



Sistemi distribuiti: processi e concorrenza


Processi e concorrenza in distribuito



Anno accademico 2017/18
Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@math.unipd.it

Laurea Magistrale in Informatica, Università di Padova 1/28



Sistemi distribuiti: processi e concorrenza

Considerazioni di costo – 2

- ❑ Il *context switch* a livello di *thread* può non aver bisogno dell'intervento del S/O
 - Perché si risolve all'interno dello stesso spazio di memoria virtuale e i *thread* possono non essere noti al S/O
- ❑ Il *context switch* a livello di *process* ha costo alto
 - Perché coinvolge il S/O
- ❑ Creare e distruggere processi costa molto
- ❑ Farlo con i *thread* costa meno ma non poco

Laurea Magistrale in Informatica, Università di Padova 3/28



Sistemi distribuiti: processi e concorrenza

Considerazioni di costo – 1

- ❑ Contesto di *processor*
 - I registri (e i loro significati d'uso)
- ❑ Contesto di *thread*
 - Il contesto del *processor* e la memoria che contiene lo stato del *thread* (*stack, heap, ...*)
- ❑ Contesto di *process*
 - Il contesto dei *thread* e lo stato della memoria virtuale assegnata al processo
 - I *thread* di un processo condividono lo spazio di indirizzamento assegnato al processo

Laurea Magistrale in Informatica, Università di Padova 2/28



Sistemi distribuiti: processi e concorrenza

Considerazioni di costo – 3

- ❑ Non è evidente che convenga supportare *thread* a livello di S/O (*kernel*)
 - In quel caso ogni operazione a livello *thread* (gestione di I/O bloccante e di eventi esterni) coinvolge il S/O
- ❑ Soluzioni nello spazio utente sono possibili
 - Ma gli interventi del S/O hanno effetto su tutti i *thread* del processo, riducendone il parallelismo
- ❑ Serve un approccio più intelligente
 - LWP (*light-weight process*) nato in Solaris e acquisito in Linux

Laurea Magistrale in Informatica, Università di Padova 4/28

Sistemi distribuiti: processi e concorrenza

Considerazioni di costo – 4

User space

Thread state

Thread

Basso costo di creazione (non coinvolge il S/O)

Kernel space

Lightweight process

Creati una sola volta (in un *pool*)

LWP executing a thread

Le operazioni di S/O non rompono il legame tra *thread* (in *user space*) e LWP (in *kernel space*)

Laurea Magistrale in Informatica, Università di Padova

5/28

Sistemi distribuiti: processi e concorrenza

Il server in Node.js

Node.js Server

Event Queue

Thread Pool

Database

File system

Network

Others

Requests

Event Loop

Operation Complete

Source: <http://abdelraoof.com/blog/2015/10/28/understanding-nodejs-event-loop>

Laurea Magistrale in Informatica, Università di Padova

7/28

Sistemi distribuiti: processi e concorrenza

Considerazioni di costo – 5

- ❑ L'uso di *thread* nell'applicazione richiede un importante sforzo di pensiero concorrente e di sua gestione
- ❑ La facilità di creazione (*new*) può indurre al consumismo
 - Esempio: Apache crea un *thread* per ogni richiesta HTTP senza curarsi del rapporto costi-benefici rispetto ai dati da essa trasferiti
- ❑ Un approccio più conveniente per applicazioni *IO-bound* è usare eventi
 - Esempio: Node.js: un singolo *thread* per programma
 - Esecuzione a «*event loop*» per ogni azione bloccante pendente
 - Il *thread* serve la coda di *callback* (TODO) del programma fino a esaurimento

Laurea Magistrale in Informatica, Università di Padova

6/28

Sistemi distribuiti: processi e concorrenza

Cliente e servente concorrenti – 1

- ❑ **Multi-threading di lato cliente**
 - La concorrenza interna mitiga l'effetto del ritardo di rete
 - In un *Web browser* (lato cliente) conviene eseguire in parallelo
 - L'attivazione della connessione TCP/IP è operazione bloccante
 - Lettura ed elaborazione dei dati in ingresso sono eseguibili in *pipeline*
 - Il trasferimento su video è eseguibile in *pipeline*
 - Google Chrome (2008) primo *browser multi-threaded* (!)
 - Il cliente può supportare più sessioni parallele
 - P.es., i "tab" di un *browser*?
 - Uno dei presupposti su cui si basa AJAX
 - P.es.: <http://www.cmarshall.net/MySoftware/ajax/Threads/>

Laurea Magistrale in Informatica, Università di Padova

8/28

Sistemi distribuiti: processi e concorrenza
Cliente e servente concorrenti – 2

❑ **Multi-threading di lato servente**

- **La concorrenza interna offre**
 - Maggiore efficienza prestazionale ancor più utile e desiderabile che nel cliente
 - Maggiore modularità (specializzazione, semplicità) architetturale

The diagram shows a central box labeled 'Dispatcher' with an arrow pointing to it from the left. To its right, two boxes labeled 'Worker 1' and 'Worker N' are connected to the Dispatcher by lines. The entire system is enclosed in a larger box labeled 'Servente' at the bottom.

Laurea Magistrale in Informatica, Università di Padova
9/28

Sistemi distribuiti: processi e concorrenza
Problematiche di lato cliente – 1

Trasparenza	Ruolo del MW di lato cliente
Accesso	Fondamentale – a carico di <i>stub</i> (RPC) o <i>proxy</i> (RMI)
Collocazione	Fondamentale – tramite gestione delle corrispondenze nome-indirizzo (<i>naming</i>)
Migrazione / Spostamento	Desiderabile – serve <i>naming</i> a gestione dinamica
Replicazione	Utile per nascondere la possibile interazione con più repliche del servente
Transazione	Utile (ma molto di più dal lato servente)
Malfunzionamento	Desiderabile – p.es. il <i>caching</i> del <i>Web browser</i>
Persistenza	Non significativa (ma fondamentale dal lato servente)

Laurea Magistrale in Informatica, Università di Padova
11/28

Sistemi distribuiti: processi e concorrenza
Corto-circuitazione: TCP hand-off

The diagram illustrates a three-tier architecture. On the left, 'Client requests' enter a 'Logical switch (possibly multiple)'. These requests are 'Dispatched request' to a 'Second tier' of 'Application/compute servers'. These servers connect to a 'Third tier' 'Distributed file/database system'. A text box says 'Per scaricare il dispatcher di 1 livello'. Below, a 'Client' sends a 'Request' to a 'Switch', which then sends a 'Request (handed off)' to a 'Server'. A 'Response' is sent back from the 'Server' to the 'Switch' and then to the 'Client'. A note says 'Logically a single TCP connection'.

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

Laurea Magistrale in Informatica, Università di Padova
10/28

Sistemi distribuiti: processi e concorrenza
Problematiche di lato cliente – 2

The diagram compares two architectures. On the left, 'Architettura Fat-client' shows a 'Client machine' with 'Application', 'Middleware', and 'Local OS' layers, connected via 'Application-specific protocol' to a 'Server machine' with 'Application', 'Middleware', and 'Local OS' layers. On the right, 'Architettura Thin-client' shows a 'Client machine' with 'Appl.' and 'Middleware' layers, connected via 'Application-independent protocol' to a 'Server machine' with 'Appl.', 'Middleware', and 'Local OS' layers. Both are connected to a 'Network'.

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

Laurea Magistrale in Informatica, Università di Padova
12/28

Sistemi distribuiti: processi e concorrenza

Problematiche di lato cliente – 3

- ❑ Un *thin client* sa solo riflettere quello che riceve dal server tramite la rete
 - Non sa cosa fare in assenza di comunicazioni dal *server*
 - L'architettura dell'X-Window System (X11, oggi l'X.org di Linux) era basata su questo paradigma
- ❑ Un *fat client* sa svolgere lavoro in proprio
 - Ha cose da fare anche in assenza di comunicazioni di rete
 - Quindi scarica di oneri il *server*
- ❑ Come classificare le *single-page web app*?

Laurea Magistrale in Informatica, Università di Padova

13/28

Sistemi distribuiti: processi e concorrenza

Distribuzione verticale – 1

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

Laurea Magistrale in Informatica, Università di Padova

15/28

Sistemi distribuiti: processi e concorrenza

Problematiche di lato server: organizzazione

- ❑ **Organizzazione iterativa o ricorsiva → distribuzione verticale** IMPORTANT!
 - Il servente utilizza i servizi di altri serventi (interni o esterni)
 - La richiesta successiva riceverà attenzione solo dopo il completamento di quella corrente
 - Per soddisfare più richieste in parallelo bisogna replicare l'intero servente
- ❑ **Organizzazione concorrente → distribuzione orizzontale**
 - Il *front-end dispatcher* del servente si limita ad accogliere richieste demandandone il soddisfacimento a un *worker thread* distinto
 - Nuove richieste possono essere accolte appena quella corrente sia stata affidata all'esecutore selezionato
 - Per soddisfare più richieste in parallelo basta replicare gli esecutori (1 *dispatcher* – N *worker*) facendo però attenzione alle problematiche «Apache»

Laurea Magistrale in Informatica, Università di Padova

14/28


Sistemi distribuiti: processi e concorrenza

Distribuzione verticale – 2

- ❑ Paradigma di *name resolution* del DNS
- ❑ La richiesta iterativa sposta l'onere chi la emette
- ❑ La richiesta ricorsiva sposta l'onere su chi la riceve

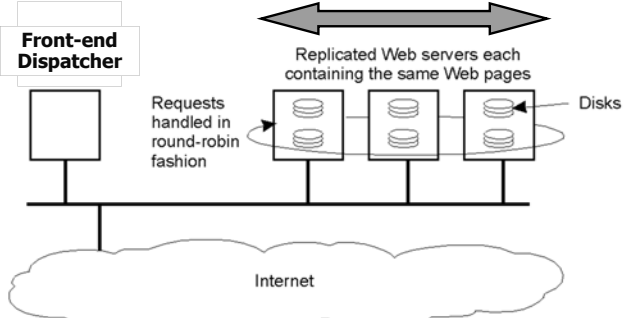
Laurea Magistrale in Informatica, Università di Padova

16/28



Sistemi distribuiti: processi e concorrenza

Distribuzione orizzontale



Front-end Dispatcher

Replicated Web servers each containing the same Web pages


Disks

Requests handled in round-robin fashion

Internet

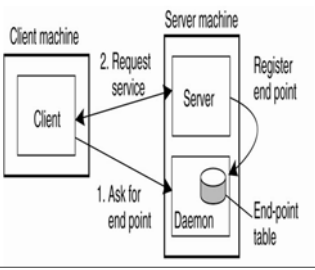
Nell'architettura a **distribuzione orizzontale** la parte più onerosa del servizio può essere completamente replicata su più elaboratori distinti operanti in parallelo

Laurea Magistrale in Informatica, Università di Padova
17/28



Sistemi distribuiti: processi e concorrenza

Localizzazione del servente



Client machine

Server machine

Client

Server

Daemon

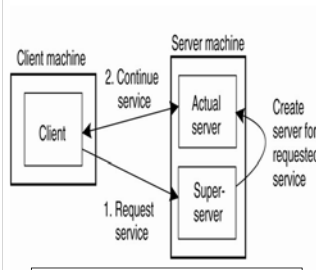
Register end point

End-point table

1. Ask for end point

2. Request service

Assegnazione dinamica di porta servente (interazione tramite *daemon*)



Client machine

Server machine

Client

Actual server

Super-server

Create server for requested service


1. Request service

2. Continue service

Attivazione dinamica di servente (interazione tramite *super-server*)

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

Laurea Magistrale in Informatica, Università di Padova
19/28




Sistemi distribuiti: processi e concorrenza

Problematiche di lato servente: localizzazione

- ❑ Al servente corrisponde una porta (*end-point*) del suo nodo di residenza, su cui ascolta un processo dedicato
- ❑ La porta può essere preassegnata
 - Da IANA (*Internet Assigned Numbers Authority*) per protocolli (serventi) base: p.es., HTTP:80, FTP:20-1, SMTP:25, ...
- ❑ Oppure assegnata dinamicamente
 - Un *daemon* ascolta su porta di convenzione le richieste in arrivo e poi – per ogni servizio gestito – le inoltra a una porta assegnata dinamicamente
 - Per richieste rare, un Super-Servente (in *inetd* di UNIX) ascolta tutte le porte dei corrispondenti serventi e li risveglia (o crea dinamicamente) secondo bisogno

Laurea Magistrale in Informatica, Università di Padova
18/28




Sistemi distribuiti: processi e concorrenza

Problematiche di lato servente: interrompibilità

- ❑ **Modello TCP/IP**
 - La rottura della connessione (p.es., abbandono del cliente) causa interruzione del servizio
 - Non immediata, ma garantita senza confusione con richieste successive
- ❑ **Dati "out-of-band"**
 - Il cliente può chiedere di dare precedenza a dati fuori sequenza ma di maggiore urgenza
 - Designazione di urgenza nell'intestazione del *payload*
 - Cliente e servente devono intrattenere più di una sotto-connessione logica entro la stessa connessione di servizio
 - Con porta distinta per ogni sotto-connessione


Laurea Magistrale in Informatica, Università di Padova
20/28



Sistemi distribuiti: processi e concorrenza


Problematiche di lato servernte: stato – 1

- ❑ **Servernte senza stato (*stateless*)** → lo stato non è nel servizio
 - Non ricorda lo stato di servizio del cliente
 - Non deve informarlo di eventuali cambi di stato di lato servernte
- ❑ **L'esempio classico è NFS**
 - Il cliente opera localmente su *virtual inode*
 - La *cache* di lato cliente è *write-through* (con scrittura asincrona) e non coerente tra clienti diversi
 - Il servernte tratta ogni operazione in sessione distinta
 - Il *file system* di lato servernte può cambiare locazione, stato ed esistenza dei propri *file* senza doverne informare alcun cliente
- ❑ La ***statelessness*** è la base della scalabilità elastica



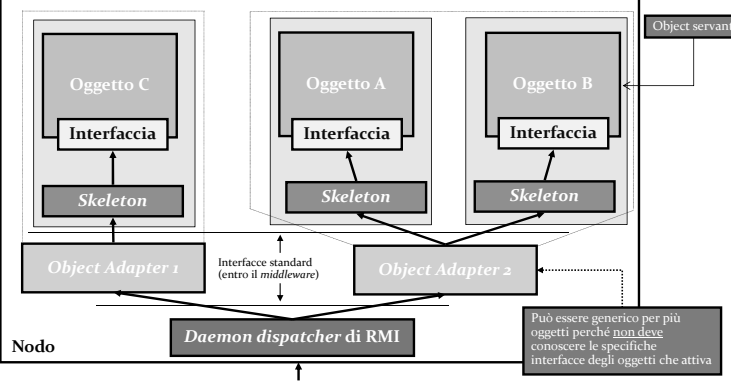
Laurea Magistrale in Informatica, Università di Padova

21/28



Sistemi distribuiti: processi e concorrenza

Servernte di oggetto – 1



The diagram illustrates the Object Server architecture. At the top, three boxes represent 'Oggetto C', 'Oggetto A', and 'Oggetto B'. Each object box contains an 'Interfaccia' (Interface) and a 'Skeleton'. Below these, 'Object Adapter 1' and 'Object Adapter 2' are shown. Arrows indicate that the Skeletons implement the Interfaces, and the Object Adapters implement the Skeletons. The Object Adapters also implement an 'Interfaccia standard (entro il middleware)'. A 'Daemon dispatcher di RMI' is shown at the bottom, which manages the communication between the client and the server. A note on the right states: 'Può essere generico per più oggetti perché non deve conoscere le specifiche interfacce degli oggetti che attiva'.

Laurea Magistrale in Informatica, Università di Padova

23/28



Sistemi distribuiti: processi e concorrenza


Problematiche di lato servernte: stato – 2

- ❑ **Servernte con stato (*stateful*)** → lo stato è nel servizio
 - Ricorda lo stato di servizio del cliente e offre sempre stato di servizio coerente
 - L'esempio classico è nei sistemi transazionali
 - *begin* (Op_1, Op_2, \dots, Op_n) *commit*
 - **Atomicity**: gli effetti sullo stato sono di tipo *all-or-nothing*
 - **Consistency**: lo stato di lato servernte è sempre consistente (risultante da transazioni eseguite in un qualche ordine totale)
 - **Independence (isolation)**: le transazioni non interferenti possono eseguire in parallelo
 - **Durability**: gli effetti delle transazioni terminate con successo sono persistenti
- ❑ **Promesse non scalabili elasticamente al variare dello stato**



Laurea Magistrale in Informatica, Università di Padova

22/28



Sistemi distribuiti: processi e concorrenza

Servernte di oggetto – 2

- ❑ **Ospita la concretizzazione dell'oggetto distribuito**
 - Non fornisce alcun servizio in proprio
 - È il tramite per l'invocazione locale per conto del vero chiamante remoto
- ❑ **La sua implementazione determina la separazione effettiva tra l'interfaccia e lo stato dell'oggetto**
- ❑ **Può supportare diverse politiche di attivazione dell'oggetto distribuito**
 - Modello più potente ed espressivo di RPC

Laurea Magistrale in Informatica, Università di Padova

24/28



Politiche di attivazione – 1

- ❑ Fissano le modalità con cui un oggetto remoto può essere invocato, cioè il suo ciclo di vita
- ❑ Creazione e distruzione dell' *object reference*
 - Ciò che rende disponibile al cliente remote l'entità che realizza le operazioni dell'interfaccia distribuita
- ❑ Attivazione e de-attivazione del *servant*
 - L'insieme di risorse (CPU, memoria) che realizzano l'oggetto nel servente



Compiti dell' *object adapter*

- ❑ Registrare implementazioni di interfacce
- ❑ Mappare, generare, interpretare riferimenti a tali implementazioni
- ❑ Attivare e disattivare *servant* e relativa implementazione
- ❑ Inoltrare invocazioni di metodi ai *servant* corrispondenti
- ❑ Partecipare a garantire la sicurezza delle interazioni con il cliente

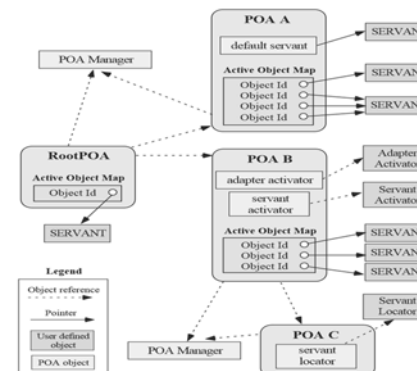


Politiche di attivazione – 2

- ❑ Politiche comuni per nodo sono attuate da un singolo *object adapter*
 - Noto *design pattern* della GoF
 - "A reusable class that cooperates with unrelated or unforeseen classes"
- ❑ Un OA fornisce metodi per
 - Ricevere invocazioni remote in arrivo dal MW e inviarle al *servant* destinatario [ruolo funzionale]
 - Registrare/rimuovere *servant* e abilitare politiche di servizio [ruolo amministrativo]



Portable Object Adapter



Pyarali & Schmidt, An Overview of the CORBA Portable Object Adapter