Introduzione: Sistemi Distribuiti

Walter Cazzola

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano

Outline

Sistemi Distribuiti: Nozioni Generali
Definizione.

Sviluppo delle reti di computer:

- Reti locali ad alta velocità;
- Reti geografiche che collegano milioni di computer.

Possibilità di utilizzare sistemi di calcolo basati su più computer connessi per mezzo di una rete.

Definizione.
Un *sistema distribuito* è una collezione di computer indipendenti che appare all'utente come un singolo sistema.

Sistemi Distribuiti: Nozioni Generali
Sistemi Centralizzati vs Sistemi Distribuiti.

Sistemi Centralizzati

- le applicazioni girano in un singolo processo, o comunque su un solo host, che costituisce l'unico componente autonomo nel sistema;
- tale componente è condiviso da vari utenti;
- tutte le risorse del componente sono sempre accessibili;
- il componente costituisce il *single point of control* ed il *single point of failure* del sistema.

Sistemi Distribuiti

- le applicazioni sono costituite da più processi, cooperanti, eseguiti in parallelo su un insieme di unità di elaborazione (CPU) autonome;
- una applicazione viene detta *concorrente* se non viene rispettato il requisito sull'autonomia delle CPU.

Sistemi Distribuiti: Nozioni Generali
Distribuzione: Vantaggi e Svantaggi.

Vantaggi

- i sistemi con grandi capacità di calcolo sono molto costosi, mentre il costo delle CPU e dell'infrastruttura di comunicazione si è ridotto notevolmente;
- è più facile gestire la crescita del sistema (nodi o numero di utenti) in modo incrementale;
- è possibile ripartire il carico di lavoro sulle varie macchine in maniera più efficiente;
- è possibile condividere dati e risorse computazionali;
- alcune applicazioni sono inerentemente distribuite, ad es. i servizi di prenotazione aerea o alberghiera;
- maggiore tolleranza ai guasti.

Sistemi Distribuiti: Nozioni Generali
Distribuzione: Vantaggi e Svantaggi (Segue).

"Un sistema distribuito è un sistema in cui il mio programma non funziona per colpa di una macchina di cui non ho mai sentito parlare." **Leslie Lamport.**

Svantaggi

- la differenza tra l'invocazione locale e remota di un servizio è di 4-5 ordini di grandezza (*latenza*);
- fallimenti parziali e concorrenza:
  - una computazione locale può essere concorrente; una distribuita lo è sicuramente;
  - una computazione locale fallisce soltanto in maniera completa, ma una distribuita può fallire anche parzialmente;
  - in una computazione locale, esiste un solo punto di controllo per l'allocazione delle risorse, la sincronizzazione, la gestione dei malfunzionamenti: ciò non è più vero in un sistema distribuito;
- cambia il modo con cui si accede alle risorse, es. i puntatori;
- sicurezza.

Sistemi Distribuiti: Nozioni Generali
Proprietà Desiderabili.

La distribuzione deve agevolare e non ostacolare lo sviluppo e l'esecuzione dell'applicazione.

Trasparenza nell'Accesso.
Non si distingue tra accessi locali e remoti alle risorse:

- codifica little-endian o big-endian dei dati;
- modalità e convenzioni diverse nell'uso dei nomi dei file.

Indipendenza dalla Locazione.
Dati e servizi sono acceduti ignorando la loro locazione fisica:

- si usano nomi che non dipendono dalla locazione fisica (es. URL);
- richiede un meccanismo di localizzazione delle risorse (naming).

Riconfigurazione Dinamica e Trasparente.

- le risorse devono poter essere rilocate senza doverne modificare le modalità di accesso;
- le risorse si devono poter rilocare anche se in uso (utile nei sistemi mobili).

Sistemi Distribuiti: Nozioni Generali
Proprietà Desiderabili (Segue).

Maggiore Disponibilità e Affidabilità

- le risorse possono essere replicate, migliorandone sia la velocità di accesso (es. cache) che la disponibilità (es. load balancing);
- tramite replicazione è possibile anche migliorare l'affidabilità del sistema mascherando dei fallimenti.

Trasparenza nella Concorrenza
Le risorse vengono condivise senza evidenti interferenze:

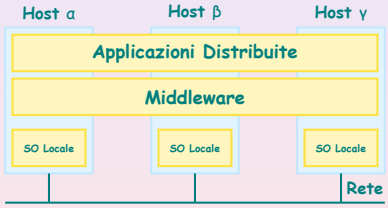
- meccanismi per sincronizzare gli accessi alle risorse;
- meccanismi per garantire la consistenza di una risorsa.

Scalabilità
Le componenti coinvolte dal sistema possono aumentare o diminuire senza che la struttura del sistema o gli algoritmi applicativi debbano cambiare.

Trasparenza

Middleware
Definizione e Caratteristiche.

I sistemi distribuiti sono organizzati con uno strato di software (middleware) che gestisce in modo trasparente un insieme eterogeneo di computer e reti.



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

Middleware
Definizione e Caratteristiche (Segue).

Definizione.

Strato SW che risiede tra l'applicazione e il SO che permette di astrarre ed arricchire le funzionalità del SO e renderle fruibili dall'applicazione.

Il disaccoppiamento tra i livelli di sistema consente la semplificazione del progetto sia della parte applicativa sia del supporto.

Il middleware distribuito permette:

- un uso più efficiente delle risorse;
- la condivisione controllata delle risorse;
- il supporto per ambienti collaborativi e lo scambio di informazioni.

Contro: la necessità di considerare la sicurezza del sistema, coordinare l'accesso alle risorse e realizzare il deployment del sistema.

Middleware
Distributed Middleware

I distributed middleware astraggono i meccanismi di distribuzione e propongono un'interfaccia uniforme alle applicazioni.

Il middleware si inserisce tra il sistema operativo e l'applicazione, ed è a suo volta stratificato:

1. Host Infrastructure Middleware (più vicino al sistema operativo).
2. Distribution Middleware.
3. Common Middleware Services.
4. Domain-Specific Middleware Service (più vicino all'applicazione).

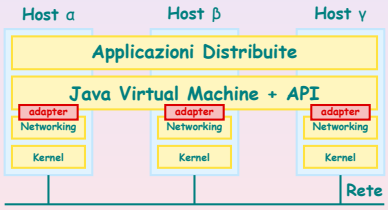
Ogni strato fornisce agli strati superiori servizi differenti.

Noi ci dedicheremo in particolare ai primi due strati.

Sistemi Distribuiti: Nozioni Generali
Distributed Middleware: Host Infrastructure Middleware.

Strato che incapsula e prepara i servizi locali per la distribuzione e per facilitare la comunicazione.

Esempi: JVM, .NET, altri modelli locali.



Si forniscono delle API che permettono di avere un supporto locale unificato per i diversi sistemi.

Middleware
Distributed Middleware: Distribution Middleware.

Livello che fornisce i modelli della programmazione distribuita e facilita le applicazioni distribuite configurando e gestendo le risorse distribuite.

Esempi: RMI, CORBA, DCOM, SOAP, etc ...

Sistemi che permettono una più facile comunicazione e coordinamento dei diversi nodi che partecipano al sistema introducendo:

- un modello delle risorse,
- API di comunicazione secondo un modello concettuale, e
- altre funzionalità accessorie per la comunicazione, per il supporto dei nomi, per il discovery, etc. ...

Middleware
Distributed Middleware: Common Middleware Services.

Servizi aggiuntivi di più alto livello per facilitare la vita del progettista applicativo supportando una programmazione orientata ai componenti.

Esempi: CORBAservices, J2EE, .NET Web Services.

Alcune funzioni addizionali per consentire operazioni facilitate spesso ispirate ad una architettura comune ed ad un modello di supporto.

Molti servizi addizionali per componenti:

- eventi, logging, streaming, sicurezza, fault tolerance, etc. ...

Middleware
Distributed Middleware: Domain-Specific Middleware Service.

Insiemi di livelli applicativi dedicati a domini diversi.

Diversi gruppi (OMG, HL7, Siemens, etc. ...) sono al lavoro per definire e standardizzare funzionalità ad-hoc per diversi settori:

- commercio elettronico;
- finance (home-banking, insurance, brokering, etc. ...);
- telemedicina;
- etc. ...

Middleware
Tipologie di (Distribution) Middleware.

La varie tipologie di distribution middleware dipendono dal modello adottato per la distribuzione.

- RPC middleware;
- message oriented middleware (MOM);
- database Middleware;
- distributed object computing (DOC) o object-based middleware;
- adaptive e reflective middleware.

Altre tipologie, special purpose:

- mobile e QoS multimedia middleware;
- agent-based middleware.

Middleware
RPC Middleware.

RPC come strumento per il Client/Server

- Interface Definition Language (IDL) per uniformare l'attivazione con la definizione del servizio;
- il client è bloccato in attesa della risposta dal server;
- gestione eterogeneità dei dati;
- uso di stub (wrapper) per ottenere la trasparenza nell'accesso;
- il binding è spesso statico o poco dinamico.

Modello un po' rigido, poco scalabile e replicabile con QoS.

Il server deve essere presente e prevedere i processi necessari in modo esplicito.
Non si tiene conto di eventuali ottimizzazioni nell'uso delle risorse.

Esempi: Java RMI, .NET Remoting, DCE.

Middleware
Message Oriented Middleware (MOM).

La distribuzione dei dati e del codice avviene attraverso lo scambio di messaggi.

Scambio messaggi tipati e non tipati sia sincrono sia asincrono con strumenti ad-hoc:

- ampia autonomia tra i componenti;
- asincronicità e persistenza delle azioni;
- gestori (broker) con politiche diverse e QoS diverso;
- facilità nel multicast, broadcast, publish/subscribe.

Esempi: PVM, MPI, CodA, GARF, etc. ...

Middleware
Distributed Object Computing (DOC) o Object-Based Middleware.

La distribuzione dei dati e del codice avviene attraverso richieste di operazioni tra clienti e servitori remoti.

Uso di oggetti come framework e di un broker come intermediario nella gestione delle operazioni:

- il modello ad oggetti semplifica il progetto;
- il broker fornisce i servizi base e molti servizi addizionali;
- alcune operazioni si possono fare in modo automatico;
- l'integrazione di sistemi è facilitata e incentivata.

Esempi: CORBA, .NET e COM, Java

Middleware
Adaptive e Reflective Middleware.

In alcuni casi la visibilità dei livelli sottostanti può diventare fondamentale per migliorare le prestazioni del sistema.

Uso di middleware che si possono adattare alla applicazione specifica anche in modo dinamico, reattivo e radicale:

- variazioni statiche dipendenti dal componente;
- variazioni dinamiche dipendenti dal sistema.

Con la Riflessività, le politiche di azione sono espresse e visibili nel middleware stesso e si possono cambiare come normali componenti del sistema.

Si ottiene adattamento e flessibilità durante l'esecuzione.

Esempi: mChaRM, dynamicTAO, jBoss e molti altri (in generale poco diffusi).

Middleware
Scalabilità.

Definizione.

Scalabilità indica la capacità di un sistema (distribuito) di migliorare le proprie prestazioni all'aumentare delle risorse che lo compongono.

Un sistema le cui prestazioni migliorano quando nuove risorse sono aggiunte è detto sistema scalabile.

La scalabilità può coinvolgere:

- la quantità di risorse coinvolte;
- la distanza geografica tra le risorse stesse;
- l'amministrazione delle risorse coinvolte.

La scalabilità è una proprietà difficile da ottenere.

Middleware
Scalabilità: Difficoltà.

Difficoltà per scalare rispetto alle risorse coinvolte:

- un server centralizzato è un collo di bottiglia per il sistema;
  - spesso la soluzione centralizzata per alcuni servizi è una scelta obbligata (ad es. la gestione dei conti correnti);
  - in altri casi (es. DNS) l'approccio distribuito facilita la realizzazione;
- sistemi completamente distribuiti (es. reti peer-to-peer) hanno problemi analoghi legati al traffico generato fra i singoli nodi.

Difficoltà per scalare rispetto alla distanza geografica:

- la comunicazione sincrona è scoraggiata dai tempi di latenza che su una WAN possono essere di tre ordini di grandezza superiori rispetto ad una LAN;
- la comunicazione su una WAN è meno affidabile e punto a punto (non sono disponibili meccanismi di broadcast o multicast).

Infine, la scalabilità può portare ad annettere sistemi amministrati con criteri di sicurezza differenti.

Middleware
Scalabilità: Tecniche per Migliorarla.

Nascondere la latenza di comunicazione.

- Evitare di attendere la risposta dal server (comunicazione asincrona); il tempo di attesa è usato per altre computazioni, il client è avvertito quando la risposta arriva.
- Spostare parte della computazione dal server al client (es. controllo dei dati immessi).

Distribuzione del Servizio.

- Comporta lo scomporre una componente in parti più piccole che vengono distribuite nel sistema, es. il DNS.

Replicazione.

Le prestazioni possono aumentare replicando le componenti:

- una copia vicina riduce i tempi di latenza (es. copia in cache locale);
- le richieste vengono indirizzate alle repliche che garantiscono un tempo di risposta migliore;
- problema di consistenza tra le copie.

Modelli di Cooperazione per Sistemi Distribuiti.
Outline.

I principali modelli di cooperazione su cui si basano i sistemi distribuiti sono:

- Client-server
- Cooperative processing
- Peer-to-peer
- Messaging model
- Distributed database
- Group communication

Modelli di Cooperazione per Sistemi Distribuiti.

Modello Client-Server: Definizione e Caratteristiche.

Definizione.

Client/Server è un'architettura computazionale che coinvolge dei processi, detti client, che richiedono dei servizi ad altri processi detti server.

Interazione asimmetrica tra due processi

- le computazioni client/server sono la naturale evoluzione della programmazione modulare;
- il processo client invia una richiesta di servizio al server che esegue il servizio e restituisce un risultato;
- la logica di comunicazione è half-duplex: dal client al server e ritorno;
- non è escluso che client e server possano scambiarsi i ruoli;

Variante: function shipping i processi si scambiano anche codice

Modelli di Cooperazione per Sistemi Distribuiti.

Modello Client-Server: Definizione e Caratteristiche (Segue).

Client: Definizione.

- Il client è un'entità (processo, oggetto, ...) che chiede (invia un messaggio, invoca un metodo, ...) ad un server un servizio (eseguire un metodo, del codice ...).

Normalmente il client rappresenta il front-end dell'applicazione che l'utente vede e con cui interagisce.

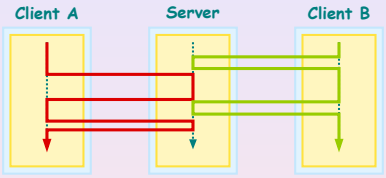
Server: Definizione.

- Il server è un'entità (processo, oggetto, ...) che adempie alle richieste dei clienti fornendo dei servizi.

Il server può essere locale al client o essere eseguito su una macchina remota.

Modelli di Cooperazione per Sistemi Distribuiti.

Modello Client-Server Single-Thread.

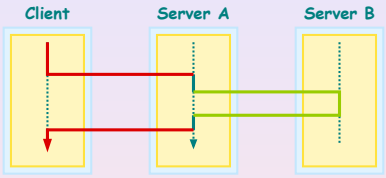


Modello Client-Server Single-Thread

- è la situazione più semplice che si possa avere;
- il server offre uno o più servizi, ma le richieste sono servite una alla volta;
- il server non deve preoccuparsi di garantire la mutua esclusione sulle proprie risorse;
- per poter soddisfare più richieste contemporaneamente occorre eseguire più istanze dello stesso server.

Modelli di Cooperazione per Sistemi Distribuiti.

Modello Client-Server Multi-Tiered.

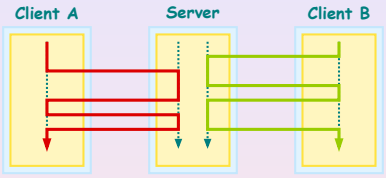


Modello Client-Server Multi-Tiered

- i servizi offerti dal server si basano su servizi offerti da altri server;
- i server sono più semplici da costruire e duttili da usare;
- il soddisfacimento di una richiesta è meno performante a causa del maggior overhead dovuto alle comunicazioni.

Modelli di Cooperazione per Sistemi Distribuiti.

Modello Client-Server Multi-Thread.



Modello Client-Server Multi-Thread

- il server è in grado di gestire, più richieste concorrentemente;
- va garantito un accesso mutuamente esclusivo alle risorse condivise;
- l'uso di un server multi-thread è più efficente dell'usare tanti server single thread.

Modelli di Cooperazione per Sistemi Distribuiti.

Cooperative Processing e Master/Slave.

Cooperative Processing: Definizione.

Cooperative processing (o computazione cooperativa) è una computazione che richiede l'esecuzione di due o più processi distinti.

Il cooperative processing può essere considerato una variante del modello client/server in cui:

- due o più processi sono richiesti per completare una computazione;
- la comunicazione tra processi è ottenuta tramite scambio di messaggi.

Master/Slave: Definizione.

Master/Slave è un modello di cooperazione in cui un'entità (processo, oggetto, ...), detta master, controlla una o più entità, dette slave.

Modelli di Cooperazione per Sistemi Distribuiti.

Peer to Peer (P2P).

Definizione

- è un modello di comunicazione in cui ogni entità ha le stesse capacita delle altre entità coinvolte nella sessione.

Il modello P2P è confrontabile con il modello Client/Server e Master/Slave. Spesso è implementato tramite il Client/Server dotando sia il client che il server delle stesse capacità.

Nell'uso corrente, il termine P2P è usato per descrivere applicazioni che condividono file su internet direttamente o tramite un server intermedio.

Riferimenti Bibliografici

- Anthony J. G. Hey.
  Experiments in MIMD Parallelism.
  In Eddy Odijk, Martin Rem, and Jean-Claude Syre, editors, *Proceedings of Parallel Architectures and Languages Europe (PARLE'89)*, LNCS 366, pages 28–42, Eindhoven, The Netherlands, June 1989. Springer-Verlag.
- Andrew S. Tanenbaum.
  Distributed Operating Systems.
  Prantice-Hall, 1995.

Trasparenza nell'Accesso.

Eterogeneità: Gioie e Dolori.

Gli host di una rete eterogenea possono avere architettura e/o sistema operativo differenti.

Accedere a risorse remote vuol dire dover uniformare l'eterogeneità degli host:

- tutti i dati devono avere una rappresentazione omogenea, ad es. un numero in virgola mobile (float) deve essere rappresentato con lo stesso numero di bit;
- la rappresentazione dei dati deve essere indipendente dalla rappresentazione usata dall'architettura sottostante, ad es. little- e big-endian;
- il reperimento delle risorse locali deve essere indipendente da come le gestisce il sistema operativo, ad es. le directory nel path potranno essere separate da / o \ indistintamente;
- spazi di indirizzamento distinti, ad es. un puntatore NON riferisce alla stessa area di memoria.

Indipendenza dalla Locazione.

Reperire Dati e Servizi.

Per accedere a dati e servizi remoti, è necessario sapere dove sono memorizzati. Abbiamo diverse tecniche:

Indirizzamento à la Internet.

id. rete + id. host + id. locale (arbitrario)

Problemi:

- mancanza di trasparenza;
- non fault tolerant (se l'host fallisce, si potrebbe voler attivare il servizio su di un altro host: problema della migrazione).

Uso di hardware specializzato.

- Nei chip di interfaccia con la rete fisica si possono memorizzare gli identificatori dei processi operanti su quell'host.
- I frame usano direttamente l'identificatore del processo, che viene riconosciuto più velocemente.

Indipendenza dalla Locazione.

Reperire Dati e Servizi (Segue).

Identificatore unico su tutta la rete, indipendente dall'host.

Problema:

- serve una componente centralizzata che distribuisca gli identificatori, che devono essere unici sulla rete (BOTTLENECK).

Si può consentire agli host di scegliersi (in un ampio rango) l'identificatore.

Però, come si fa a sapere su quale macchina si trova veramente la risorsa? Soluzioni possibili:

- Si manda a tutti gli host in broadcast un pacchetto di localizzazione per un certo indirizzo simbolico:
  - ricevuto il pacchetto con il proprio indirizzo simbolico, il server risponde con il proprio indirizzo fisico;
  - l'indirizzo è memorizzato dal kernel del cliente per evitare il broadcast la volta successiva in cui ce n'è necessità (soluzione distribuita).
- Si usa un name server che opera su una macchina particolare e nota, che risponde alle richieste di localizzazione inviate dai client (soluzione centralizzata).