Sistemi distribuiti: introduzione

SCD

Anno accademico 2019/2020
Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@unipd.it

Definizione

- **Un sistema distribuito è un insieme di nodi di calcolo indipendenti capaci di apparire all'applicazione come un sistema unitario e coerente**
  - **La comunicazione di coordinamento tra i nodi è trasparente all'applicazione**
  - **L'interazione tra applicazione e nodi non dipende dal tempo locale e dalla locazione in cui avviene**

Le dimensioni della trasparenza

Trasparenza di	Per nascondere
Accesso	Differenze nella - rappresentazione dei dati (per HW eterogeneo) - modalità di accesso a risorse (per organizzazioni logiche diverse)
Collocazione	Il luogo di residenza effettiva delle risorse (distinzione tra nome fisico e nome logico)
Migrazione	Che una risorsa possa cambiare collocazione nel tempo
Spostamento	Che una risorsa possa cambiare collocazione durante l'uso
Replicazione / Transazione	Esistenza di copie multiple di una risorsa Coordinamento di attività per gestire una configurazione di risorse
Malfunzionamento	Guasto ed eventuale ripristino delle risorse
Persistenza	Grado di persistenza della risorsa logica (residente in memoria primaria oppure in memoria secondaria)

ISO/IEC 10746-1:1998, *Open Distributed Processing – Reference model: Overview*

Altre caratteristiche desiderabili – 1

- ***Openness***
  - **Portabilità e interoperabilità**
  - **Modalità di invocazione definita secondo regole pubbliche e stabili**
    - Servizi sintatticamente specificati in termini di **interfacce** espresse in linguaggio neutro (***Interface Definition Language***, IDL)
    - **Completezza**: la specifica di interfaccia non nasconde dettagli essenziali alla sua realizzazione da parte di terzi
    - **Neutralità:** la specifica di interfaccia non impone una specifica realizzazione

Altre caratteristiche desiderabili – 2

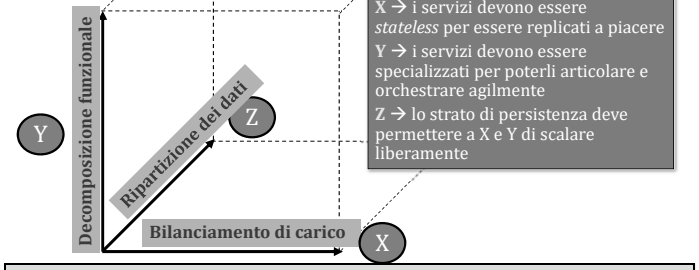
- **Separazione tra politiche e meccanismi**
  - **La politica di servizio deve essere facilmente modificabile, adattabile e configurabile al variare delle necessità**
    - Per questo motivo, la politica di servizio deve essere interna al servente e trasparente al cliente
  - **I meccanismi realizzativi devono essere abbastanza generali da poter supportare diverse politiche, e non dover cambiare al loro variare**

Altre caratteristiche desiderabili – 3

- ***Scalability*: adattabilità alle necessità**
  - **Rispetto alla cardinalità dei componenti del sistema**
    - Poter rimuovere/aggiungere risorse e nodi secondo bisogno, in agilità
  - **Rispetto all'estensione spaziale**
    - Utenti e risorse non risentono della loro distanza geografica
  - **Rispetto alle problematiche locali di gestione**
    - L'amministrazione locale non pregiudica quella globale
- ***Elasticity*: scalabilità senza interruzione di servizio e senza spreco di risorse**

Il cubo della scalabilità

https://www.nginx.com/blog/introduction-to-microservices/



X → i servizi devono essere *stateless* per essere replicati a piacere
Y → i servizi devono essere specializzati per poterli articolare e orchestrare agilmente
Z → lo strato di persistenza deve permettere a X e Y di scalare liberamente

L'opposto della scalabilità

- **Centralizzazione dei servizi**
  - **Singolo servente per tutti gli utenti del sistema**
    - Massimo collo di bottiglia
- **Centralizzazione dei dati**
  - **Tutte le informazioni significative in un unico luogo**
    - Dimensioni e complessità gestionale diventano proibitive
    - Esempi opposti: DNS (ca. 1985), Blockchain (ca. 2008)
- **Centralizzazione degli algoritmi**
  - **Dover conoscere lo stato corrente dell'intero sistema**
    - Insostenibile onere di raccolta e ricostruzione

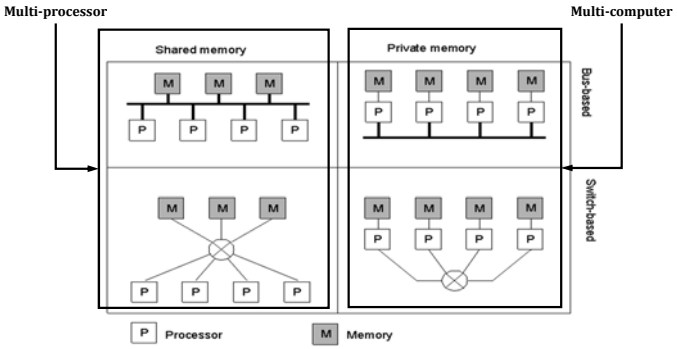
Prerequisiti di distribuzione – 1

- **Un algoritmo è distribuito se**
  - **Ogni sua parte agisce (bene) su base di conoscenza locale**
    - Conoscenza partizionata (DNS), replicata con garanzie (Blockchain)
  - **Non richiede informazione sullo stato globale del sistema**
    - Risposte locali contribuiscono a risposta globale (DNS), risposte locali hanno effetto se convalidate tra pari (Blockchain)
  - **Il dispiegarsi dell'effetto globale non viene pregiudicato da guasti locali**
  - **Non necessita di un tempo di sistema unico e globale**
  - **Consente ripartizione dei compiti e replicazione delle risorse e ne garantisce la consistenza necessaria**

Prerequisiti di distribuzione – 2

- **La comunicazione sincrona ostacola la distribuzione**
  - **Perché blocca le parti (ritardando l'avanzamento) e causa accoppiamento**
- **La comunicazione asincrona abilita la distribuzione**
  - **Perché disaccoppia (nascondendo i ritardi di rete) e favorisce l'avanzamento indipendente**

Distribuzione HW



Architettura di memoria

- **Uniforme (UMA) → *multi-processor***
  - **Spazio di indirizzamento unico e comune**
    - Assunzione di base nell'architettura ***Symmetric Multi-Processor***
  - **Accesso alla memoria uniforme**
    - Ma le richieste di accesso vanno arbitrate (coda e blocco)
  - ***Cache* (in generale) coerente rispetto ai riferimenti condivisi**
- **Non-uniforme (NUMA) → *multi-computer***
  - **Spazio di indirizzamento comune ma non unico**
  - **Accesso alla memoria non uniforme**
    - Costo di accesso ottimizzabile ma pagando in termini di complessità organizzativa
  - **Tenere la *cache* coerente è molto più costoso**

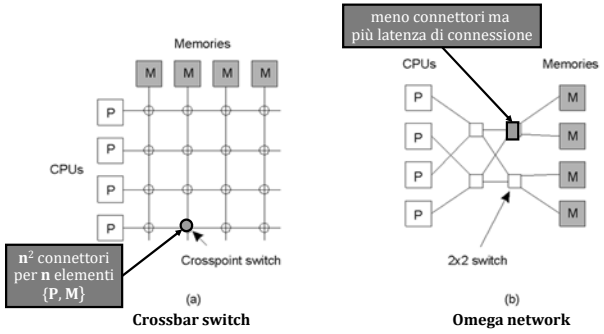
Cache coherence

- **Causa del problema è la *cache* L1 privata di *core***
  - **R/W parallele sulla stessa locazione vedono copie diverse**
- **Possibili rimedi**
  - **Fare senza *cache*: no! Pessimo per le prestazioni**
  - **Condividere *cache* L1 tra *core*: no! Idem**
  - ***Cache write-through*: no! Le R possono non accorgersi delle W**
  - **Ogni R deve vedere l'effetto di ogni W**
    - Ogni W modifica ogni *cache* L1 (***write update***), oppure
    - Ogni W invalida ogni altra copia ovunque (***write invalidate***)
  - **Ogni R deve vedere lo stesso ordine di W**
    - Ogni W è propagata sul *bus* determinandone l'ordine (***snooping***)

Sistemi *multi-processor* – 1

- **Unico spazio di indirizzamento tra le CPU**
  - **La comunicazione P-M su *bus* richiede arbitraggio e diventa collo di bottiglia**
  - **La comunicazione P-M punto-a-punto (*switched*) bilancia meglio il carico, al costo di maggiore complessità realizzativa**
    - Connessione completa (*crossbar switch*) con matrice $P \times M$: maggiore velocità per maggior costo
    - Combinazione di sotto-reti più semplici (p.es. *omega network* 2×2) : minor costo strutturale per maggior complessità di collegamento

Sistemi *multi-processor* – 2



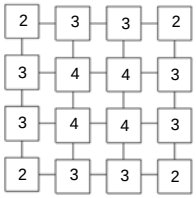
Sistemi *multi-computer* – 1

- **Quelli omogenei**
  - **Possono condividere spazio di indirizzamento**
  - **L'accesso ad esso necessita però attraversamenti di interconnessione P-P via *router* o *switch***
- **Meglio interconnessione punto-a-punto**
  - **Topologia a griglia (*grid*)**
  - **Ipercubo**
    - Cubi n-dimensionali con 2^n vertici (nodi) e $n2^{n-1}$ archi (connessioni)

Sistemi *multi-computer* – 2

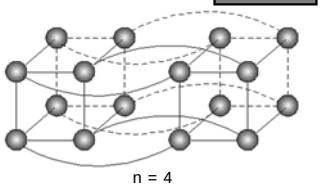
Ogni singolo nodo si occupa di elaborazione e di instradamento

2^n vertici
$n2^{n-1}$ archi



Griglia

La posizione del nodo determina il suo numero di vicini: il *routing* va specializzato



Ipercubo

Il numero di vicini è invariante: il *routing* non va specializzato

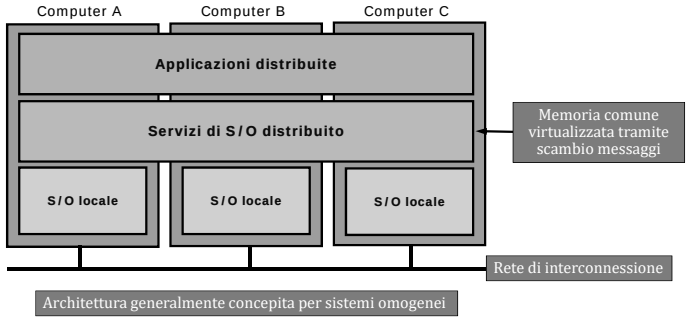
Sistemi *multi-computer* – 3

- **Quelli eterogenei**
  - **Lo sono sia rispetto alla tipologia degli elaboratori che alla topologia di interconnessione**
  - **Sono il modello architetturale più generale**
    - Vero termine di riferimento dei sistemi distribuiti
- **Nota storica**
  - **I sistemi omogenei erano visti come architetture a parallelismo massiccio per applicazioni specializzate**
    - L'avvento dei processori *multi-core* ne ha cambiato la percezione
    - I nuovi processori *many-core* sono *multi-computer* eterogenei organizzati come *cluster* di *multi-core*

Distribuzione SW

- **Secondo la struttura del S/O**
- **Accoppiamento stretto → S/O distribuito**
  - **Gestione uniforme delle risorse di sistema**
    - In analogia con le funzioni di S/O per *mono-processor*
- **Accoppiamento lasco → S/O di rete (NOS)**
  - **Per offrire a utenti remoti l'accesso ad alcune risorse e servizi locali**
  - **Le funzionalità di gestione della distribuzione possono essere arricchite da un livello SW interposto tra NOS e applicazioni → *middleware***

Sistemi operativi distribuiti – 1

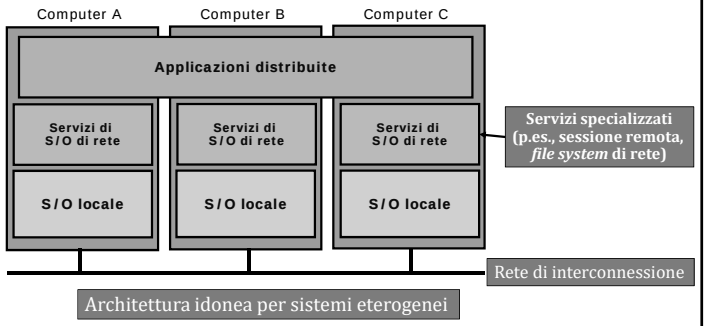


Architettura generalmente concepita per sistemi omogenei

Sistemi operativi distribuiti – 2

- Programmare sistemi distribuiti *multi-computer* è più complesso che per sistemi *multi-processor*
  - Mentre lo *scheduling* diventa più semplice 
- La comunicazione via memoria condivisa e primitive di sincronizzazione è più facile da realizzare di quella via scambio messaggi
  - Lo scambio messaggi scala bene ma viene reso complesso da problematiche di accodamento, sincronizzazione (coordinamento) e affidabilità della rete di interconnessione
  - Per sincronizzare la condivisione di risorse in presenza di parallelismo, non è ovvio scegliere tra *suspend lock* e *spin lock*

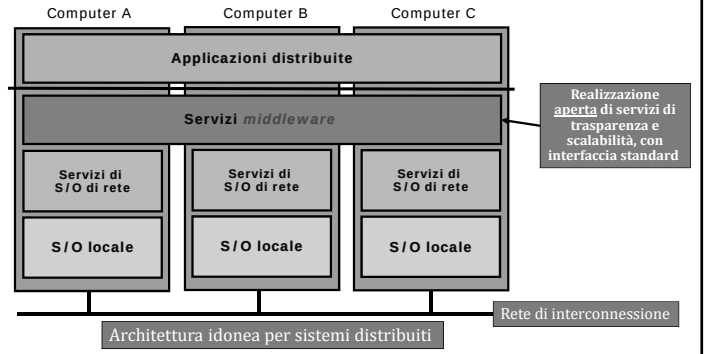
Sistemi operativi di rete



Sistemi distribuiti: *middleware* – 1

- Né i S/O distribuiti né i S/O di rete aderiscono alla definizione di sistema distribuito
  - S/O distribuiti hanno caratteristiche di trasparenza ma non coordinano un insieme di nodi indipendenti
  - S/O di rete hanno caratteristiche di *openness* e *scalability* ma non forniscono la visione di un sistema unitario e coerente
- I sistemi distribuiti moderni aggiungono a (o rimpiazzano) lo strato NOS un livello di astrazione SW chiamato *middleware*

Sistemi distribuiti: *middleware* – 2



Sistemi distribuiti: *middleware* – 3

- Esistono svariati paradigmi di *middleware*
- *File system* distribuito → NFS su UNIX
  - Trasparenza limitata a *file* di tipo tradizionale
- Chiamate di procedura remota (RPC)
  - Trasparenza estesa alla comunicazione distribuita
- Oggetti distribuiti
  - Interazioni come tra oggetti rappresentati da interfacce semplici

Sistemi distribuiti: *middleware* – 4

- Web 1.0
  - Documenti distribuiti → www
- Web 2.0
  - Risorse distribuite → paradigma REST
  - Servizi distribuiti → paradigma SOA
  - Microservizi → paradigma «contenitori orchestrati»
- Problematiche trasversali
  - Trasparenza, *naming*, sicurezza

Sistemi distribuiti: *middleware* – 4

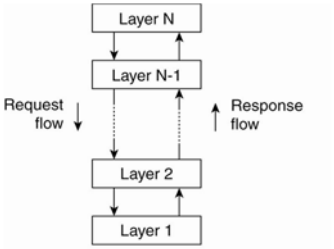
	S/O distribuito		S/O di rete	Sistema distribuito basato su *middleware*
	Multi-processor	Multi-computer		
Grado di trasparenza	Eccellente	Buono	Scarso	Buono
Stesso sistema operativo su ogni nodo	Si	Si	No	No
Istanze di sistema operativo	1	N	N	N
Paradigma di comunicazione	Memoria condivisa	Scambio messaggi	NFS	Svariati
Gestione delle risorse	Centralizzata per risorse globali	Distribuita per risorse globali	Per nodo	Per nodo
Scalability	Nulla	Modesta	Buona	Dipende dal paradigma
Openness	Nulla	Nulla	Buona	Buona

Stili architetturali – 1

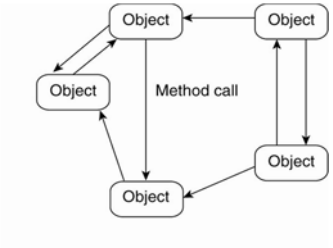
- Espressi in termini di definizione e uso di
  - Componenti per la produzione e il consumo di dati
    - Unità modulare coesa dotata di interfacce fornite e richieste ben definite
  - Connettori per il flusso di dati e l'interazione tra parti
    - Mezzo per comunicazione, coordinamento e cooperazione tra componenti
- Alternative comuni
  - A livelli
  - A oggetti
  - Orientate ai dati
  - Basate su eventi

*An **architectural style** is a named collection of architectural design decisions that*
- *are **applicable** in a given development context*
- ***constrain** architectural design decisions that are specific to a particular system within that context*
- *elicit **beneficial qualities** in each resulting system*

Stili architetturali – 2



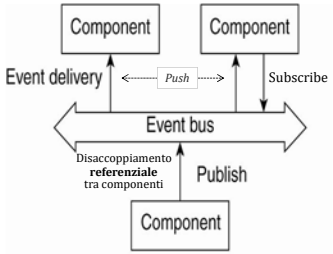
Architettura a livelli



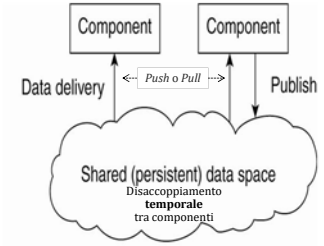
Architettura a oggetti

Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

Stili architetturali – 3



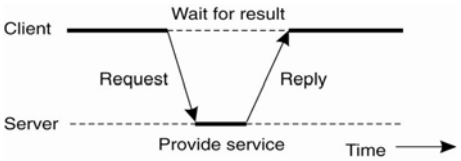
Architettura basata su eventi



Architettura orientata ai dati

Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

Architetture centralizzate



- **L'interazione tra cliente e servente implica un comportamento "*request-reply*"**
  - **Sorgente del problema prestazionale in Web 1.0**
  - **Alcune richieste (ma non tutte!) sono idempotenti**
    - Se lo sono, possono essere ripetute più volte senza causare danni o problemi
    - Proprietà molto importante a fronte di comunicazioni inaffidabili
    - Rendere affidabile una interconnessione fisica inaffidabile ha costo molto elevato

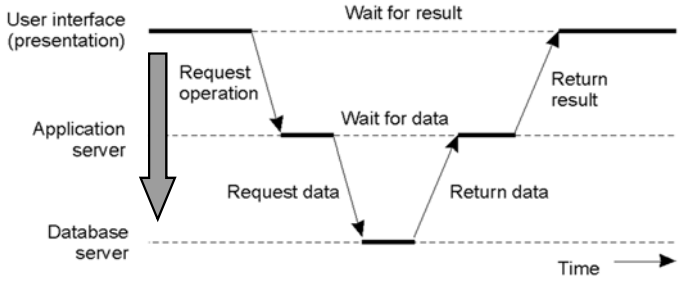
Architetture distribuite – 1

- **Due le varianti principali di architetture distribute di tipo cliente-servente**
  - **Distinte in funzione della loro organizzazione del servizio e dei dati**
- **Distribuzione verticale**
  - **Ripartizione di autorità**
- **Distribuzione orizzontale**
  - **Ripartizione del carico di lavoro**

Architetture distribuite – 2

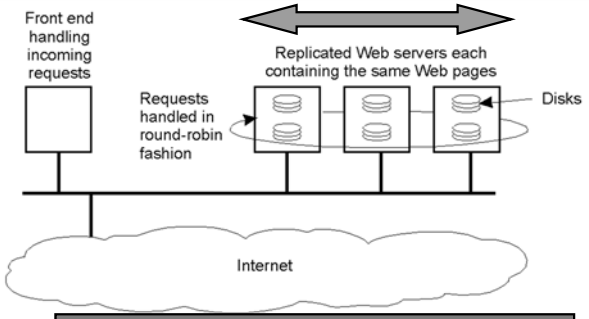
- **Distribuzione verticale → specializzazione**
  - **Componenti diversi dello stesso macro-servizio possono essere assegnati a nodi distinti**
    - Sia sul lato servente che sul lato cliente (delegazione parziale)
  - **Il servizio richiede cooperazione articolata di componenti distribuiti**
- **Distribuzione orizzontale → clonazione**
  - **Servente e cliente possono essere partizionati ma ogni loro componente può operare da solo**
  - **Ogni componente sa fornire "il" servizio richiesto**

Architetture distribuite – 2



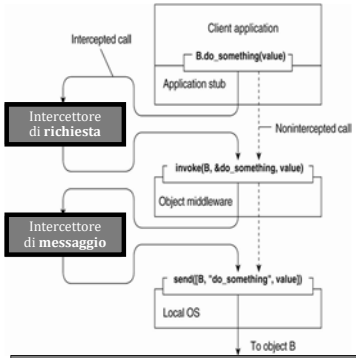
Nell'architettura a **distribuzione verticale** il servente visto dal cliente può essere esso stesso cliente di un componente servente cui sia stata demandata parte del servizio

Architetture distribuite – 4



Nell'architettura a **distribuzione orizzontale** la parte più onerosa del servizio può essere completamente replicata su più elaboratori distinti operanti in parallelo

Middleware moderno



Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

- **Un approccio architetturale al *middleware* offre**
  - **Semplicità progettuale**
  - **Modesta adattabilità**
- **Un approccio più flessibile si basa su**
  - **"*Separation of concerns*"**
  - **"*Computational reflection*"**
  - **Progettazione per componenti e connettori**
    - Gli intercettori in figura mostrano il posizionamento logico dei connettori