



Sistemi distribuiti: processi e concorrenza

Processi e concorrenza in distribuito



Anno accademico 2013/14
Sistemi Concorrenti e Distribuiti

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

Laurea Magistrale in Informatica, Università di Padova 1/23



Sistemi distribuiti: processi e concorrenza

Considerazioni di costo – 2

- ❑ Il *context switch* a livello di *thread* non ha bisogno dell'intervento del S/O
- ❑ Il *context switch* a livello di *process* ha costo alto perché, operando a livello di virtualizzazione (memoria), deve coinvolgere il S/O
- ❑ Creare e distruggere *thread* costa molto meno che farlo con processi

Laurea Magistrale in Informatica, Università di Padova 3/23



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 del processo
 - I *thread* di un processo condividono lo stesso spazio di indirizzamento

Laurea Magistrale in Informatica, Università di Padova 2/23



Sistemi distribuiti: processi e concorrenza

Considerazioni di costo – 3

- ❑ Non è evidente che convenga supportare *thread* a livello di S/O (*kernel*)
 - 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 4/23

Sistemi distribuiti: processi e concorrenza

Considerazioni di costo – 4

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
5/23

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
7/23

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* fino a esaurimento

Laurea Magistrale in Informatica, Università di Padova
6/23

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

Laurea Magistrale in Informatica, Università di Padova
8/23

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
9/23

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) 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
11/23

Sistemi distribuiti: processi e concorrenza

Problematiche di lato cliente – 2

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

Laurea Magistrale in Informatica, Università di Padova
10/23

Sistemi distribuiti: processi e concorrenza

Problematiche di lato servente – 1

- ❑ Due possibili organizzazioni
 - Iterativa o ricorsiva → **distribuzione verticale**
 - Il servente utilizza i servizi di altri serventi (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 servente
 - Concorrente → **distribuzione orizzontale**
 - Il *front-end dispatcher* del servente 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*)

Laurea Magistrale in Informatica, Università di Padova
12/23

Sistemi distribuiti: processi e concorrenza

Distribuzione verticale – 1

The diagram shows a vertical stack of three layers: User interface (presentation), Application server, and Database server. A timeline at the bottom indicates the flow of data. The User interface sends a 'Request operation' to the Application server, which then sends 'Request data' to the Database server. The Database server returns 'Return data' to the Application server, which then returns 'Return result' to the User interface. The User interface has a 'Wait for result' period, and the Application server has a 'Wait for data' period.

Nell'architettura a distribuzione verticale il server 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
13/23

Sistemi distribuiti: processi e concorrenza

Distribuzione orizzontale

The diagram shows a 'Front-end Dispatcher' on the left, connected to a network. It sends 'Requests handled in round-robin fashion' to three 'Replicated Web servers each containing the same Web pages'. Each server has 'Disks' and is connected to a common network line that leads to the 'Internet' cloud.

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
15/23

Sistemi distribuiti: processi e concorrenza

Distribuzione verticale – 2

The diagram shows a 'Workstation' (Resolver @ client) sending a 'Richiesta ricorsiva' (recursive request) to a 'Preferred DNS Server'. The Preferred DNS Server then sends a 'Richiesta iterativa' (iterative request) to a 'Root DNS Server', which points to a 'COM DNS Server', which in turn points to a 'CONTOSO.COM DNS Server'.

- ❑ Paradigma di *name resolution* del DNS
- ❑ La richiesta iterativa sposta l'onere sul cliente
- ❑ La richiesta ricorsiva sposta l'onere sul servernte

Laurea Magistrale in Informatica, Università di Padova
14/23

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
16/23

Sistemi distribuiti: processi e concorrenza

Localizzazione del servente

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

Attivazione dinamica di servente (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
17/23

Sistemi distribuiti: processi e concorrenza

Problematiche di lato servente – 4

- ❑ **Servente senza stato (*stateless*)**
 - Non ricorda lo stato (di servizio) del cliente e non deve informarlo di eventuali cambi di stato di lato servente
 - Esempio: un *Web server* accoglie richieste HTTP sulla porta 80, le soddisfa, dimentica il cliente subito dopo, e può cambiare locazione, stato ed esistenza dei propri *file* senza doverne informare alcun cliente
- ❑ **Servente con stato (*stateful*)**
 - Il servente ricorda lo stato (di servizio) del cliente
 - Esempio: il MW di lato cliente deposita *cookie* per fornire al servente informazioni sullo stato di servizio del cliente

Laurea Magistrale in Informatica, Università di Padova
19/23

Sistemi distribuiti: processi e concorrenza

Problematiche di lato servente – 3

- ❑ **Interrompibilità del servente**
 - **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 servente 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
18/23

Sistemi distribuiti: processi e concorrenza

Servente di oggetto – 1

Può essere generico per più oggetti perché non deve conoscere le specifiche interfacce degli oggetti che attiva

Laurea Magistrale in Informatica, Università di Padova
20/23



Sistemi distribuiti: processi e concorrenza

Servente di oggetto – 2

- ❑ Non fornisce alcun servizio di per se ma opera come tramite di invocazione locale per conto di clienti remoti
- ❑ La realizzazione concreta del servente determina il modo in cui l'interfaccia e lo stato dell'oggetto remoto a lui associato possano essere separati
- ❑ Il servente di oggetto può supportare diverse «politiche di attivazione» dell'oggetto remoto

Laurea Magistrale in Informatica, Università di Padova21/23



Sistemi distribuiti: processi e concorrenza

Servente di oggetto – 4

- ❑ Politiche di attivazione comuni per nodo sono attuate da un singolo *object adapter*
 - Un noto *design pattern* della GoF
- ❑ 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

Laurea Magistrale in Informatica, Università di Padova23/23



Sistemi distribuiti: processi e concorrenza

Servente di oggetto – 3

- ❑ Le politiche di attivazione fissano le modalità 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 servente

Laurea Magistrale in Informatica, Università di Padova22/23