Indice

1 Introduzione

1.1 Definizione

> Un sistema distribuito è un insieme di computer indipendenti che creano l'illusione per l'utente di un singolo sistema

Nota bene che non vengono fatte presupposizioni sul tipo di computer, il sistema potrebbe essere un mainframe come un insieme di nodi di una rete sensoriale.
L'illusione del singolo sistema viene creata grazie a un livello tra le applicazioni e i sistemi operativi. Questo livello viene spesso chiamata *middleware*.

1.2 Obiettivi

Solo perché è possibile creare un sistema distribuito non significa che sia necessario. Ci sono 4 obiettivi importanti da considerare quando si decide se costruire un sistema distribuito.

1.2.1 Rendere le risorse accessibili

L'obiettivo principale di un sistema distribuito è quello di rendere le risorse facilmente accessibili al utente e di condividere in maniera semplice ed efficiente. Una risorsa può essere qualsiasi cosa: stampante, computer, file ecc.
Un motivo ovvio per condividere una risorsa è per risparmiare soldi, sia evitando di comprare molteplice macchine, sia evitando costi di manutenzione extra.
Inoltre connettere utente e risorse facilita la collaborazione.
Un problema della condivisione dei dati è la privacy, non è possibile dare per scontato che la comunicazione non venga tracciata.

1.2.2 Distribution Transparency

Un importante obiettivo dei sistemi distribuiti è quello di nascondere il fatto che il processo e le risorse vengono condivise tra molteplici computer.
Ci sono vari tipi di *transparency*:

- *accesso*: nascondi come vengono salvate e condivise le risorse. In termini pratici si sceglie un rappresentazione standard attraverso vari macchine e sistemi operativi.
- *posizione*: nascondi la posizione della risorsa. È semplice da fare, è sufficiente avere un nome che non rappresenta la sua posizione, come un link URL.
- *migrazione*: nascondi il fatto che una risorsa possa essere trasferita in un altra posizione. Non deve in nessuno modo influenzare l'accesso a questa risorsa
- *relocation*: nascondi il fatto che una risorsa possa essere spostata **mentre** viene usata
- *replicazione*: nascondi il fatto che una risorsa è duplicata. È necessario per rendere la risorsa più facilmente accessibile, per esempio attraverso regioni geografiche.
- *concorrenza*: nascondi che la risorsa possa essere condivisa tra più di un utente in maniera competitiva (data race). Un metodo semplice per evitare data race è usare lock.
- *fallimento*: nascondi il fallimento e il recupero di una risorsa

È necessario considerare bene il grado di transparency che si vuole ottenere, andando ad aumentare la transparency si rischia di perdere performance e comprensibilità.

1.2.3 Opennes

Un sistema "open" è un servizio che viene distribuito in base a una serie di regole standard di sintassi e semantica. Tale regole devono essere definite in maniera univoca e neutra, in modo da poter creare due implementazione diverse a partire da un unica interfaccia che abbiano lo stesso servizio offerto. Questo garantisce la portabilità del sistema e permette di estendere o modificare senza compromettere il sistema in altri punti.
Tale livello di "opennes" è augurabile ma difficilmente raggiungibile.

1.3 Scalabilità

La scalabilità è uno degli obiettivi più importanti dei sistemi distribuiti. È possibile misurare la scalabilità di un sistema lungo 3 dimensioni.
In primis tramite la sua grandezza, ovvero quanto è facile aggiungere utenti e risorse. In seconda battuta la sua scalabilità geografica, quanto scala bene se si hanno utenti a grossi distanze. Infine deve essere scalabile per l'amministrazione, ovvero deve essere facile da maneggiare anche attraverso varie amministrazioni.
Sfortunatamente un sistema scalabile in una o più di queste dimensioni finisce per perdere performance.

1.3.1 Problemi della scalabilità

Scalando un sistema vengono fuori problematiche diverse. Un importante problema è la limitazione di avere un servizio, dato o algoritmo centralizzato.

Concetto	**Esempio**
Servizio centralizzato	Un singolo server
Dati centralizzati	Una singola linea telefonica
Algoritmo centralizzato	Un singolo punto di elaborazione

Un server unico(servizio centralizzato), anche avendo possibilità di scalare verticalmente all'infinito, finirà comunque per fare da *bottleneck*.
Similarmente un singolo database che mantiene 50 milioni di utenti richiederà un file di 2.5 gigabyte creando un enorme rallentamento per il nostro servizio.
Infine un algoritmo centralizzato richiede che ogni individuale macchina inoltri i dati da processare a un unica grossa macchina. Naturalmente si ripete

lo stesso problema di congestione come visto nei precedenti casi. Si aggiunge però un ulteriore problema, ovvero è necessario che le macchine abbiano una qualche conoscenza dello stato delle altre macchine e che facciano decisione che si basi su queste informazioni. Un simile approccio non è permesso in un sistema distribuito.
La scalabilità geografica richiede un implementazione asincrona dei servizi, siccome il tempo di risposta può essere molto lungo tra regioni e una trasmissione wireless è molto meno sicura rispetto a una rete VLAN.
Infine è un grosso problema fare una scalabilità di amministrazione, perché ogni dominio ha una serie di policy da rispettare.

1.3.2 Tecniche di scalabilità

Ci sono 3 tecniche principali di scalabilità: nascondere la latenza, distribuzione e replicazione.
Un modo per nascondere la latenza è permetter al utente di usare altri servizi mentre aspetta la risposta dal server, ovvero il processo avviene in maniera asincrona.
Non è sempre possibile effettuare operazioni asincrone, quindi in alternativa si può spostare la logica in altri punti. Nel caso in cui un utente debba compilare un *form* invece che richiedere al server la correttezza dei dati inseriti, questo logica può essere effettuata dal client.
La distribuzione prende un componente grande e lo divide in componenti più piccoli sparsi nel sistema. Un esempio è il DNS, che dividei domini in zone. Il tutto è un albero che viene risalito fin tanto che si ha un riscontro e ritorna l'indirizzo IP corretto.
La nota dolente più grandi di un sistema distribuito è la performance quindi è un ottima idea la replicazione dei dati. Questo aumenta la disponibilità, diminuisce il carico di lavoro e diminuisce la latenza. Il *caching* è un caso speciale di replicazione effettuata da client, che tiene una copia temporanea del dato per motivi di latenza. Invece la replicazione viene pianificata ed effettuata in anticipo.
Il *caching* e la replicazione creano a loro volta un problema: la *consistency*. In molti casi nel mondo del web questo non è un problema, in altri particolari, come il trading, è imperativo garantire la *consistency*. Questo problema è difficile da risolvere perché comunque c'è un limite fisico di trasmissione dei dati.

1.4 Problematiche

Quando si costruisce un sistema distribuito ci sono una serie di false assunzioni:

1. La rete è affidabile
2. La rete è sicura
3. La rete è omogenea
4. La topologia(del sistema?) non cambia
5. La latenza è 0
6. La lunghezza banda è infinita
7. Costo di trasporto è 0
8. C'è solo un amministrazione

Queste assunzioni creano una serie di problematiche da risolvere quando si sviluppa un sistema distribuito.

gotem