



Capitolo 8

Sistemi con processori multipli

**CERQUA ROSARIA
DI GIAMPAOLO BARBARA**

Introduzione

Fin dagli inizi, l'industria dei computer è stata guidata alla continua ricerca di una maggiore potenza di calcolo

Possibile soluzione:

- Impiego di computer massicciamente paralleli.
- Tali macchine si compongono di molte CPU.
- Collettivamente hanno una potenza ben maggiore di quella di una singola CPU.

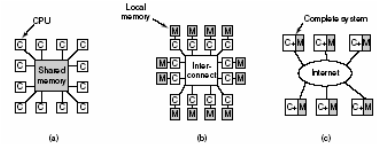
I Moderni Sistemi Operativi Cap. 8 A.S. Tanenbaum

Introduzione

Le differenti tecnologie di interconnessione danno origine a differenti tipologie di architetture di sistema.

Ad esempio:

- Sistema Multiprocessore a Memoria Condivisa (a)
- Multicomputer a Scambio di Messaggi (b)
- Sistema Distribuito (c)



I Moderni Sistemi Operativi Cap. 8 A.S. Tanenbaum

Introduzione

- Sistema Multiprocessore a Memoria Condivisa
 - 2..1000 CPU che comunicano tramite memoria condivisa
 - Leggono e scrivono le parole di memoria singole (10..50 ns)
 - Implementazione alquanto complessa
- Sistema Multicomputer a Scambio di Messaggi
 - Varie coppie CPU-Memoria Locale collegate tramite una rete ad alta velocità (messaggio spedito in 10..50 µs)
 - Più semplici da costruire ma più difficili da programmare
- Sistema Distribuito
 - Sono sistemi multicomputer collegati tramite una WAN (*Wide Area Network*), come Internet.
 - Ciascun sistema ha la sua memoria
 - Comunicano tramite scambio di messaggi.

I Moderni Sistemi Operativi Cap. 8 A.S. Tanenbaum

Multiprocessori

I multiprocessori a memoria condivisa sono sistemi di computer in cui due o più CPU condividono il pieno accesso ad una RAM comune.

- Denominati "Shared-memory MultiProcessor" (SMP).

I sistemi operativi multiprocessori hanno:

- In alcune aree, caratteristiche comuni ai S.O. usuali: filesystem, memoria, dispositivi di I/O e system call.
- Caratteristiche proprie in altre aree: sincronizzazione dei processi, gestione delle risorse e schedulazione.

Cap. 8 A.S. Tanenbaum

Hardware multiprocessore

Una differenza tra i sistemi multiprocessore è legata alla capacità di poter accedere con tempi uniformi a tutte le parole di memoria:

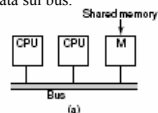
- Multiprocessori UMA (*Uniform Memory Access*): sistemi di processori che possono leggere ogni parola di memoria tanto velocemente quanto ogni altra.
- Multiprocessori NUMA(*Non Uniform Memory Access*): sistemi di processori che non godono di questa proprietà.

I Moderni Sistemi Operativi Cap. 8 A.S. Tanenbaum

Architetture SMP UMA basate sul bus

I multiprocessori più semplici sono basati su un singolo bus:

- Due o più CPU, e uno o più moduli di memoria utilizzano lo stesso bus per la comunicazione.
- Prima di leggere una parola di memoria, la CPU controlla se il bus è occupato. In tal caso aspetta che si liberi.
- Se il bus è libero, la CPU pone l'indirizzo della parola di memoria richiesta sul bus, attiva alcuni segnali di controllo e aspetta che la memoria invii la parola desiderata sul bus.



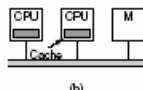
Architetture SMP UMA basate sul bus

- Con un numero elevato di CPU il problema della contesa del bus fa sì che la maggior parte dei processori rimanga inattivo per gran parte del tempo.
 - Vantaggi: flessibilità, efficienza.
 - Svantaggio: incrementa l'inattività sulla rete di interconnessione.

CPU con cache

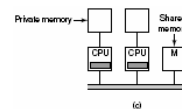
La soluzione al problema consiste nell'aggiunta di una cache a ciascuna CPU

- Molte letture soddisfatte dalla cache locale, meno traffico sul bus, il sistema supporta un maggior numero di CPU.
- Se una CPU tenta di scrivere una parola contenuta in una o più cache remote, l'hardware del bus intercetta la scrittura ed emette un segnale sul bus informando tutte le altre cache della scrittura.
- Se qualche cache ha una copia "sporca" (modificata), deve riscriverla in memoria prima che inizi la scrittura, o trasferirla direttamente alla CPU scrittrice sul bus.



CPU con cache e memorie private

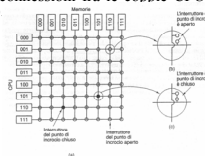
- Altra possibilità: ciascuna CPU, oltre alla cache, ha una memoria locale e privata, cui accede tramite un bus privato.
- La memoria condivisa è usata solo per le variabili condivise scrivibili.
- Questo posizionamento ridurrà il traffico sul bus, richiedendo un'attiva collaborazione da parte del compilatore (posiziona il testo del programma, le costanti, gli stack e le variabili locali nelle memorie private).



Multiprocessori UMA con Crossbar

L'uso del bus singolo limita la dimensione dei multiprocessori UMA a circa 16-32 CPU

- Per andare oltre, si può utilizzare un diverso tipo di interconnessione di rete: l'interruttore crossbar (un incrocio di linee orizzontali e verticali rappresentanti CPU e memorie).
- Ogni singolo punto di incrocio è un piccolo interruttore che può essere aperto o chiuso a seconda che le linee orizzontali e verticali debbano essere connesse o no.
- Nell'interruttore crossbar 8x8, riportato sotto, vi sono tre punti di incrocio chiusi che permettono le connessioni fra le coppie CPU-Memoria (010, 000), (101, 101), (110, 010).



Multiprocessori UMA con Crossbar

- **Rete non bloccante:** non viene mai negata a nessuna CPU la connessione di cui ha bisogno, se alcuni punti di incrocio o la linea sono già occupati.
- Problema: il numero di punti di incrocio cresce come n^2 ; con mille CPU e mille moduli di memoria, si ha bisogno di un milione di punti di incrocio.

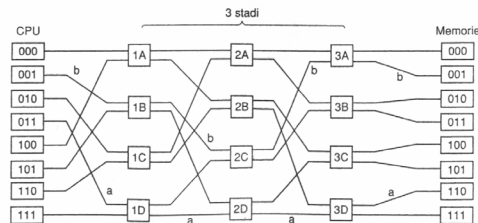
Multiprocessori UMA con reti di interconnessione a più stadi

Architettura basata su connessioni 2 x 2 (cioè 2 ingressi e 2 uscite).

- I messaggi che arrivano ad una qualunque delle linee di ingresso possono essere smistati a ciascuna delle linee di uscita.
- Ogni messaggio avrà quattro campi: Modulo (indica quale memoria utilizzare), Indirizzo (specifica un indirizzo entro il modulo), Codice Operativo (indica l'operazione) e Valore (contiene un operando, è un campo facoltativo).
- Le connessioni 2x2 possono essere sistemate in modo da costruire reti di connessioni a più stadi.
- La classe più comune di rete di connessione a stadi multipli è la cosiddetta "rete omega".

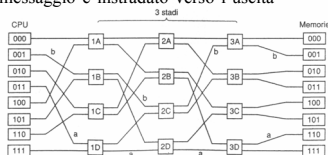
Una rete di connessione omega

- Vi sono 8 CPU connesse a 8 memorie attraverso 12 connessioni.
- Per n CPU ed n memorie necessita di $(n/2)\log_2 n$ connessioni che è molto meglio di n^2 punti di incrocio.



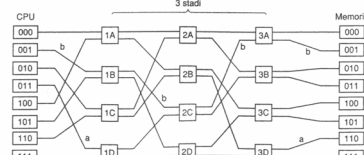
Rete omega: Esempio

- La CPU 011 vuole leggere una parola dal modulo di memoria 110.
- La CPU manda un messaggio READ alla connessione 1D, contenente 110 nel campo Modulo.
- Uno 0 conduce verso l'uscita più alta, un 1 verso quella più bassa.
- Lo switch prende il primo bit (che è 1) e lo usa per l'instradamento, il messaggio è instradato verso l'uscita più bassa, cioè 2D.



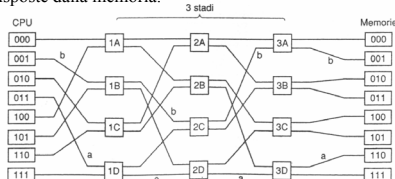
Rete omega: Esempio

- Le connessioni del secondo stadio, tra cui 2D, utilizzano il 2° bit per l'instradamento. Poiché è un 1, il messaggio è spedito verso l'uscita 3D.
- Viene testato il 3° bit, è uno 0 e quindi il messaggio passa attraverso l'uscita superiore ed arriva alla memoria 110.
- La risposta è instradata all'indietro utilizzando 011, leggendo da destra a sinistra.



Rete omega: Esempio

- Se la CPU 000 vuole contemporaneamente accedere al modulo di memoria 000, la sua richiesta cade in conflitto con quella della CPU 001 al 3° livello della connessione.
- La rete omega è **bloccante** perché possono verificarsi conflitti sia per richieste verso la memoria che per risposte dalla memoria.



Analisi delle architetture SMP UMA basate su bus

- I multiprocessori UMA a bus singolo sono generalmente limitati a non più di poche dozzine di CPU.
- I multiprocessori crossbar o basati su connessioni richiedono hardware (costoso).
- Per avere sistemi multiprocessore con un numero elevato di CPU spesso si rinuncia all'idea che tutti i moduli di memoria abbiano lo stesso tempo di accesso.

Multiprocessori NUMA

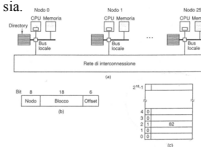
- **Caratteristiche chiave:**
 - Singolo spazio degli indirizzi visibile a tutte le CPU.
 - Accesso alla memoria remota tramite **LOAD** e **STORE**.
 - L'accesso ai moduli della memoria locale è più veloce rispetto all'accesso ai moduli remoti.
- Quando il tempo di accesso alla memoria remota non è nascosto il sistema è chiamato NC-NUMA. Quando sono presenti cache coerenti, si parla di CC-NUMA (Cache-Coherent NUMA).

Multiprocessore basato su directory

- **Approccio più seguito per costruire grandi sistemi multiprocessore CC-NUMA.**
 - L'idea è di mantenere un database che dica dove sia ciascuna linea di cache e quale sia il suo stato.
 - Ad ogni riferimento ad una linea di cache si interroga il database per trovare dov'è e se è pulita (cioè ha una copia esatta di ciò che si trova in memoria) o sporca.
 - Tale database deve essere interrogato per ogni istruzione che faccia riferimento alla memoria.
 - Conviene mantenere il database in un hardware dedicato estremamente veloce che possa rispondere in una frazione di ciclo di bus.

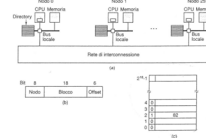
Multiprocessore basato su directory: Esempio(1)

- Si consideri un'istruzione **LOAD** dalla CPU 20, che faccia riferimento ad una linea di cache.
- La CPU spedisce l'istruzione alla propria MMU, che la trasforma in un indirizzo fisico e lo suddivide in tre parti:
Nodo 36
Linea 4
Offset 8
- La MMU si accorge che la parola di memoria riferita appartiene al nodo 36, non al nodo 20.
- Spedisce una richiesta attraverso la rete di interconnessione alla linea del nodo 36.
- Chiede se la sua linea 4 sia nella cache, e in caso affermativo, dove sia.



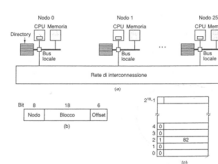
Multiprocessore basato su directory: Esempio(1)

- La richiesta arriva al nodo 36 attraverso la rete di interconnessione.
- E' instradata all'hardware della directory, che ricerca l'indice nella sua tabella di 2^{18} entrate, una per ciascuna linea di cache, ed estrae l'entry 4.
- La linea non è nella cache, e così l'hardware estrae la linea 4 dalla RAM locale, la spedisce al nodo 20 e aggiorna la entry 4 della directory, per indicare che la linea si trova ora nella cache al nodo 20.



Multiprocessore basato su directory: Esempio(2)

- Si considera la richiesta relativa alla linea 2 del nodo 36.
- Questa linea è nella cache al nodo 82. L'hardware aggiorna l'entry 2 e memorizza che la linea si trova al nodo 20.
- Mandando un messaggio al nodo 82 istruendolo a inviare la linea al nodo 20 e ad invalidare la propria cache.
- **Limitazione:**
una linea può essere nella cache di un solo nodo.



Sistemi Multiprocessore

Aspetti SW (SO)

- Tipologie di SO per Architetture Multiprocessore
- La Sincronizzazione nei Sistemi Multiprocessore
- Lo *Scheduling* nei Sistemi Multiprocessore

Tipi di sistema operativo multiprocessore

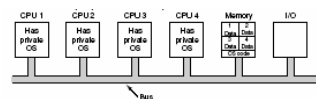
Tipologie di SO per Architetture Multiprocessore:

- Ciascuna CPU ha il proprio sistema operativo.
- Sistemi multiprocessori Master-Slave.
- Sistemi multiprocessori simmetrici.

Ciascuna CPU ha un sistema operativo proprio

Modello primitivo e desueto. Consiste nel suddividere staticamente la memoria in tante partizioni quante sono le CPU, fornendo a ciascuna CPU la propria memoria privata e la propria copia del S.O.

- Le n CPU quindi operano come n computer indipendenti.
- Quando un processo effettua una chiamata di sistema, essa viene intercettata e gestita dalla propria CPU, utilizzando le strutture dati nelle tabelle del proprio sistema operativo.



Ciascuna CPU ha un sistema operativo proprio

Vantaggi:

- Permette a tutte le macchine di condividere un insieme di dischi e altri dispositivi di I/O.
- Condivisione flessibile della memoria.

Svantaggi

- Non c'è condivisione di processi: se un utente si connette ad una CPU, tutti i suoi processi sono eseguiti su quella CPU.
- Carico sbilanciato.
- Non c'è condivisione di pagine.
- Ogni sistema operativo mantiene indipendentemente una cache di blocchi del disco portando a risultati inconsistenti.
 - (un certo blocco del disco può essere presente e sporco in diverse cache contemporaneamente).

Multiprocessori Master-Slave

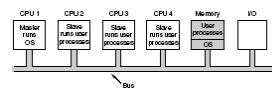
Tutte le chiamate di sistema sono indirizzate alla CPU Master su cui sono presenti il S.O. e le sue tabelle.

Vantaggi :

- Unica struttura dati per tenere traccia dei processi pronti.
- Si occupa di smistare i processi; quindi, non si verifica mai che una CPU è inattiva mentre un'altra è sovraccarica.
- Unica cache per evitare inconsistenze.
- Condivisione di pagine.

Svantaggi:

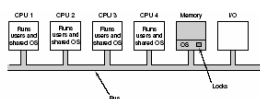
- Con molte CPU Slave, la CPU Master diventa un collo di bottiglia.
- Organizzazione adatta per piccoli sistemi multiprocessori.



Multiprocessori simmetrici

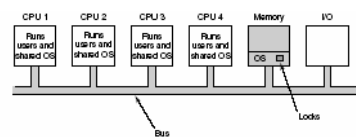
In memoria è presente una copia del sistema operativo, ma ogni CPU può eseguirlo.

- La CPU, che effettua la chiamata di sistema, effettua un trap al kernel.
- Bilanciamento dei processi e della memoria (c'è solo un insieme di tabelle del S.O.).
- Si assegna un lock (o mutex) al S.O. rendendo quest'ultimo una grande regione critica.
- Con molte CPU può formarsi una lunga coda di CPU in attesa di accedere a parti di S.O.
- Soluzione: il S.O. è formato di parti che sono indipendenti da altre.
 - Si suddivide il S.O. in regioni critiche indipendenti.
 - Ognuna è protetta da un mutex.



Multiprocessori simmetrici

- Alcune tabelle possono essere usate da più regioni critiche, quindi anche loro hanno bisogno del proprio mutex.
- Solo una CPU alla volta può accedere a ciascuna tabella critica.
- Bisogna evitare i deadlock (si assegnano dei valori interi alle tabelle, le regioni critiche acquisiscono le tabelle in ordine crescente).



Sincronizzazione dei processori

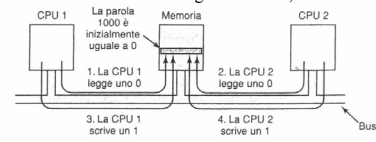
Le CPU di un multiprocessore necessitano di essere sincronizzate di frequente. Occorrono appropriate primitive per la sincronizzazione.

- Tutte le CPU devono usare e rispettare un protocollo mutex appropriato, per garantire che la mutua esclusione funzioni.
- Il cuore di ogni protocollo mutex consiste in un'istruzione che permette di leggere e scrivere una parola in una sola operazione indivisibile (TSL, Test and Set Lock).
- TSL: legge una parola di memoria e la memorizza in un registro; nello stesso tempo, scrive un 1 (o un valore diverso da 0), nella parola di memoria.

Test and Set Lock: Esempio

Peggior successione di eventi temporali:

- La parola di memoria 1000, utilizzata come lock, è inizialmente a 0.
- Al passo 1, la CPU1 legge la parola e ottiene uno 0.
- Al passo 2, prima che la CPU possa riscrivere la parola con un 1, la CPU2 legge a sua volta la parola, ottenendo uno 0.
- Al passo 3, la CPU1 scrive un 1 nella parola, e al passo 4 anche la CPU2 scrive un 1 nella parola.
- Entrambe le CPU hanno ottenuto 0 dall'istruzione TSL, e così entrambe ora hanno accesso alla regione critica, e la mutua esclusione fallisce.



Test and Set Lock

- **Soluzione:** La TSL deve prima bloccare il bus, in modo da evitare che altre CPU accedano ad esso, effettua entrambi gli accessi alla memoria e sblocca il bus.
- Può accadere che due CPU contemporaneamente leggano che il lock è libero e che facciano simultaneamente una TSL per acquisirlo.
- Il lock del bus è effettuato richiedendolo al bus stesso; ponendo a livello logico alto una linea speciale del bus (cioè 1), nessun'altra CPU può accedere al bus.
- Se TSL è implementata e usata correttamente, la mutua esclusione funziona sempre.
- Questo metodo di mutua esclusione utilizza uno **spin lock**: la CPU richiedente si impegna in un loop stretto per testare il lock più rapidamente possibile sprecando tempo e ponendo un carico massiccio sul bus o sulla memoria, il che rallenta tutte le altre CPU che cercano di svolgere il loro lavoro.

Spinning e Switching

Se un thread di una CPU ha bisogno di accedere alla cache del file system, ma tale cache ha un lock, la CPU può decidere di passare ad un thread differente, anziché aspettare.

- Operazione **Spinning**: testa continuamente il lock che, quindi, risulta poco produttiva sprecando cicli di CPU.
- Operazione **Switching**: è l'operazione di selezionare un secondo thread da eseguire una volta che si è certi che il primo è bloccato.
- Anche lo switching spreca cicli di CPU, poiché occorre salvare lo stato corrente del thread, acquisire il lock della lista dei pronti, selezionare un thread, caricare il suo stato e infine mandarlo in esecuzione.
- La ricerca è impegnata a capire quando e su quali sistemi è opportuno adoperare l'una o l'altra strategia.

Schedulazione dei multiprocessori

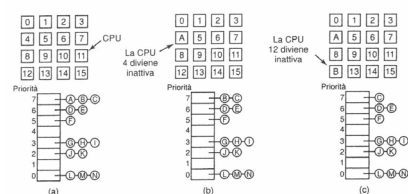
- Schedulazione bidimensionale: in un multiprocessore lo scheduler deve decidere quale processo eseguire e su quale CPU eseguirlo.
- Lo scheduler deve tenere conto anche del grado di correlazione dei processi da mandare in esecuzione.

Esempio: Nei sistemi time-sharing utenti indipendenti iniziano processi indipendenti; i processi non sono correlati e ciascuno può essere schedulato senza considerare gli altri.

Tre tipi di schedulazione: Timesharing, Space Sharing, Gang Scheduling.

Timesharing

- L'algoritmo di scheduling più semplice per processi non correlati organizza i processi pronti in una o più liste per processi a differente priorità.
 - Esempio: Le 16 CPU sono occupate ed un insieme di 14 processi con priorità sta aspettando di andare in esecuzione. La CPU4 finisce il lavoro corrente ed effettua il lock delle code di schedulazione e seleziona il processo a maggior priorità A. Poi la CPU 12 diviene inattiva e sceglie il processo B.



Timesharing

- **Vantaggi:**
La schedulazione con una sola struttura dati, utilizzata da tutte le CPU, effettua il time sharing delle CPU in gran parte come se ci trovassimo in un sistema monoprocesso, fornendo anche un bilanciamento automatico del carico.
- **Svantaggi potenziali:**
 - Contesa per la struttura dati di schedulazione, quando il numero di CPU cresce.
 - Overhead per il cambio di contesto quando un processo si blocca per un'operazione di I/O.

Timesharing

Se un processo detiene un lock gestito con cicli di attesa, le CPU in attesa del lock sprecono tempo ciclando finché quel processo è nuovamente schedato e rilascia il lock.

Soluzione: *Smart scheduling*

- Il processo, che acquisisce un lock, utilizza un flag a livello di processo per evidenziare che detiene lock.
- Quando rilascia il lock, azzerà il flag.
- Lo schedatore fornisce al processo che detiene il lock un pò di tempo in più per completare la sua regione critica e rilasciare il lock.

Timesharing

Alcuni processori utilizzano la schedulazione per affinità.

- **Schedulazione per affinità:** si esegue un processo sulla stessa CPU dove è stato eseguito l'ultima volta, perché la cache di tale CPU può contenere ancora blocchi del processo.
- Questa affinità è creata usando un algoritmo di **schedulazione a due livelli:**
 - **Livello1:** un processo, quando viene creato, viene assegnato alla CPU che al momento ha il minor carico.

Timesharing

□ **Livello2:** la schedulazione effettiva dei processi è effettuata da ciascuna CPU separatamente.

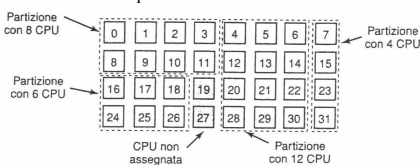
- Per mantenere un processo sulla stessa CPU si massimizza l'affinità della cache.
- Se, però, una CPU non ha processi da eseguire, li prende da un'altra per non rimanere inattiva.

Vantaggi:

- Distribuisce il carico tra le CPU in modo paritario.
- Sfrutta l'affinità della cache dove possibile.

Condivisione dello spazio

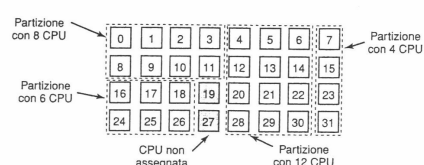
- **Space Sharing:** schedare diversi thread nello stesso tempo; utilizzato quando c'è correlazione tra i processi.
- Un processo crea un certo numero di thread correlati; l'algoritmo controlla se esiste una CPU libera per ogni thread.
- In caso positivo si fa partire l'esecuzione di tutti i thread, altrimenti si attende finché non si dispone del numero necessario di CPU libere.



Condivisione dello spazio

- Ciascun thread "conserva" la CPU fino al termine della sua esecuzione (anche se si blocca per operazioni di I/O).
- La CPU viene rimessa nell'insieme delle CPU disponibili.
- Si è soliti suddividere le CPU in partizioni, il cui numero e dimensione cambierà secondo i processi presenti.

Ogni partizione esegue i thread di un processo.



Condivisione dello spazio

Periodicamente si devono prendere decisioni di schedulazione.

- Possibile algoritmo: scegliere il processo che ha bisogno del minor numero di cicli di CPU.
- Algoritmo “primo-arrivato primo-servito”: un processo richiede un certo numero di CPU e, o le ottiene tutte, o deve aspettare finché siano disponibili.

Condivisione dello spazio

- Approccio alternativo: un server centrale tiene traccia dei processi che vogliono andare in esecuzione. Ciascuna CPU chiede al server centrale di quante CPU il server stesso disponga; in relazione alla disponibilità, aggiusta verso l'alto o verso il basso il numero di processi o dei thread.
 - Esempio: Web Server ha 10 thread, c'è maggiore richiesta di CPU e deve scendere a 5; ai prossimi 5 thread, che finiscono il loro lavoro, non viene assegnato nessun nuovo lavoro.
 - Variazione dinamica delle dimensioni delle partizioni delle CPU.

Schedulazione “gang”

- Vantaggio dello “space sharing”: elimina l'overhead dovuto al cambiamento di contesto.
- Svantaggio dello “space sharing”: tempo perduto quando una CPU si blocca e non fa niente finché non diviene di nuovo pronta.
- Da qui l'esigenza di ricercare algoritmi che tentano di schedulare spazio e tempo insieme.
 - In particolare per thread che hanno bisogno di comunicare tra loro.

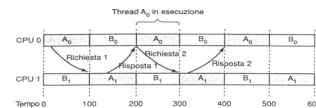
Schedulazione “gang”(1)

Problemi che si verificano quando i thread di un processo sono schedulati indipendentemente:

Esempio:

Sistema con thread A0 e A1 che appartengono al processo A. B0 e B1 al processo B.

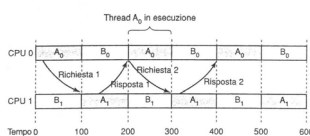
- A0 e B0 sono in time-sharing sulla CPU 0, A1 e B1 sulla CPU 1.
- A0 e A1 devono comunicare scambiandosi messaggi.
- Supponiamo che A0 e B1 partono per primi.



Schedulazione “gang”(2)

- Nella porzione di tempo 0, A0 manda una richiesta ad A1.
- A1 la riceve alla porzione di tempo 1, che inizia dopo 100 ms.
- A1 spedisce la risposta.
- A0 la riceve quando viene nuovamente eseguito dopo 200 ms.

Risultato: sequenza di richiesta-risposta ogni 200 ms, che non va bene.



Schedulazione “gang” (3)

Soluzione fornita dalla **schedulazione gang**.

- La schedulazione gang è costituita da 3 parti:
 1. Gruppi di thread correlati sono schedulati come un'unità, una gang.
 2. Tutti i membri di una gang sono in esecuzione simultaneamente, in differenti CPU in timesharing.
 3. Tutti i membri di una gang iniziano e finiscono le loro porzioni di tempo insieme.

Schedulazione "gang" (4)

- La schedulazione gang funziona perchè tutte le CPU sono schedolate in sincronia; il tempo è suddiviso in quanti discreti.
- All'inizio di ciascun nuovo quanto, tutte le CPU sono di nuovo schedolate, e un nuovo thread inizia su ciascuna di esse.
- All'inizio del quanto seguente, avviene un altro evento di schedulazione; in mezzo non si fa alcuna schedulazione, e se un thread si blocca, la sua CPU rimane inattiva fino alla fine del quanto.

Schedulazione "gang"(5): Esempio

Si ha un multiprocessore con sei CPU, usate da cinque processi da A ad E, per un totale di 24 thread pronti .

- Intervallo di tempo 0: sono schedolati ed eseguiti i thread da A0 ad A5.
- Intervallo di tempo 1: i thread B0, B1, B2, C0, C1, C2.
- Intervallo di tempo 2: sono mandati in esecuzione i cinque thread di D ed E0.
- Intervallo di tempo 3: i rimanenti sei thread del processo E.
- Il ciclo si ripete, con l'intervallo di tempo 4, identico all'intervallo 0.

	CPU					
	0	1	2	3	4	5
0	A ₀	A ₁	A ₂	A ₃	A ₄	A ₅
1	B ₀	B ₁	B ₂	C ₀	C ₁	C ₂
2	D ₀	D ₁	D ₂	D ₃	D ₄	E ₀
3	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆
4	A ₀	A ₁	A ₂	A ₃	A ₄	A ₅
5	B ₀	B ₁	B ₂	C ₀	C ₁	C ₂
6	D ₀	D ₁	D ₂	D ₃	D ₄	E ₀
7	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆

Schedulazione "gang" (6)

- Idea della schedulazione gang: avere tutti i thread di un processo in esecuzione insieme.
- Se un thread manda una richiesta ad un altro, quest'ultimo riceverà il messaggio e sarà in grado di rispondere quasi immediatamente.
- Se sono in esecuzione insieme durante un quanto, in tale intervallo di tempo possono mandare e ricevere un gran numero di messaggi.

Riepilogo

- Hardware multiprocessore :
 - Multiprocessori UMA (basate sul bus, con crossbar e a stadi multipli)
 - Multiprocessori NUMA
CC-NUMA
- Aspetti SW (SO)
- Tipologie di SO per Architetture Multiprocessore.
 - Ogni CPU -> proprio S.O.
 - Multiprocessori master-slave
 - Multiprocessori simmetrici

Riepilogo (2)

- Sincronizzazione nei sistemi multiprocessori
 - Test and Set Lock
 - Spinning e Switching
- Scheduling nei sistemi multiprocessori
 - Timesharing
 - Space-sharing
 - Schedulazione gang

Sistemi Multicomputer

- I sistemi multiprocessore offrono un semplice modello per la comunicazione, ma al crescere delle dimensioni sono difficili da costruire, e quindi costosi.
- Sono nati i sistemi multicomputer composti da CPU strettamente accoppiate, che non condividono memoria
 - cluster di computer
 - COWS(cluster di workstation)
- I multicomputer sono facili da costruire perchè la componente base è un normale PC con l'aggiunta di una scheda di rete

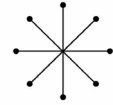
Hardware dei multicomputer

- Un multicomputer è costituito da centinaia o migliaia di nodi.
- Il nodo base di un multicomputer è formato da:
 - Una CPU
 - Una memoria
 - Un'interfaccia di rete
 - Hard disk (talvolta)

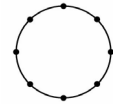
Tecnologie di interconnessione

- Ogni nodo è connesso ad altri nodi o a switch secondo una delle seguenti topologie:

➢ **Stella** : c'è uno switch singolo cui tutti i nodi sono connessi.

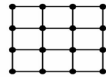


➢ **Anello** : i nodi possono formare un anello con due cavi che escono dall'interfaccia di rete, uno che va al nodo di sinistra e l'altro al nodo di destra.

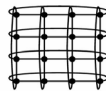


Tecnologie di interconnessione

■ **Griglia o mesh**: è un'architettura bidimensionale che ha un diametro corrispondente al più lungo cammino tra due nodi.



■ **Doppio toro**: è una griglia con i lati connessi; è più resistente agli errori e ha un diametro minore, dato che i lati opposti possono comunicare in soli due passi.

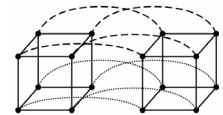


Tecnologie di interconnessione

■ **Cubo**: è una topologia regolare, tridimensionale.



■ **Ipercubo**: è un cubo n-dimensionale. Il diametro è $\log_2 n$ dove n è il numero di nodi.



Tecnologie di interconnessione

■ Nei multicomputer si utilizzano due tipi di schemi di connessione:

- commutazione di pacchetto store-and-forward
- commutazione di circuito

Tecnologie di interconnessione

■ **Commutazione di pacchetto store-and-forward**

Ciascun messaggio è prima suddiviso in parti di una determinata lunghezza massima, chiamata pacchetto. I pacchetti vengono inviati (i bit arrivano uno ad uno) dalla scheda di rete del nodo sorgente al primo switch e da questo al successivo switch lungo il cammino. Quando il pacchetto giunge allo switch del nodo di destinazione viene copiato nell'interfaccia di rete di quel nodo, ed eventualmente nella sua RAM.

➢ **Vantaggi**: flessibilità, efficienza

➢ **Svantaggi**: incrementa la latenza (ritardo) attraverso la rete di interconnessione; non si può iniziare nessuna copia finché la precedente non è finita.

Tecnologie di interconnessione

■ Commutazione di circuito

Si stabilisce un cammino prestabilito attraverso tutti gli switch, dal primo a quello di destinazione; i bit sono spediti per tutto il tempo senza pausa, dalla sorgente alla destinazione, e senza buffering intermedia negli switch.

- **Vantaggi:** è una connessione più veloce.
- **Svantaggi:** comporta una fase di inizializzazione che richiede un po' di tempo.

■ Wormhole routing

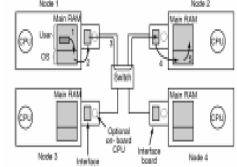
E' una variante della commutazione di circuito in cui ogni pacchetto viene suddiviso in sotto-pacchetti, permettendo al primo di fluire anche prima di stabilire l'intero cammino.

Interfacce di rete

- Tutti i nodi in un multicomputer hanno una scheda inserita per consentire la connessione del nodo alla rete di interconnessione.

La scheda di interfaccia contiene:

- **RAM:** per conservare i pacchetti in ingresso e in uscita, che, di solito, vanno copiati nella RAM della scheda di interfaccia, prima di poter essere trasmessi al primo switch (per la sincronicità delle reti).
- **DMA:** uno o più canali che permettono di copiare pacchetti fra la scheda di interfaccia e la RAM principale ad alta velocità.
- **CPU:** non necessaria, eventualmente in aggiunta ai canali DMA, consentendo alla CPU principale di scaricare parte del lavoro alla scheda di rete (multicasting, gestione della trasmissione affidabile, protezione in un sistema con processi multipli).



Software di comunicazione di basso livello

- Il nemico della comunicazione ad alte prestazioni nei sistemi multicomputer è l'eccesso di copia nei pacchetti:

- Una copia dalla RAM alla scheda di interfaccia del nodo sorgente
- Una copia dalla scheda di interfaccia del nodo sorgente a quella del nodo di destinazione
- Una copia dalla scheda di interfaccia del nodo di destinazione alla RAM di destinazione
- Eventuali copie nelle schede dei nodi intermedi

- In molti sistemi la situazione è peggiore, se la scheda di interfaccia è mappata nello spazio degli indirizzi virtuali del kernel e non in quello utente, un processo utente può mandare un pacchetto solo mediante una chiamata di sistema, che effettua una trap al kernel.

Software di comunicazione di basso livello

- Per evitare riduzione delle prestazioni, alcuni multicomputer mappano la scheda di interfaccia direttamente nello spazio utente, e permettono al processo utente di mettere i pacchetti direttamente sulla scheda, senza coinvolgere il kernel.

- Questo approccio è vantaggioso, ma presenta due problemi:

- Problemi di **condivisione tra processi**
- Problemi di **accesso da parte del kernel**

Problemi di condivisione tra processi

- Che cosa accade se più processi sono in esecuzione in un nodo e hanno bisogno di accedere alla rete per spedire pacchetti? Quale processo mette la scheda di interfaccia nel proprio spazio di indirizzi?
- Una soluzione consiste nel mappare la scheda di interfaccia in tutti i processi che se ne servono, però occorre un meccanismo per evitare le corse critiche

- **Esempio:** se al termine del processo A, a causa dell'esaurimento del tempo, il processo B passa in esecuzione e richiede lo stesso buffer di A, succede un disastro.

Problemi di accesso da parte del kernel

- Il kernel potrebbe voler accedere esso stesso alla rete di interconnessione, ad esempio per accedere al file system di un nodo remoto.

➢ Soluzione:

L'architettura più semplice consiste nell'aver due interfacce di rete, una mappata nello spazio utente per il traffico delle applicazioni, l'altra nello spazio del kernel per il sistema operativo

Comunicazione dell'interfaccia nodo-rete

■ Problemi

- Un altro problema è ricevere pacchetti sulla scheda di interfaccia: la via più rapida è servirsi del chip DMA per copiarli dalla RAM. Il problema è che il DMA utilizza soltanto indirizzi fisici, a differenza di un processo utente che riconosce soltanto l'indirizzo virtuale del pacchetto.
- Inoltre se il sistema operativo rimpiazza una pagina mentre il chip DMA vi sta copiando un pacchetto, il pacchetto verrà perso e la pagina di memoria si rovinerà.

Comunicazione dell'interfaccia nodo-rete

■ Possibili soluzioni

1. **Pinning**: chiamate di sistema per bloccare e sbloccare le pagine in memoria, marcandole come temporaneamente non paginabili.
 - Soluzione costosa, poiché bisogna effettuare prima una chiamata per bloccare la pagina contenente ciascun pacchetto uscente e poi più tardi un'altra per sbloccarla.
2. Si fa in modo che il processo utente blocchi prima una pagina all'avvio e richieda il suo indirizzo fisico. I pacchetti in uscita sono prima copiati lì, e quindi nell'interfaccia di rete
 - Richiede una copia in più
3. Utilizzare l'**I/O programmato** da e per la scheda di interfaccia è di solito la scelta più sicura, poiché ogni fault di pagina incontrato è un normale fault di pagina della CPU, ed il sistema operativo può gestirlo nel modo consueto.

Comunicazione dell'interfaccia nodo-rete

- Se le schede di interfaccia di rete hanno la loro propria CPU, queste CPU si possono usare per accelerare la comunicazione.
- Per coordinare la CPU principale con la CPU della scheda si utilizzano gli anelli di spedizione e di ricezione
- Tutti i nodi li hanno entrambi e ciascun anello ha spazio per n pacchetti e ha una mappa di n bit al fine di indicare quali posizioni dell'anello siano correntemente valide.

Comunicazione dell'interfaccia nodo-rete

■ Anello di spedizione

Quando un mittente ha un nuovo pacchetto da spedire, controlla prima se esista una posizione disponibile nell'anello di spedizione. In caso negativo deve aspettare per evitare una sovrascrittura. In caso positivo, copia il pacchetto nella successiva posizione disponibile, e alla fine imposta il bit corrispondente nella mappa di bit.

Quando la CPU della scheda ha terminato il suo compito, controlla l'anello di spedizione, se contiene pacchetti trasmette il più grande, e, quando ha finito, azzerà il bit corrispondente nella mappa di bit.

Non ci sono corse critiche, dato che la CPU principale è la sola che imposta i bit, e la CPU sulla scheda è la sola che li azzerà.

Comunicazione dell'interfaccia nodo-rete

■ Anello di ricezione

Funziona in modo simmetrico all'anello di spedizione, con la CPU della scheda che imposta un bit per indicare l'arrivo di un pacchetto, e la CPU principale che lo azzerà per indicare che ha copiato il pacchetto e ha liberato il buffer.

Non ci sono corse critiche, dato che la CPU della scheda è la sola che imposta i bit, e la CPU principale è la sola che li azzerà.

Software di comunicazione a livello utente

- I processi sulle diverse CPU comunicano attraverso messaggi. Nella forma più semplice tale scambio di messaggi è visibile ai processi utenti. I servizi di comunicazione possono essere ridotti al minimo a due chiamate:

- **Send**
 - Per inviare messaggi
- **Receive**
 - Per ricevere messaggi

Queste comunicazioni possono essere bloccanti (sincrone) o non bloccanti (asincrone).

Primitive bloccanti

■ Send(dest, &mptr)

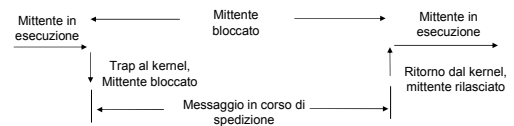
Invia il messaggio puntato da mptr al processo identificato da dest determinando il blocco del chiamante finché il messaggio non viene spedito.

■ Receive(addr, &mptr)

Determina il blocco del chiamante finché non arriva un messaggio. Quando arriva, il messaggio è copiato nel buffer puntato da mptr e il chiamante è sbloccato; il parametro addr specifica l'indirizzo su cui il ricevente è in ascolto.

Primitive bloccanti

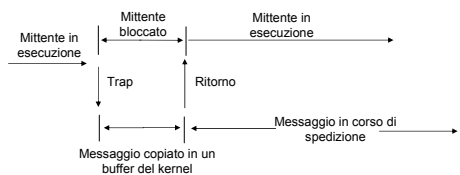
■ Send bloccante



Primitive non bloccanti

■ Send

Restituisce immediatamente il controllo al chiamante, prima che il messaggio venga spedito.



Vantaggi vs Svantaggi

■ Il vantaggio della send non bloccante deriva dal fatto che il mittente può continuare il calcolo in parallelo alla trasmissione del messaggio, invece di lasciare disoccupata la CPU.

■ Lo svantaggio è che il mittente non può modificare il buffer che contiene il messaggio fino a quando il messaggio non è stato spedito. Inoltre il processo mittente non ha nessuna idea di quando venga completata la trasmissione.

Ci sono tre possibili soluzioni...

Vantaggi vs Svantaggi

■ **Prima soluzione:** il kernel si crea una copia del messaggio in un suo buffer interno, per poi permettere al processo di continuare nell'esecuzione

> Svantaggio: ogni messaggio in uscita deve essere copiato dallo spazio utente allo spazio del kernel

■ **Seconda soluzione:** si genera un'interruzione verso il mittente quando il messaggio è stato effettivamente spedito per informarlo che il buffer è nuovamente disponibile

> Svantaggio: rende la programmazione difficile

■ **Terza soluzione:** si marca il buffer come copia in scrittura, cioè si marca in sola lettura finché il messaggio non è stato spedito; se il buffer viene riutilizzato prima viene effettuata una copia.

Primitive non bloccanti

■ Receive

Dice al kernel dove si trova il buffer e ritorna quasi immediatamente. Ci sono diversi meccanismi per avvisare dell'arrivo di un messaggio:

> Si genera un'interruzione

> Si utilizza una procedura **poll**, che rileva se ci siano dei messaggi in attesa e in caso affermativo con `get_message` viene restituito il primo messaggio arrivato

> **Thread di pop-up:** viene creato all'arrivo di un messaggio nello spazio di indirizzi del processo ricevente; dopo aver gestito il messaggio termina e viene distrutto automaticamente.

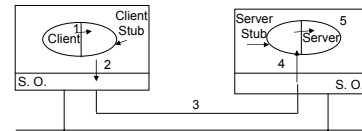
> **Schema a messaggi attivi:** ogni messaggio contiene l'indirizzo del gestore dell'interruzione, in modo tale che esso può essere chiamato con poche istruzioni.

Chiamate di procedura remota

- Un'alternativa al modello di scambio di messaggi è rappresentata dalla tecnica nota come RPC (Remote Procedure Call).
- Quando un processo sulla macchina A chiama una procedura sulla macchina B, il processo chiamante su A è sospeso, e l'esecuzione della procedura chiamata avviene su B. La procedura chiamante è nota come client, quella chiamata come server.
- L'idea è di far sembrare una chiamata di procedura remota il più possibile come una chiamata locale.
- Nessuno scambio di messaggi o I/O è visibile al programmatore.

Chiamate di procedura remota

- Passi effettivi per fare una RPC:
 1. Chiamata dal client al client stub (chiamata di procedura locale)
 2. Impacchettamento dei parametri in un messaggio(marshalling) ed esecuzione di una chiamata di sistema per spedire il messaggio.
 3. Il kernel spedisce il messaggio dalla macchina client a quella server.
 4. Il kernel passa il pacchetto in arrivo al server stub.
 5. Il server stub chiama la procedura sul server.
- La risposta effettua lo stesso cammino in direzione opposta.



Problemi implementativi

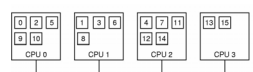
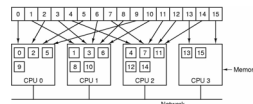
- Il passaggio dei puntatori è impossibile, perché client e server hanno un diverso spazio degli indirizzi.
- Un client stub non può fare il marshalling dei parametri se non ne conosce la dimensione.
 - Esempio: array in C
- Non è sempre possibile dedurre i tipi dei parametri
 - Esempio: printf
- Le variabili globali non sono condivise, quindi la procedura chiamante e quella chiamata(che si trova in una macchina remota) non possono comunicare tramite le variabili globali.

Memoria condivisa distribuita

- DSM (Distributed Shared Memory)
 - Lo spazio degli indirizzi è suddiviso in pagine, distribuite in tutti i nodi del sistema.
 - Quando una CPU si riferisce ad un indirizzo che non è locale, avviene una trap, il software DSM preleva la pagina che contiene l'indirizzo, e fa ripartire l'istruzione che ha provocato il fault, che ora si completerà con successo.
 - Il SO soddisfa i fault dalla RAM remota, invece che dal disco locale, e dal punto di vista dell'utente, sembra che la macchina abbia una memoria condivisa .

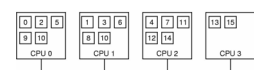
Memoria condivisa distribuita

- Come esempio abbiamo sedici pagine dello spazio degli indirizzi distribuiti fra quattro macchine
- Se la CPU 0 si riferisce alla pagina 10, causa una trap al software DSM, che muove la pagina 10 dal nodo 1 al nodo 0, come possiamo vedere.



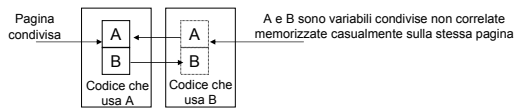
Replicazione

- Un miglioramento del sistema consiste nel replicare le pagine in sola lettura (testo del programma, costanti a sola lettura, ecc..).
- Ad esempio se la pagina 10 è una sezione di testo del programma, il suo uso tramite la CPU 0 può far spedire una copia alla CPU 0, senza che l'originale nella memoria della CPU 1 sia disturbato.



Condivisione falsa

- La dimensione di un blocco di memoria nei sistemi DSM deve essere un multiplo della dimensione della pagina. Una dimensione di pagina effettiva troppo grande introduce il problema della condivisione falsa.
- Nella situazione sottostante la pagina che contiene entrambe le variabili, viaggerà di continuo tra le due macchine. Più grande è la dimensione effettiva della pagine e più spesso avremo condivisione falsa.



Ottenere consistenza sequenziale

- Se un processo tenta di scrivere su una pagina replicata sorge un problema potenziale di consistenza. La soluzione nei sistemi DSM è la seguente:
 - Prima che una pagina condivisa sia scritta si invia un messaggio a tutte le altre CPU che hanno una copia della pagina, dicendo loro di invalidarla e scaricarla.
 - Dopo che tutte hanno risposto positivamente, la CPU originale può effettuare la scrittura.
- Si possono anche tollerare copie multiple di pagine scrivibili in situazioni circoscritte
 - **Esempio:** utilizzando un lock su una porzione dello spazio degli indirizzi virtuale

Schedulazione dei multicomputer

- Nei sistemi multiprocessore tutti i processi risiedono nella stessa memoria e potenzialmente ogni CPU può eseguirne uno qualunque.
- Nei sistemi multicomputer ogni nodo ha memoria e un insieme di processi propri e una determinata CPU non può improvvisamente decidere di eseguire un processo che si trova su un altro nodo senza prima fare un certo lavoro per acquisirlo.
- In pratica lo scheduling è più facile ed è possibile utilizzare qualsiasi algoritmo di schedulazione locale.
- Diventa fondamentale l'allocazione dei processi sui vari nodi al fine del bilanciamento del carico.

Bilanciamento del carico

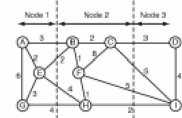
- Gli algoritmi di allocazione del processore hanno lo scopo di assegnare i processi ai nodi.
- Le proprietà di un processo che potrebbero essere note comprendono:
 - Il fabbisogno di CPU
 - L'utilizzo di memoria
 - La quantità di comunicazione con ogni altro processo
- I possibili obiettivi sono:
 - La minimizzazione dei cicli di CPU sprecati per la carenza di lavoro locale
 - La minimizzazione della larghezza di banda di comunicazione totale
 - Condizioni eque per gli utenti e i processi

Algoritmo deterministico basato sulla teoria dei grafi

- Sono noti i fabbisogni di CPU e di memoria per i processi, e una matrice che memorizza le quantità di traffico fra ciascuna coppia di processi.
- Se il numero di processi è maggiore del numero di CPU, k , diversi processi dovranno essere assegnati alla stessa CPU, minimizzando il traffico sulla rete.
 - Il sistema si può rappresentare come un grafo pesato, con un processo associato a ciascun vertice, e ciascun arco rappresenta il flusso dei messaggi fra due processi.
 - L'obiettivo è trovare la partizione che minimizza il traffico di rete, soddisfacendo tutti i vincoli (fabbisogno di CPU e di memoria per ogni sottografo).

Algoritmo deterministico basato sulla teoria dei grafi

- Vediamo un esempio in cui si allocano nove processi a tre nodi.
 - Si è partizionato il grafo con i processi A, E e G sul nodo 1, i processi B, C, F e H sul nodo 2, ed i processi D, I sul nodo 3.
 - Il traffico totale di rete è la somma degli archi attraversati dai tagli, quindi 28 unità.



Algoritmo euristico distribuito iniziato dal mittente

- Un processo, una volta creato, viene eseguito sul nodo che lo ha creato, sempre che esso non sia sovraccarico.
- Se il nodo è sovraccarico, seleziona casualmente un altro nodo e gli chiede quale sia il suo carico.
- Se questo carico è inferiore ad una certa soglia, il nuovo processo è inviato a quel nodo, altrimenti si sonda un'altra macchina.
- I tentativi di richiesta sono al massimo N, è dopo di ciò l'algoritmo termina e il processo viene eseguito sulla macchina di partenza
- Svantaggio: in condizioni di carico pesante il sistema rischia di incorrere in un considerevole overhead di richieste.

I Moderni Sistemi Operativi Cap. 8 A.S. Tanenbaum

Algoritmo euristico distribuito iniziato dal ricevente

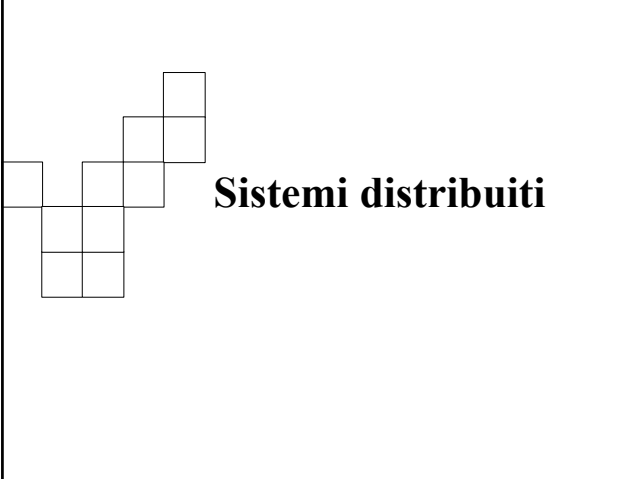
- Quando un processo termina, il sistema controlla se abbia abbastanza lavoro e, in caso negativo, sceglie una macchina a caso e la interroga riguardo al suo carico di lavoro.
- Se quella macchina non ha nulla da offrire, vengono interrogate una seconda e una terza macchina.
- Se non si trova lavoro disponibile entro N tentativi, il nodo cessa temporaneamente la ricerca, esegue il lavoro che ha accodato, e prova di nuovo quando il processo successivo termina.
- Se non è disponibile alcun lavoro, la macchina resta inattiva, e dopo un certo intervallo fisso di tempo inizia a provare di nuovo.
- Vantaggio: non impone carichi aggiuntivi sul sistema nei momenti critici; si crea traffico solo quando il sistema è poco carico.

I Moderni Sistemi Operativi Cap. 8 A.S. Tanenbaum

Algoritmo basato sull'offerta

- Ciascun nodo rende noto il suo prezzo mettendolo in un file leggibile da tutti. Nodi diversi possono avere prezzi diversi (velocità, dimensione di memoria, ecc..).
- Quando un processo vuole iniziare un processo figlio, si guarda attorno e controlla chi stia correntemente offrendo il servizio di cui ha bisogno, poi determina l'insieme dei nodi su cui servizi può fare affidamento.
- Da questo insieme, calcola il candidato "migliore" (il più economico, il più veloce, ecc...); produce un'offerta, più alta o più bassa del prezzo pubblicizzato, e la manda al primo nodo scelto.
- I processori raccolgono tutte le offerte inviate loro, ed effettuano una scelta, informano i vincitori e i perdenti ed eseguono il processo vincente.
- Il prezzo pubblicato del server è quindi aggiornato, in base al nuovo prezzo corrente.

I Moderni Sistemi Operativi Cap. 8 A.S. Tanenbaum



Sistemi distribuiti

Sistemi distribuiti

- I Sistemi Distribuiti sono costituiti da una serie di computer completi che formano una rete locale oppure geografica.

Caratteristiche principali:

- Ogni nodo è un computer completo, con una completa dotazione di periferiche.
- Ogni nodo può essere disseminato ovunque nel mondo.
- Ogni nodo ha un proprio sistema operativo con un proprio file system e può appartenere a diverse amministrazioni.

I Moderni Sistemi Operativi Cap. 8 A.S. Tanenbaum

Caratteristiche di Multiprocessori, Multicomputer e Sistemi Distribuiti

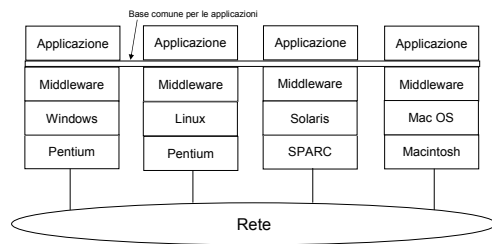
Problematica	Multiprocessore	Multicomputer	Sistema distribuito
Configurazione nodi	CPU	CPU, RAM, Interfaccia di rete	Computer completo
Periferiche dei nodi	Tutte condivise	Condivise, eccetto a volte il disco	Ogni nodo un insieme completo
Ubicazione	In un'unica macchina	Nella stessa stanza	Anche in tutto il mondo
Comunicazione fra nodi	RAM condivisa	Sistema di interconnessione dedicato	Rete tradizionale
Sistemi operativi	Uno, condiviso	Multipli, dello stesso tipo	Eventualmente tutti diversi
File system	Uno, condiviso	Uno, condiviso	Ciascun nodo ha il proprio
Amministrazione	Unica organizzazione	Unica organizzazione	Organizzazioni diverse

I Moderni Sistemi Operativi Cap. 8 A.S. Tanenbaum

Middleware

- I sistemi distribuiti aggiungono alla rete sottostante un paradigma(modello comune) che fornisce un modo uniforme di guardare all'intero sistema.
- Per ottenere una certa uniformità in presenza di hardware e sistemi operativi eterogenei, si utilizza uno strato software sopra il sistema operativo, detto middleware.
- Il middleware fornisce strutture dati e operazioni che permettono a processi e utenti su macchine lontane di interagire in modo coerente. Esso può essere interpretato come il sistema operativo di un sistema distribuito.

Posizione del middleware in un sistema distribuito



Hardware di rete

- I sistemi distribuiti sono costruiti su reti di computer.

Esistono due principali tipi di reti:

- **LAN (Local Area Networks):** sono reti locali che coprono l'estensione di un edificio o di un campus.
- **WAN (Wide Area Networks):** sono reti geografiche che possono estendersi ad una città, uno stato o anche al mondo intero.

Ethernet

- È il più importante tipo di LAN, descritta nello standard IEEE 802.3; si compone di un cavo coassiale cui sono collegati diversi computer.
- La comunicazione tra host avviene tramite pacchetto.
- Ogni computer che deve spedire un pacchetto esegue i seguenti passi:
 - > "Ascolta" sul cavo di rete per vedere se in quel momento un altro host sta trasmettendo.
 - > Se nessuno sta trasmettendo comincia la trasmissione.
 - > Se qualcuno sta trasmettendo aspetta la fine della trasmissione per iniziare la propria.

Ethernet (2)

- Se due o più host iniziano a trasmettere simultaneamente si ha una collisione, rilevata da entrambi, che rispondono terminando la trasmissione, e aspettando per una quantità di tempo casuale, per poi ripartire. Per ogni collisione il tempo massimo di attesa viene duplicato, riducendo la probabilità di nuove collisioni (alg di backoff esponenziale binario).
- Limiti di ethernet:
 - > Il cavo di rete è limitato in lunghezza.
 - > Il numero massimo di computer connessi è limitato.
- Soluzione:
 - > Diverse reti ethernet possono essere collegate tramite dispositivi chiamati **Bridge**, che permettono al traffico di passare da un'ethernet all'altra, quando la sorgente si trova in un punto e la destinazione in un altro.

Internet

- INTERNET è una federazione di reti e si è evoluta da ARPANET (1969), una rete a comunicazione di pacchetto finanziata dal Dipartimento della Difesa degli Stati Uniti.
- Internet si compone di due principali tipi di computer:
 - > **Host:** PC, portatili, server, mainframe che vogliono connettersi alla rete.
 - > **Router:** computer specializzati per la connessione, che accettano pacchetti di ingresso provenienti da una o più linee, e li ripescono attraverso una delle loro molte linee di uscita; tutti i router sono connessi insieme in grandi reti.

Internet (2)

- Tutto il traffico su internet è spedito in forma di pacchetti, ciascuno dei quali contiene il proprio indirizzo di destinazione.
- Quando un pacchetto arriva ad un router:
 - > Il router estrae l'indirizzo di destinazione e cerca una parte di esso nella sua tabella di instradamento.
 - > Invia il pacchetto sulla linea corrispondente al router indicato nella tabella.
- La tabella di instradamento di un router è molto dinamica e continuamente aggiornata poiché i router e i collegamenti possono attivarsi e disattivarsi e le condizioni di traffico cambiano.

Servizi di rete

- Esistono due tipi principali di servizi forniti agli host e ai processi di una rete:
 - > **Servizi orientati alla connessione:** l'utente stabilisce una connessione, la usa spedendo messaggi al destinatario (che li riceve nello stesso ordine) e poi la rilascia.
 - > **Servizi senza connessione:** ciascun messaggio è instradato nella rete indipendentemente da tutti gli altri. Se due messaggi sono indirizzati allo stesso destinatario non si ha la certezza su quale sia ricevuto per primo.

Servizi di rete (2)

- I servizi di rete si distinguono anche a seconda della **qualità**:
 - > **Servizi affidabili:** sono caratterizzati dal fatto che non perdono mai i dati. Questa proprietà è implementata facendo uso di acknowledgement.
 - > **Servizi non affidabili:** non si è sicuri sulla perdita dei dati.

Protocolli di rete

- Un protocollo di rete è un insieme di regole secondo le quali più computer comunicano in rete.
- I due principali protocolli internet sono:
 - > IP
 - > TCP

Middleware basato sui documenti

- Il più usato middleware basato su documenti è il **World Wide Web**.
- Ciascun computer in rete può contenere una o più pagine web, ciascuna delle quali può contenere testo, immagini, icone, suoni, video ecc, oltre a link ipertestuali ad altre pagine web.
- Ciascuna pagina web ha un unico indirizzo chiamato **URL** (Uniform Resource Locator) della forma:
 - > *protocollo://nome-DNS/nome-file*
- *Dove protocollo può essere http(il più comune), poi c'è il nome DNS dell'host che contiene il file, infine un nome di file locale, che specifica di quale file si abbia bisogno.*

Middleware basato su file system

- Utilizzare un modello di file system per un sistema distribuito significa avere un solo file system globale con gli utenti di tutto il mondo in grado di leggere e scrivere file secondo le autorizzazioni di cui dispongono.
- Un file system distribuito presenta molte problematiche tra cui:
 - > Scelta del modello di trasferimento: upload/download oppure accesso remoto
 - > Come definire la gerarchia delle directory: decidere se tutti i client abbiano la stessa vista della gerarchia delle directory o meno.
 - > Scelta della trasparenza dei nomi: trasparenza di localizzazione e indipendenza dalla localizzazione.
 - > Scelta della semantica della condivisione dei file: i cambiamenti ad un file aperto sono visibili solo al processo che li ha generati; solo quando il file viene chiuso essi diventano visibili agli altri processi (semantica di sessione).

Middleware basato su oggetti condivisi: CORBA

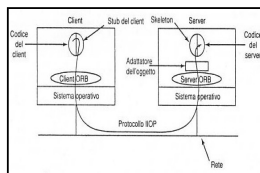
- CORBA (Common Object Request Broker Architecture) è un sistema client-server dove processi client possono invocare operazioni su oggetti localizzati su macchine server.
- CORBA è stato progettato per un sistema eterogeneo con:
 - Diverse piattaforme hardware.
 - Diversi sistemi operativi.
 - Diversi linguaggi di programmazione.
- La comunicazione tra client e server avviene tramite delle ORB (Object Request Brokers).
- Il protocollo di comunicazione di rete utilizzato da corba è detto IIOP (Internet InterORB Protocol).

Middleware basato su oggetti condivisi: CORBA (2)

- Ciascun oggetto CORBA è specificato tramite un'interfaccia, definita in un linguaggio chiamato IDL (Interface Definition Language), che determina quali metodi siano esportati e il tipo dei loro parametri.
- È possibile compilare le specifiche IDL in uno:
 - **stub**: componente usato nel client
 - **skeleton**: componente usato nel server

Middleware basato su oggetti condivisi: CORBA (3)

- Quando si crea un oggetto CORBA, si genera anche un riferimento ad esso, restituito al processo che ha creato l'oggetto stesso.
- Per chiamare un metodo di un oggetto, un processo client deve:
 - Acquisire un riferimento all'oggetto stesso
 - Impacchettare i parametri delle chiamate al metodo in una struttura adatta, e contattare il client ORB.
 - Il client ORB invia un messaggio al server ORB
 - Il server ORB chiama effettivamente il metodo dell'oggetto.
- Per rendere possibile con i sistemi CORBA l'utilizzo di oggetti scritti per altri sistemi si fornisce ad ogni oggetto un adattatore.



Middleware basato su oggetti condivisi: GLOBE

- CORBA funziona in modo accettabile in sistemi di piccole dimensioni.
- **GLOBE** è un sistema distribuito basato sugli oggetti, appositamente progettato per estendersi a miliardi di utenti.
- Le caratteristiche principali di Globe sono:
 - **Scalabilità**: si ottiene tramite la replicazione di oggetti.
 - **Flessibilità**: il sistema permette ad utenti e oggetti diversi di comportarsi in modo diversi, fornendo nello stesso tempo un modello globale coerente.
 - **Oggetti condivisi distribuiti**: gli oggetti Globe possono essere condivisi da molti processi nello stesso momento.

Middleware basato su oggetti condivisi: GLOBE (2)

- Ciascun oggetto globe è costituito da:
 - Una classe che contiene il codice dei metodi dell'oggetto.
 - Uno stato che è un'istanza dell'oggetto.
 - Una o più interfacce ciascuna delle quali contiene coppie (puntatore al metodo, puntatore allo stato).
- Per utilizzare un oggetto globe, un processo deve:
 - Trovare l'indirizzo dell'oggetto e collegarsi ad esso.
 - L'oggetto della classe dell'oggetto (il suo codice) è caricato nello spazio degli indirizzi del chiamante.
 - Viene istanziata una copia dello stato dell'oggetto e viene restituito un puntatore alla sua interfaccia.
 - Utilizzando il puntatore dell'interfaccia, il processo può chiamare i metodi sull'istanza dell'oggetto.

Middleware basato sulla coordinazione: Linda

- Linda è un sistema per la comunicazione e la sincronizzazione basato su processi indipendenti che comunicano tramite uno spazio delle tuple astratto.
- Lo spazio delle tuple è globale all'intero sistema e all'utente appare come una grande memoria condivisa globale perché i processi su una macchina possono inserire o rimuovere tuple dal loro spazio senza preoccuparsi come o dove esse siano memorizzate.
- Una tupla è composta da uno o più campi ciascuno dei quali ha un valore di un certo tipo supportato dal linguaggio di programmazione di base utilizzato.
- Le tuple sono dati puri e non hanno metodi associati

Middleware basato sulla coordinazione: Publish/subscribe

- Publish/Subscribe si compone di un certo numero di processi connessi da una rete broadcast, ciascuno dei quali può essere produttore o consumatore di informazioni.
- Quando un produttore ha delle nuove informazioni, trasmette a tutti l'informazione come una tupla; questa azione è detta pubblicazione (publish).
- I processi interessati a determinate informazioni possono sottoscrivere (subscribe) determinati soggetti.
- Una sottoscrizione si effettua richiedendo quali oggetti cercare ad un processo demone delle tuple in esecuzione sulla stessa macchina, che controlla le tuple pubblicate.

Middleware basato sulla coordinazione: Jini

- Jini è stato prodotto dalla Sun Microsystems con lo scopo di definire un modello di sistema distribuito "rete-centrico" basato su Java.
- Jini si compone di un numero di dispositivi ciascuno dei quali offre e utilizza dei servizi.
- I dispositivi Jini sono inseriti in una rete (non in un computer) e possono essere sia computer tradizionali sia altri dispositivi quali stampanti, palmari, cellulari, ecc..., purché abbiano una CPU, memoria e connessione di rete.
- Un sistema Jini è chiamato anche **federazione** di dispositivi Jini.

Middleware basato sulla coordinazione: Jini(2)

- Quando un dispositivo Jini vuole unirsi ad una federazione:
 - Trasmette un pacchetto alla LAN locale chiedendo se è presente un servizio di ricerca.
 - Per trovare un servizio di ricerca si usa un **protocollo di scoperta**.
- Quando un servizio di ricerca vede che un nuovo dispositivo vuole registrarsi:
 - Il servizio di ricerca invia una porzione di codice in JVM che permette di effettuare la registrazione.
 - Il nuovo dispositivo esegue il codice che contatta il servizio di ricerca e si registra.
- La registrazione avviene per un periodo di tempo fissato chiamato **lease** e prima della scadenza un dispositivo può registrarsi nuovamente.
- I client e i servizi Jini comunicano tramite gli **JavaSpace** (tipo Linda).
- Ogni JavaSpace si compone di un certo numero di entry fortemente tipate.

Bibliografia

- "I moderni Sistemi Operativi"
A.S. Tanenbaum