



Sistemi distribuiti: processi e concorrenza

Processi e concorrenza in distribuito



Anno accademico 2015/16
Sistemi Concorrenti e Distribuiti

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

Laurea Magistrale in Informatica, Università di Padova 1/27



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
- ❑ Il *context switch* a livello di *process* ha costo alto perché la gestione della memoria virtuale coinvolge il S/O
- ❑ Creare e distruggere *thread* costa molto meno che farlo con processi

Laurea Magistrale in Informatica, Università di Padova 3/27



Sistemi distribuiti: processi e concorrenza

Considerazioni di costo – 1

- ❑ Contesto di *processor*
 - I registri
- ❑ 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 stesso spazio di indirizzamento

Laurea Magistrale in Informatica, Università di Padova 2/27



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 «intelligente»: LWP (*light-weight process*) nato in Solaris e ritenuto in Linux

Laurea Magistrale in Informatica, Università di Padova 4/27

Sistemi distribuiti: processi e concorrenza

Considerazioni di costo – 4

User space

Thread state

Thread

Kernel space

Lightweight process

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 5/27

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 7/27

Sistemi distribuiti: processi e concorrenza

Considerazioni di costo – 5

❑ **L'uso di *thread* nell'applicazione richiede un importante sforzo di logica concorrente**

- Oppure un uso massiccio di memoria: Apache crea un *thread* per ogni richiesta HTTP a prescindere dal rapporto costi-benefici rispetto ai dati da essa trasferiti

❑ **Per applicazioni più «pragmatiche» può convenire un approccio a eventi**

- **Node.js: un singolo *thread* per programma**
 - Esecuzione a «*event loop*» per ogni azione bloccante effettuata
 - Il *thread* serve la coda di *callback* del programma fino al suo esaurimento

Laurea Magistrale in Informatica, Università di Padova 6/27

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) architeturale

Dispatcher

Worker 1

Worker N

Servente

Laurea Magistrale in Informatica, Università di Padova 8/27

Sistemi distribuiti: processi e concorrenza

Cliente e servente concorrenti – 3

❑ **La soluzione «node.js» nasconde la concorrenza all'architettura del servizio**

The diagram illustrates the Node.js architecture. It shows a 'Main Event Loop' with steps: Initialize, Run V8, Event Loop, and Exit. A 'Thread n' can 'Create Thread', 'Delegate Task', 'Perform Task', and 'I/O Operation' before exiting. A 'Node.js Application' sends '(function, callback)' to the 'Event Loop', which then delegates tasks (Task 1, Task 2, Task 3, Task 4) to various threads. Callbacks are returned to the Event Loop.

Laurea Magistrale in Informatica, Università di Padova
9/27

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 an '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., Middleware, and Local OS layers, connected via an 'Application-independent protocol' to a Server machine with Appl., Middleware, and Local OS layers. Both architectures use a 'Network' for communication.

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

Laurea Magistrale in Informatica, Università di Padova
11/27

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
10/27

Sistemi distribuiti: processi e concorrenza

Problematiche di lato cliente – 2

❑ **Un *thin client* sa solo riflettere quello che riceve dal *server* tramite la rete**

- Non sa funzionare 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*

Laurea Magistrale in Informatica, Università di Padova
12/27

Sistemi distribuiti: processi e concorrenza

Problematiche di lato servernte – 1

❑ **Due possibili organizzazioni**

- **Iterativa o ricorsiva → distribuzione verticale**
 - Il servernte utilizza i servizi di altri servernti (interni o esterni)
 - La richiesta successiva potrà essere accolta **solo dopo** il completamento di quella corrente
 - Per soddisfare più richieste in parallelo bisogna replicare l'intero servernte
- **Concorrente → distribuzione orizzontale**
 - Il *front-end dispatcher* del servernte si occupa solo di accogliere richieste demandandone il soddisfacimento a un *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*) → con attenzione alle problematiche «Apache»

Laurea Magistrale in Informatica, Università di Padova
13/27

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
15/27

Sistemi distribuiti: processi e concorrenza

Distribuzione verticale – 1

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

Laurea Magistrale in Informatica, Università di Padova
14/27

Sistemi distribuiti: processi e concorrenza

Distribuzione orizzontale

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
16/27



Sistemi distribuiti: processi e concorrenza

Problematiche di lato servernte – 2

❑ **Localizzazione del servernte**

- Al servernte corrisponde una porta (*end-point*) del nodo sulla quale sta in ascolto un processo dedicato
- Porta preassegnata
 - IANA (*Internet Assigned Numbers Authority*) attribuisce porte a protocolli – p.es.: HTTP:80, FTP:20-1, SMTP:25, ...
- Porta assegnata dinamicamente
 - Un *daemon* ascolta su porta preassegnata le richieste in arrivo
 - Le richieste per lo stesso servizio sono girate su porta assegnata dinamicamente di cui viene informato il servernte corrispondente
 - Se le richieste per un insieme di servizi sono rare si usa un Super-Servernte che ascolta tutte le porte di quell'insieme e per ogni richiesta in arrivo risveglia (o crea dinamicamente) il servernte corrispondente – p.es.: *inetd* di UNIX

Laurea Magistrale in Informatica, Università di Padova

17/27



Sistemi distribuiti: processi e concorrenza

Problematiche di lato servernte – 3

❑ **Interrompibilità del servernte**

- **Modello TCP/IP**
 - La rottura della connessione (p.es. per abbandono del cliente) comporta 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 dei pacchetti dati (p.es. TCP)
 - Cliente e servernte devono intrattenere più di una sottoconnessione logica entro la stessa connessione di servizio
 - Con porta distinta per ogni sottoconnessione

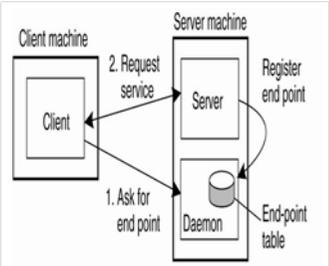
Laurea Magistrale in Informatica, Università di Padova

19/27

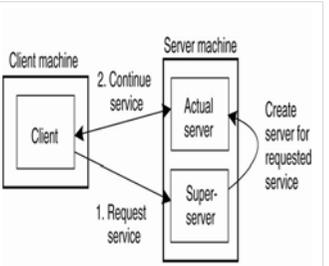


Sistemi distribuiti: processi e concorrenza

Localizzazione del servernte



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



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

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

Laurea Magistrale in Informatica, Università di Padova

18/27



Sistemi distribuiti: processi e concorrenza

Problematiche di lato servernte – 4

❑ **Servernte senza stato (*stateless*)**

- Non ricorda lo stato di servizio del cliente e 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* ma la scrittura è asincrona
 - La *cache* cliente può essere invalidata da modifiche che originano altrove
 - Il servernte tratta ogni operazione in una 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 IMPORTANT

Laurea Magistrale in Informatica, Università di Padova

20/27



Sistemi distribuiti: processi e concorrenza

Problematiche di lato servernte – 5

❑ **Servernte con stato (*stateful*)**

- Ricorda lo stato di servizio del cliente
- L'esempio classico è nei sistemi transazionali
 - `begin (op1 op2 ... opn) commit`
 - **Atomicity**: gli effetti sullo stato sono di tipo *all-or-nothing*
 - **Consistency**: lo stato di lato servernte è sempre consistente (come modificato da transazioni eseguite in un qualche ordine totale)
 - **Independence**: le transazioni non interferenti possono eseguire in parallelo
 - **Durability**: gli effetti delle transazioni terminate con successo sono persistiti
- Promesse totalmente non scalabili al crescere elastico dello stato



Laurea Magistrale in Informatica, Università di Padova

21/27



Sistemi distribuiti: processi e concorrenza

Servernte di oggetto – 2

❑ È il tramite di invocazione locale per conto di clienti remoti

- Non fornisce alcun servizio di per se

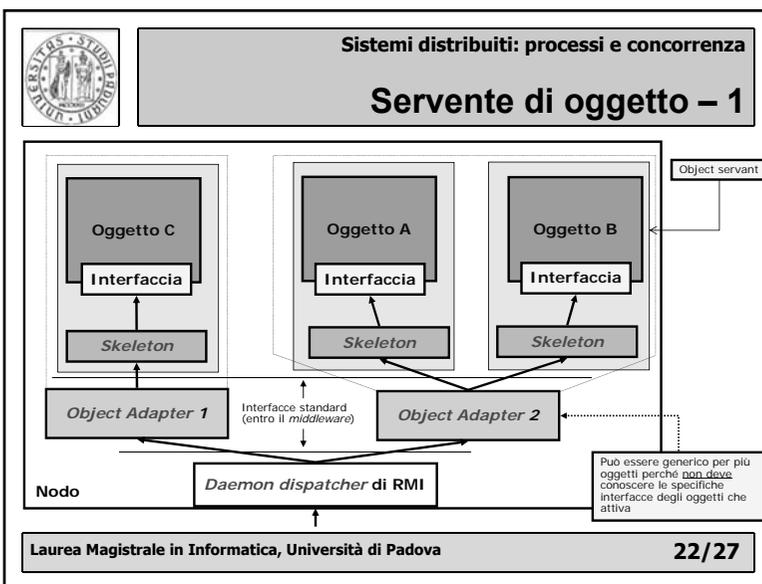
❑ La sua implementazione concreta determina come l'interfaccia e lo stato dell'oggetto remoto a lui associato possano essere separati

❑ Il servernte di oggetto può supportare diverse «politiche di attivazione» dell'oggetto remoto

- Più evoluto rispetto alla «disponibilità permanente» di RPC

Laurea Magistrale in Informatica, Università di Padova

23/27





Sistemi distribuiti: processi e concorrenza

Servernte di oggetto – 3

❑ Le politiche di attivazione fissano le modalità (il ciclo di vita) con cui un oggetto remoto può essere invocato

- Creazione e distruzione dell'*object reference*
 - Ciò che rende disponibile al cliente l'entità che realizza le operazioni dell'interfaccia
- Attivazione e de-attivazione del *servant*
 - L'insieme di risorse (CPU, memoria) che realizzano l'oggetto nel servernte

Laurea Magistrale in Informatica, Università di Padova

24/27

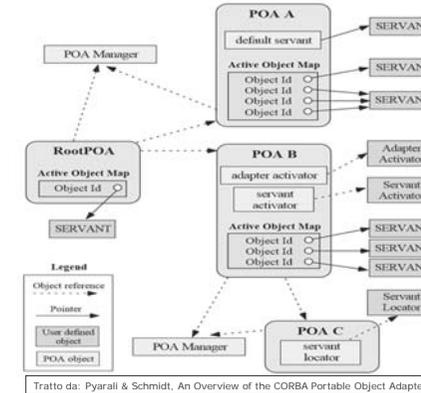


Servente di oggetto – 4

- **Politiche di attivazione comuni per nodo sono attuate da un singolo *object adapter***
 - Un 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
 - Servizio funzionale
 - Registrare o rimuovere *servant* e influenzare le politiche di servizio
 - Servizio amministrativo



Portable Object Adapter



Servente di oggetto – 5

- **I compiti dell'OA**
 - 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