Processi e concorrenza

SCD

Anno accademico 2009/10
Sistemi Concorrenti e Distribuiti

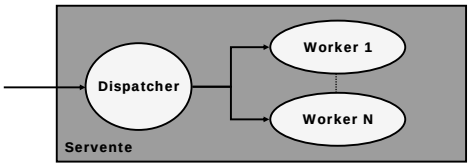
Tullio Vardanega, tullio.vardanega@math.unipd.it

Cliente e servente concorrenti – 1

- ***Multi-threading*** **di lato cliente**
  - **La concorrenza interna può mitigare l'effetto del ritardo di rete nelle comunicazioni distribuite**
    - In un *Web browser* (lato cliente) conviene eseguire in parallelo
      - Attivazione della connessione TCP/IP → operazione bloccante
      - Lettura ed elaborazione dei dati in ingresso → può operare in *pipeline*
      - Trasferimento su video → può operare in *pipeline*
  - **Il cliente può così anche supportare più sessioni parallele con le medesime caratteristiche**
    - P.es.: i "*tab*" di Mozilla

Cliente e servente concorrenti – 2

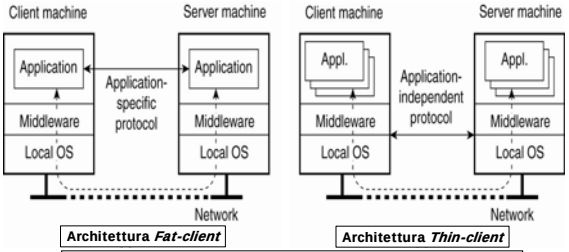
- ***Multi-threading*** **di lato servente**
  - **La concorrenza interna offre**
    - Maggiore efficienza prestazionale
      - Ancor più utile e desiderabile che nel cliente
    - Maggiore modularità (specializzazione, semplicità) architetturale



Problematiche di lato cliente – 1

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

Problematiche di lato cliente – 2

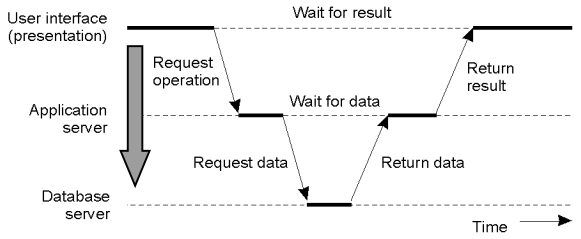


Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

Problematiche di lato servente – 1

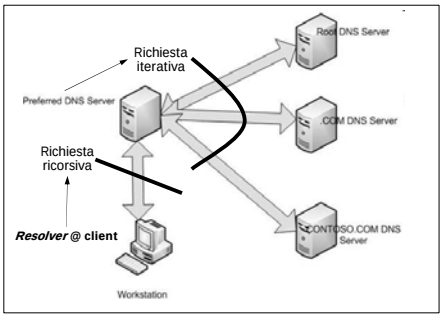
- **Due possibili organizzazioni di servente**
  - **Iterativa o ricorsiva → distribuzione verticale**
    - Il servente utilizza i servizi di altri serventi (interni o esterni)
    - La richiesta successiva potrà essere accolta **solo dopo** il completamento di quella corrente
    - Per soddisfare più richieste in parallelo bisogna replicare l'intero servente
  - **Concorrente → distribuzione orizzontale**
    - Il servente si occupa solamente di accogliere richieste demandandone il soddisfacimento a un *thread* o processo distinto
    - Nuove richieste possono essere accolte **subito dopo** che quella corrente sia stata affidata all'esecutore selezionato
    - Per soddisfare più richieste in parallelo basta replicare gli esecutori (1 *dispatcher* – N *worker*)

Distribuzione verticale – 1

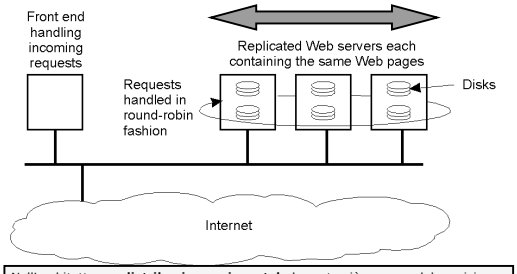


Nell'architettura a **distribuzione verticale** il servente visto dal cliente può essere esso stesso cliente di un componente servente cui sia stata demandata parte del servizio

Distribuzione verticale – 2



Distribuzione orizzontale

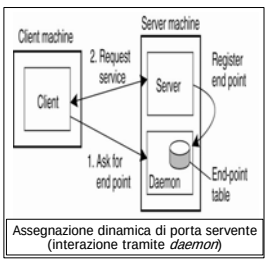


Nell'architettura a **distribuzione orizzontale** la parte più onerosa del servizio può essere completamente replicata su più elaboratori distinti operanti in parallelo

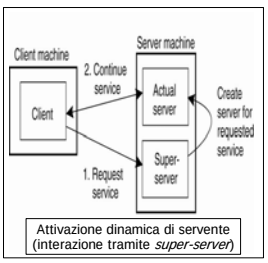
Problematiche di lato servente – 2

- **Localizzazione del servente**
  - **Al servente corrisponde una porta (*end-point*) del nodo sulla quale un processo apposito si pone in ascolto**
  - **Porta preassegnata e nota**
    - **IANA** (***Internet Assigned Numbers Authority***) attribuisce porte a specifici protocolli di livello Applicazioni, p.es.: HTTP:80, FTP:20-1, SMTP:25, …
  - **Porta assegnata dinamicamente**
    - Un *daemon* ascolta su porta preassegnata le richieste in per alcuni servizi
    - Richieste per lo stesso servizio girate su porta assegnata dinamicamente di cui viene informato il servente corrispondente
  - **Super-Servente**
    - Per richieste sporadiche non conviene mantenere serventi o *daemon* sempre attivi
    - Un super-servente ascolta **tutte** le porte di quell'insieme e per ogni richiesta in arrivo risveglia (o crea dinamicamente) il servente corrispondente, p.es.: `inetd` di UNIX

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.

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
      - Cliente e servente devono intrattenere più di una sottoconnessione logica entro la stessa connessione di servizio
        - Una porta distinta per ogni sottoconnessione
        - Designazione di urgenza nell'intestazione dei pacchetti dati (p.es. TCP)

Problematiche di lato servente – 4

- **Servente senza stato (*stateless*)**
  - **Non ricorda lo stato 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 del cliente**
  - **Esempio: il *MW* di lato cliente deposita *cookie* per fornire al servente informazioni sullo stato di servizio del cliente**
    - *Cookie* validi entro o tra sessioni

Servente di oggetto – 1

- **Non fornisce alcun servizio di per se ma agisce come tramite di invocazione locale per conto di clienti remoti**
- **La realizzazione concreta del servente determina se e come l'interfaccia e l'oggetto a lui associati possano essere separati**
- **È desiderabile che il servente di oggetto sia capace di supportare diverse politiche di attivazione (accesso e gestione) dell'oggetto remoto**

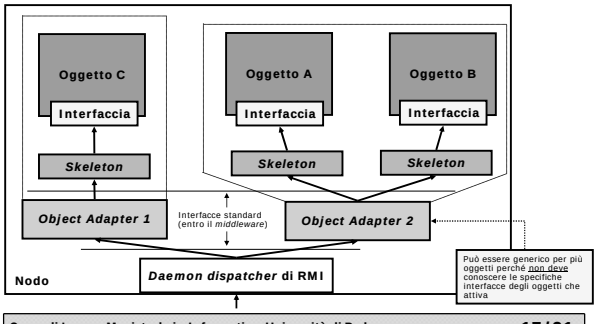
Servente di oggetto – 2

- **Politiche di attivazione determinano le modalità con le quali un oggetto remoto può essere invocato**
  - **Riguardo al caricamento dell'oggetto (detto: il *servant*) nella memoria ospite**
    - Al caricamento dell'oggetto segue la sua attivazione
    - L'oggetto può essere transitorio o persistente
  - **Conviene disaccoppiare il MW dalle politiche di servizio dell'oggetto**

Servente di oggetto – 3

- **Una politica di attivazione comune per nodo può essere realizzata da un singolo *object adapter***
  - **Concetto recepito dai *design pattern* della GoF**
  - **Fornisce metodi per**
    - Ricevere invocazioni remote in arrivo dal MW
      - Servizio funzionale
    - Inviare le invocazioni ai *servant* destinatari
      - Servizio funzionale
    - Registrare o rimuovere *servant* e influenzare le politiche di servizio
      - Servizio amministrativo

Servente di oggetto – 3



Migrazione di codice – 1

- **Utile al bilanciamento di carico**
  - **Molto importante nel passato**
  - **Oggi meno vista la potenza di calcolo disponibile**
  - **L'onere di comunicazione è diventato il vero collo di bottiglia**
  - **Migrazione del cliente presso il servente**
    - **Esempio**: un cliente deve effettuare una sequenza complessa di transazioni su base dati ove ciascuna transazione mobilizza grandi volumi di informazione
    - Conviene che la parte coinvolta del cliente operi localmente al servente per limitare il trasporto dati su rete
  - **Migrazione del servente presso il cliente**
    - **Esempio**: un servente che richiede al cliente molti dati da fornire in piccole unità di formato prefissato (p.es: *form*) come prerequisito a una transazione
    - Conviene inviare una parte del servente localmente al cliente per predisporre più velocemente i dati trattati

Migrazione di codice – 2

- **Miglioramento delle prestazioni**
  - **Esempio: parallelizzazione di ricerca su più siti inviando su ciascuno una copia dell'agente di ricerca**
- **Flessibilità di configurazione del sistema**
  - **Fissare staticamente una configurazione è difficile e rischioso quando le condizioni di lavoro non siano note o stimabili a priori**
    - Conviene di più poterla adattare dinamicamente

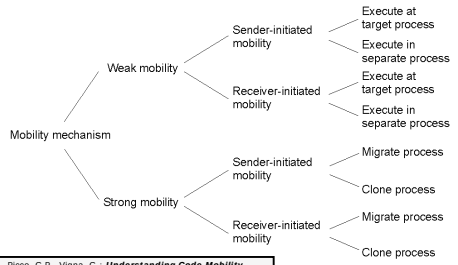
Migrazione di codice – 3

- **Modelli di mobilità (1/2)**
  - **Debole (*weak mobility*)**
    - Solo segmento codice e dati di inizializzazione → il programma migrato riparte sempre dal suo stato iniziale (p.es. Java *applet*)
    - Richiede portabilità del codice, oppure speciale supporto dal compilatore
  - **Forte (*strong mobility*)**
    - Migra anche lo stato di esecuzione → l'esecuzione continua a destinazione
    - Grande complessità realizzativa
  - **Causata dal luogo di residenza iniziale (*sender-initiated*)**
    - Da cliente verso servente → delicato, richiede **autenticazione** del cliente
  - **Causata dal luogo di destinazione (*receiver-initiated*)**
    - Da servente verso cliente (p.es. Java *applet*) → più facile e sicura

Migrazione di codice – 4

- **Modelli di mobilità (2/2)**
  - **Esecuzione nel processo destinatario**
    - **Esempio**: le Java *applet* eseguono nello spazio di indirizzamento del processo *browser* di destinazione (lato cliente)
    - Il processo destinatario deve essere protetto da codice malintenzionato
  - **Esecuzione in processo dedicato**
    - Maggiore protezione da intrusione per maggior onere di comunicazione locale
  - **Clonazione**
    - Simile al modello `fork()` di UNIX, con il processo clonato che eredita codice e stato dal processo clonante

Migrazione di codice – 5



Fuggetta, A., Picco, G.P., Vigna, G.: ***Understanding Code Mobility.*** IEEE Transactions on Software Engineering, 24(5):342-361, maggio 1998

Migrazione di codice – 6

- **Modelli di migrazione delle risorse su cui opera il codice mobile**
  - **Legame forte (*binding by identifier*)**
    - La risorsa ha identità fisica (p.es. indirizzo IP)
  - **Legame debole (*binding by value*)**
    - La risorsa non ha identità fisica ma solo specifiche caratteristiche attese e può essere copiata a destinazione
      - **Esempio**: riferimenti a librerie standard in linguaggi come C, C++, Java, Ada
  - **Legame flebile (*binding by type*)**
    - La risorsa deve solo appartenere a un tipo dato fissato che può avere più rappresentazioni
      - **Esempio**: una stampante PostScript, un dispositivo

Migrazione di codice – 7

- **Modelli di migrazione delle risorse**
  - **Risorse mobili (*unattached resources*)**
    - Il legame tra risorsa e nodo è lasco e lo spostamento ha basso costo
      - **Esempio**: un *file* dati
  - **Risorse legate (*fastened resources*)**
    - Il legame può essere lasco ma lo spostamento è oneroso
      - **Esempio**: una intera base dati, un intero sito *Web*
  - **Risorse immobili (*fixed resources*)**
    - Il legame è così stretto da non poter essere sciolto
      - **Esempio**: un dispositivo locale

Migrazione di codice – 8

Legame risorsa – nodo (+ debole → + forte)

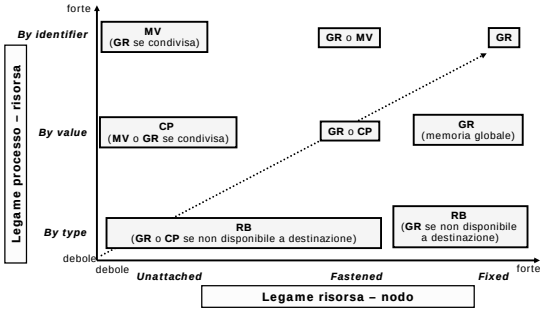
Legame processo – risorsa (+ forte → + debole)

	Unattached	Fastened	Fixed
By identifier	**MV** (**GR** se condivisa)	**GR** (o **MV**)	**GR**
By value	**CP** (**MV** o **GR** se condivisa)	**GR** (o **CP**)	**GR** (memoria globale)
By type	**RB** (**GR** o **CP** se non disponibile a destinazione)	**RB** (o **GR** o **CP**)	**RB** (o **GR**)

Cosa fare con le risorse associate a codice che migra

MV: (*move*) la risorsa può essere spostata
CP: (*copy*) il valore della risorsa può essere copiato a destinazione
GR: (*global reference*) la risorsa può essere univocamente riferita da ogni parte del sistema
RB: (*re-binding*) la risorsa può essere rappresentata a destinazione da una risorsa analoga

Migrazione di codice – 9



Agenti *software* – 1

- **Definizione 1**
  - **Unità autonome capaci di eseguire compiti collaborando con altri agenti anche remoti**
- **Definizione 2**
  - **Processo autonomo capace di reagire a cambiamenti nel proprio ambiente e di provocarne eventualmente in collaborazione con utenti o altri agenti**
- **Caratteristiche salienti**
  - **Capacità di lavoro autonomo e di collaborazione**
    - **Agenti collaborativi**

Agenti *software* – 2

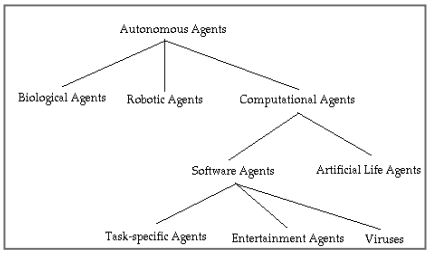
- **Agenti mobili**
  - **Con mobilità forte (più spesso) ma anche debole**
- **Agenti di interfaccia**
  - **Assistono uno o più utenti nell'uso di applicazioni**
    - **Esempio**: gli *assistant* di molte applicazioni da ufficio
  - **Hanno capacità di apprendimento**
- **Agenti di informazione**
  - **Stazionari: agiscono su informazione in ingresso**
    - **Esempio**: un filtro di posta elettronica
  - **Mobili: cercano attivamente informazione ove essa risiede**

Agenti *software* – 3

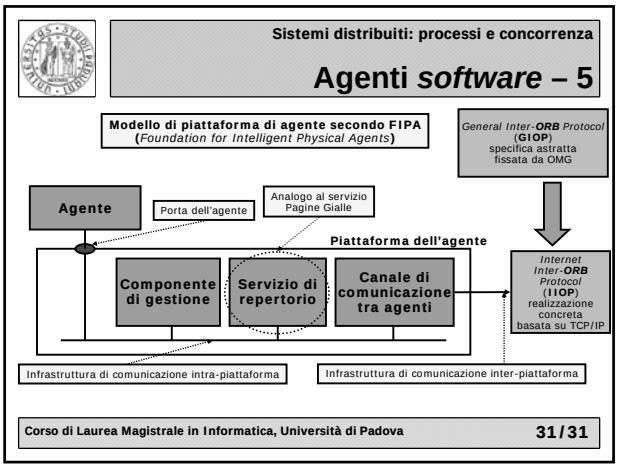
Proprietà	Di tutti?	Capacità implicata
Autonomia	**Si**	**Prendere iniziative proprie**
Reattività	**Si**	**Rispondere prontamente a cambiamenti nell'ambiente**
Proattività	**Si**	**Intraprendere azioni che producano cambiamenti nell'ambiente**
Comunicazione	**Si**	**Scambiare informazioni con utenti e altri agenti**
Continuità	**No**	**Avere un tempo di vita medio-lungo**
Mobilità	**No**	**Poter migrare da un sito all'altro**
Adattività	**No**	**Apprendimento**

Proprietà comportamentali degli agenti *software*

Agenti *software* – 4



Tratto da: http://www.msci.memphis.edu/~franklin/AgentProg.html


Sistemi distribuiti: processi e concorrenza
Agenti *software* – 5
Modello di piattaforma di agente secondo FIPA (*Foundation for Intelligent Physical Agents*)
*General Inter-**ORB** Protocol* (**GIOP**) specifica astratta fissata da OMG
Agente
Porta dell'agente
Analogo al servizio Pagine Gialle
Piattaforma dell'agente
Componente di gestione
Servizio di repertorio
Canale di comunicazione tra agenti
*Internet Inter-**ORB** Protocol* (**IIOP**) realizzazione concreta basata su TCP/IP
Infrastruttura di comunicazione intra-piattaforma
Infrastruttura di comunicazione inter-piattaforma
Corso di Laurea Magistrale in Informatica, Università di Padova
31 / 31