sistema distribuito ben progettato dovrebbe risultare facilmente **scalabile** permettendo l'aggiunta di componenti al fine di migliorarne le prestazioni realizzando un bilanciamento di carico.

Complessità

La **complessità** di un sistema distribuito ha **richiesto lo sviluppo di hardware** opportuno per garantire l'interconnessione dei suoi componenti e l'instradamento dei messaggi, in modo da fronteggiare adeguatamente la crescente richiesta dell'utenza.

Inoltre ha portato allo **sviluppo di nuovi paradigmi di programmazione e di nuovi linguaggi**, modificando radicalmente l'idea di programmazione e di sviluppo di sistemi software.

Infine sono sorte molteplici **problematiche legate alla sicurezza** che un tempo non era possibile neanche prevedere; l'accesso remoto alle risorse e lo sviluppo delle transazioni commerciali ha reso necessario lo sviluppo di sofisticate tecniche volte alla protezione dei dati e delle infrastrutture.

Problematiche sicurezza: informazioni sensibili, comunicazioni tracciabili

Classificazione dei sistemi distribuiti - software

Sistemi distribuiti moderni sono basati su diverse tecnologie, tra cui:

- **Internet, intranet, mobile computing e ubiquitous computing**
- **Bluetooth**
- **Reti mobili** di vario tipo
- **IoT (Internet of Things)**

Come per l'hardware, anche per il software si è avuta avuto una evoluzione nelle architetture distribuite che spesso hanno anticipato i cambiamenti delle architetture hardware e dei loro sistemi operativi,

Client-server

L'architettura **Client/Server** è un modello computazionale basato sull'interazione tra due tipi di processi:

- **Client**: processi che richiedono servizi;
- **Server**: processi che forniscono tali servizi.

Caratteristiche principali dell'interazione

- **Asimmetria**: il client invia una richiesta al server, il quale la elabora e restituisce un risultato.
- **Evoluzione naturale della programmazione modulare**: questo modello consente di separare la logica di elaborazione tra le due entità.
- **Comunicazione half-duplex**: il flusso avviene dal client al server e poi ritorna al client.
- **Flessibilità nei ruoli**: in alcuni casi, i ruoli di client e server possono invertirsi.

Definizione di Client

Il **client** è un'entità (come un processo o un oggetto) che invia richieste a un server per ottenere un servizio, come:

- l'esecuzione di un metodo;
- l'elaborazione di dati.

Tipicamente, il client rappresenta il **front-end** dell'applicazione, ovvero la parte con cui l'utente interagisce direttamente.

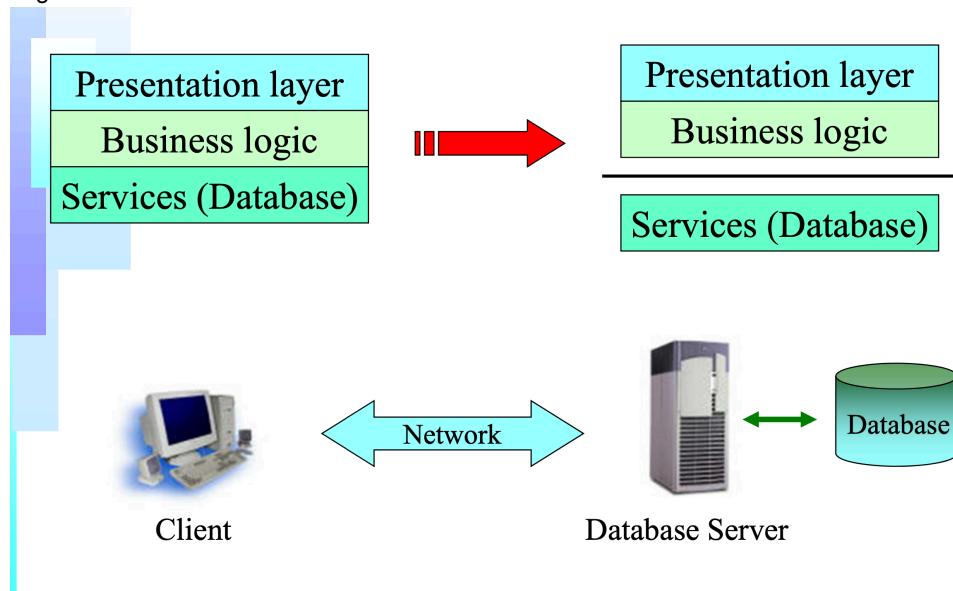
Definizione di Server

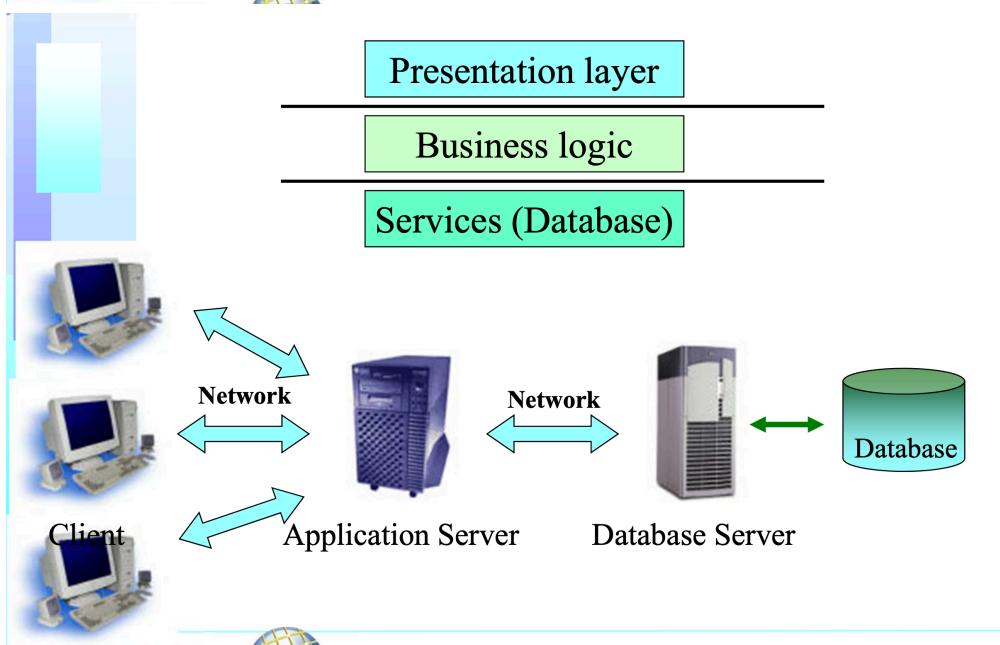
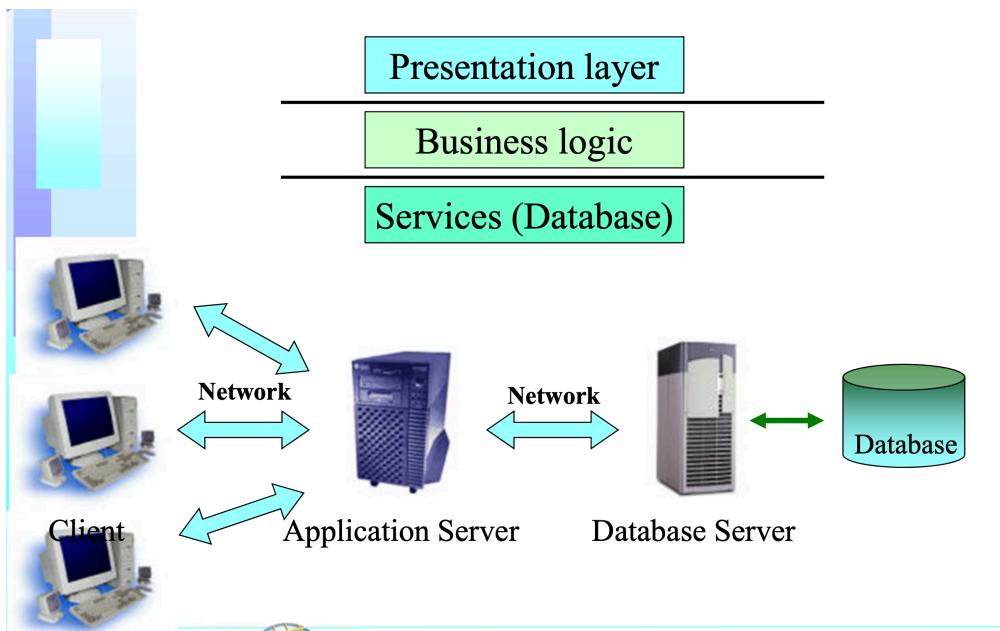
Il **server** è un'entità (processo o oggetto) che risponde alle richieste dei client fornendo i servizi richiesti.

- Può essere locale al client o eseguito su una macchina remota.
- Costituisce il **back-end** dell'applicazione, dove avvengono le principali elaborazioni.

Evoluzione dell'architettura

1. **Architettura centralizzata**: i servizi e i dati risiedevano in un unico sistema centrale.
2. **Architettura client-server**: i processi client e server sono separati, spesso su macchine diverse.
3. **Architettura multi-tier (o distribuita)**: include più livelli intermedi (ad esempio, database, applicativi, servizi), migliorando scalabilità e modularità.





Peer2Peer

Grid Computing

Esempio BOINC

- Sistemi distribuiti: [BOINC](#)

BOINC (Berkeley Open Infrastructure for Network Computing) è una piattaforma open-source sviluppata dall'Università di Berkeley che consente la distribuzione di calcolo su larga scala attraverso **computing distribuito**. In altre parole, BOINC permette di sfruttare il potere di calcolo inutilizzato di computer sparsi in tutto il mondo per eseguire simulazioni o risolvere problemi complessi.

BOINC Italy <https://www.boincitaly.org/calcolo-distribuito.html>

Come funziona BOINC:

BOINC permette a chiunque abbia un computer con accesso a Internet di partecipare a progetti scientifici. Gli utenti possono scaricare e installare un'applicazione cliente sul loro dispositivo che esegue operazioni di calcolo per conto di progetti scientifici specifici.

Caratteristiche principali:

- Progetti scientifici:** BOINC è utilizzato per sostenere progetti di ricerca in vari campi, come la **fisica**, la **biologia**, l'**astronomia**, la **medicina** e altri. Alcuni progetti noti includono SETI@home (ricerca di segnali radio provenienti dallo

spazio), Rosetta@home (simulazione delle strutture proteiche), e World Community Grid (ricerca per risolvere problemi globali come malattie e povertà).

2. **Calcolo distribuito:** La piattaforma consente di suddividere compiti di calcolo complessi in piccole unità che possono essere eseguite su migliaia o milioni di computer, riducendo enormemente il tempo necessario per completare le simulazioni.
3. **Scelta del progetto:** Gli utenti possono scegliere su quali progetti scientifici lavorare, e BOINC gestisce la distribuzione dei compiti e il ritorno dei risultati. Gli utenti possono anche decidere di partecipare a più di un progetto contemporaneamente.
4. **Computing remoto:** Il calcolo avviene su **macchine remote** (computer personali, server, etc.) e non richiede intervento umano diretto, una volta che l'applicazione è in esecuzione. I risultati vengono inviati ai server del progetto per l'elaborazione finale.
5. **Risorse del computer:** BOINC utilizza solo le risorse di calcolo non utilizzate dal computer, in modo da non interferire con le operazioni quotidiane dell'utente. Può essere configurato per funzionare solo quando il computer è inattivo o durante determinati orari.

Vantaggi di BOINC:

- **Contributo alla scienza:** Consente a chiunque abbia un computer di partecipare a ricerche scientifiche di importanza globale.
- **Efficienza:** Utilizza risorse di calcolo inutilizzate in tutto il mondo, rendendo molto più veloce il calcolo di simulazioni complesse.
- **Accessibilità:** La partecipazione è gratuita e aperta a tutti, e il software è disponibile per Windows, Linux e MacOS.

Conclusioni:

BOINC è una piattaforma che consente di unire le forze di milioni di computer per risolvere problemi scientifici complessi che richiederebbero enormi risorse computazionali, democratizzando la ricerca e rendendo la scienza accessibile a chiunque.

Cluster Computing

<https://top500.org/>

Esempio BlueGene

<https://www.ibm.com/history/blue-gene>

Esempio Leonardo

<https://leonardo-supercomputer.cineca.eu/it/home-it/>

articolo su wired: <https://www.wired.it/economia/business/2020/10/15/supercomputer-leonardo-italia-europa/>

Cloud Computing