Capitolo 5

Strumenti disponibili

Anche se è ormai convinzione comune che è in genere più conveniente utilizzare negli ambienti di calcolo un insieme di workstation, anziché un singolo potente supercalcolatore centrale, ci sono stati (ed esistono tuttora) diversi ostacoli per l'affermarsi di questo modello di calcolo distribuito.

Il problema più serio è probabilmente la difficoltà nello sviluppare programmi di tipo parallelo, adatti a questo tipo di ambiente. È infatti molto più complesso scrivere programmi paralleli, rispetto a programmi puramente sequenziali: il programmatore deve gestire direttamente diversi dettagli implementativi, quali la sincronizzazione e la comunicazione tra i vari processi che costituiscono il programma, il mapping dei processi ai processori disponibili, ecc...

Per semplificare lo sviluppo di applicazioni di questo tipo, e la corrispondente esecuzione in un sistema di calcolo distribuito, negli ultimi anni sono stati sviluppati appositi ambienti visuali. Alcuni di questi vengono esaminati in appendice B, con particolare riferimento alle caratteristiche e funzionalità richieste per il sistema DEMI. Allo stato attuale non esiste, purtroppo, nessun ambiente che soddisfi tutti i requisiti necessari per il sistema da realizzare.

In questo capitolo si pone l'attenzione su un altro tipo di strumenti, utili per gestire le elaborazioni in un sistema costituito da decine, centinaia, o migliaia di workstation. Come spiegato in [34], considerando un ambiente di questo tipo, è molto probabile che si verifichino situazioni per cui ci sia almeno un calcolatore "libero", mentre molti job sono in attesa di essere eseguiti su altre workstation. Il tempo di risposta può essere, quindi, molto alto, anche se la "potenza computazionale" complessiva del sistema è sottoutilizzata.

Il problema può essere gestito in maniera adeguata con l'impiego di sistemi di load sharing, in seguito descritti.

5.1 Sistemi di load sharing

Visti i prezzi sempre più competitivi, le workstation e i personal computer sono ormai ovunque: considerando un qualsiasi ambiente del mondo scientifico o industriale, praticamente ogni utente ha il suo calcolatore personale.

Il problema è come poter sfruttare al meglio questi computer, che spesso hanno performance molto elevate. Usare, ad esempio, un'alpha station o una workstation HP solo per leggere la posta elettronica o per scrivere documenti, non significa certo raggiungere il limite della potenza del calcolatore. Inoltre, spesso questi computer sono del tutto inutilizzati dai corrispondenti "proprietari": di notte, nei weekend, all'ora di pranzo, quando ci sono riunioni, ecc... D'altro canto ci sono applicazioni che richiedono enormi risorse computazionali, per cui l'utilizzo di una singola workstation è insufficiente (o sarebbe comunque richiesto un tempo per l'esecuzione troppo elevato).

Esistono diversi sistemi di load sharing in grado di gestire questi tipi di problemi. Un *sistema di load sharing* è un sistema software utilizzato per lo scheduling di job su un insieme di workstation connesse tra loro in rete. Attraverso questo tipo di applicazioni, le varie workstation vengono viste dall'utente come un unico grande sistema di calcolo: è compito del sistema di load sharing "assegnare" i job ai computer disponibili, in modo da garantire un utilizzo efficiente delle risorse.

Come detto, nel sistema DEMI si vuole garantire l'utilizzo ottimale delle workstation, per l'esecuzione dei vari job. Per questo motivo, e per poter gestire le diverse configurazioni (nei vari siti può essere necessario considerare diverse "politiche" per la gestione di questi calcolatori: computer "dedicati" al sistema DEMI, computer di cui si possono usare solo i cicli macchina liberi, ecc...) è necessario poter utilizzare le funzionalità offerte da un load sharing system.

Sono attualmente disponibili diversi sistemi di load sharing: tra questi citiamo *Condor*, *LSF*, *Codine* [26], *DQS* [55], *IBM LoadLeveler* [29], *NQE* [21] e *NQS*. Alcuni di essi sono utilizzabili gratuitamente, altri invece (come LSF, IBM LoadLeveler, Codine e NQE) sono prodotti commerciali, che vanno quindi regolarmente acquistati.

Vengono di seguito descritti due tra i sistemi di load sharing più interessanti: Condor e LSF.

5.1.1 Condor

Condor [16] [32] [33] [36] [46] è un sistema di load sharing sviluppato da un team di ricercatori del Computer Sciences Department, University of Wisconsin, Madison, USA, diretto dal prof. Miron Livny. La prima versione del sistema risale a circa dieci anni fa; da allora sono state introdotte nuove funzionalità e supportate nuove architetture: attualmente esistono implementazioni di Condor per molte piattaforme Unix (Sun SunOS, Sun Solaris, DECStation MIPS Ultrix, DEC Alpha Digital Unix, HP HPUX, Linux). È attualmente in corso il porting a Windows NT, grazie anche ad un finanziamento da parte di Microsoft.

I file binari del sistema Condor e i relativi sorgenti sono disponibili gratuitamente.

Condor è in grado di gestire solamente i processing di batch job: non c'è supporto per elaborazioni interattive.

L'osservazione fondamentale su cui si basa il sistema Condor è che in un ambiente di lavoro, di tipo industriale o accademico, ogni persona "proprietaria" di un computer in genere utilizza tale risorsa solo per una frazione di tempo limitata. Si pensi ad esempio alle workstation utilizzate per applicazioni di CAD: si tratta di computer che hanno prestazioni molto elevate, ma che vengono usati solamente di giorno, per un'attività interattiva. Vi sono però utenti che hanno necessità di eseguire elaborazioni molto complesse, che richiedono una potenza di calcolo molto elevata: i loro computer non sono sufficienti per tutte queste computazioni.

Il sistema Condor gestisce questo tipo di problemi: ogni computer viene inserito in un *Condor pool* (che può essere costituito anche da calcolatori di diversa architettura), in modo tale che ogni utente "doni" i cicli macchina liberi del proprio computer agli altri utenti che ne hanno necessità.

È il sistema che sceglie il calcolatore su cui eseguire un processo: un batch job viene eseguito su un computer quando questo è "libero" (cioè quando il corrispondente utente non lo sta usando). Se durante l'esecuzione l'utente "reclama" l'uso del computer, il processing deve essere sospeso: l'elaborazione può riprendere solamente quando il calcolatore ridiventa nuovamente libero. È anche possibile decidere che il processo venga "migrato" su un altro computer.

L'amministratore del Condor pool definisce, attraverso espressioni logiche, quando un batch job può essere eseguito su un computer, quando un job deve essere sospeso (espressione di 'suspend'), quando l'esecuzione può riprendere (espressione di 'resume'), quando un job deve essere "migrato" su un altro computer (espressione di 'migrate'). Il sistema è configurabile a seconda delle diverse esigenze: in queste espressioni logiche possono essere usati parametri statici (es. l'architettura dei calcolatori, il tipo di processore, la memoria presente, ecc...) e due parametri dinamici: l'utilizzo di cpu, e il tempo di inattività interattiva del calcolatore.

Gli utenti sottomettono batch job al sistema attraverso script che definiscono quali sono le risorse necessarie per l'esecuzione. Ogni utente ha una propria priorità, che riflette l'uso passato delle risorse del pool: più si utilizza il sistema, più bassa diventa la priorità.

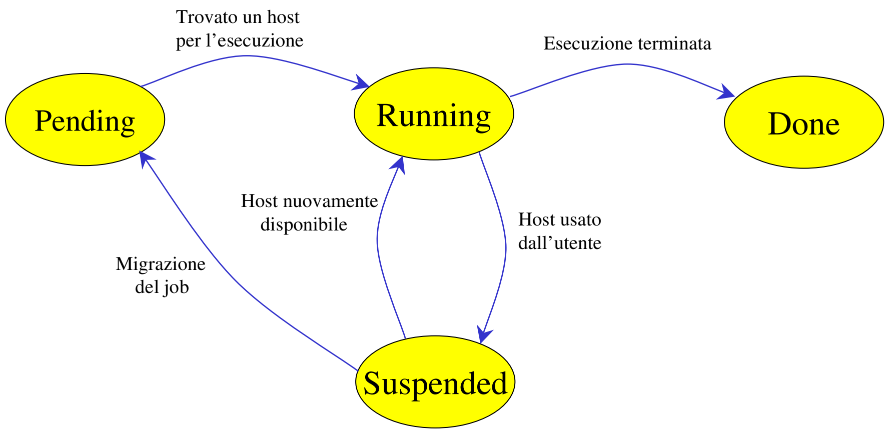

Figura 5.1: Stati di un batch job

Un job sottomesso può iniziare l'esecuzione quando vi è almeno un computer disponibile che disponga delle risorse richieste (nel caso ci siano due o più calcolatori in grado di eseguire il job, si sceglie quello che sta eseguendo un numero minore di processi).

Se durante l'esecuzione l'espressione di 'suspend' diventa vera (ad esempio perché il "proprietario" del computer riprende ad usare tale calcolatore), il job viene sospeso. Se è soddisfatta anche l'espressione di 'migrate', viene fatto un *checkpoint* del job, cioè viene fatta una copia su disco dell'immagine del job in esecuzione. A partire da questo checkpoint, il job può riprendere l'esecuzione su un altro computer, ovviamente della stessa architettura. Se invece il job non deve essere "migrato" su un altro computer, si attende finché l'espressione booleana di 'resume' diventa vera, in modo da poter riprendere l'esecuzione.

In figura 5.1 sono mostrati i possibili stati di un batch job.

Il meccanismo di checkpointing può essere utilizzato anche come sistema di fault tolerance; eseguendo periodicamente il checkpoint di un job, in caso di crash del computer dove il job è in esecuzione, il lavoro fatto fino a quel momento non viene perso: l'esecuzione può riprendere a partire dall'ultimo checkpoint.

Per quel che riguarda l'architettura del sistema Condor, una particolare workstation del pool, chiamata *Central Manager*, tiene traccia dello stato delle varie

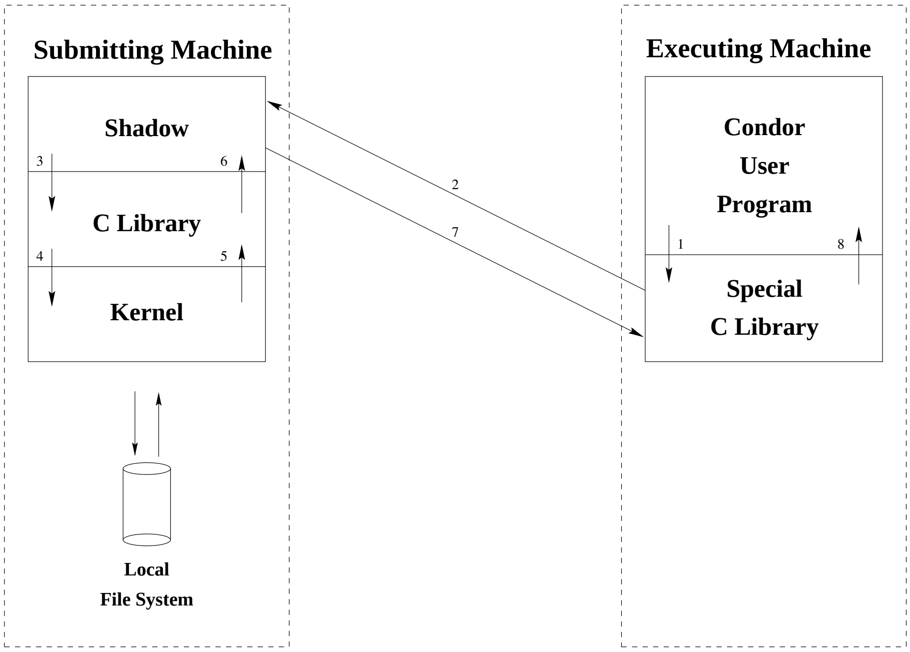

Figura 5.2: Remote system call

risorse e dei vari job. Su ogni workstation sono sempre in esecuzione due processi daemon: il processo *Startd* e il processo *Schedd*. Il processo Startd determina lo stato della workstation, verificando se essa è disponibile per eseguire job: queste informazioni vengono trasmesse al Central Manager. Il processo Startd, inoltre, gestisce l"'avvio" dell'esecuzione dei job sul computer locale, su indicazione del Central Manager. Il processo Schedd, invece, analizza quali sono i job sottomessi dagli utenti, e comunica queste informazioni al Central Manager.

Il Central Manager è il componente fondamentale del sistema: decide infatti dove i job devono essere eseguiti, in base alle informazioni comunicategli dai vari processi Schedd e Startd. Se la workstation che funge da Central Manager non è disponibile, l'attività dell'intero Condor pool è sospesa.

Non è richiesto che ogni utente abbia un account su tutte le workstation del pool, e non è nemmeno necessario che ci sia un file system comune tra tutti questi computer: un job Condor, infatti, viene eseguito come se l'ambiente di esecuzione fosse quello del computer dove il job è stato sottomesso. In figura 5.2 è mostrato come questo viene realizzato.

Il job in esecuzione sulla "executing machine" usa un'apposita versione della C library: le system call standard vengono "sostituite" da system call remote, eseguite da un processo "shadow" in esecuzione sulla "submitting machine", cioè sul computer dal quale il job è stato sottomesso. Il risultato di queste system call viene poi inviato alla macchina che sta eseguendo il job. Il processo shadow è del tutto trasparente all'utente.

Il codice dei programmi che vengono eseguiti attraverso Condor non deve essere modificato. Se si utilizza il meccanismo del checkpointing vi sono però delle limitazioni: non è possibile usare system call che creano altri processi (es. *fork*, *exec*), non è supportata la gestione dei signal, e non è possibile utilizzare meccanismi di interprocess communication (IPC).

Per quel che riguarda il linking dell'applicazione, si deve utilizzare la particolare versione della C library e, nel caso si utilizzi il meccanismo del checkpointing, un'altra specifica libreria.

Non esiste un'API che permetta di scrivere scrivere applicazioni che utilizzano direttamente le funzionalità del sistema Condor. Esiste però un'API, chiamata *CARMI* [47], che permette di sviluppare programmi PVM da eseguire poi in un Condor pool.

È anche possibile configurare un sistema costituito da diversi Condor pool, distribuiti geograficamente tra diversi siti: si parla in questo caso di *Condor flocking* [23].

5.1.2 LSF

LSF (Load Sharing Facility) [43] [44] [57] [61], precedentemente chiamato *Utopia*, è un sistema di load sharing, inizialmente sviluppato presso l'Università di Toronto, e ora venduto e supportato da Platform Computing Corporation. LSF è attualmente implementato per diversi sistemi operativi Unix (Digital Alpha, HP-UX/HPPA, IBM AIX, Linux, SGI IRIX, SunOS/Solaris, ecc...), e per Windows NT.

Attraverso questo sistema di load sharing viene gestita l'esecuzione di job, interattivi e/o batch, su un *cluster*, ovvero un insieme di computer, che possono anche essere di architetture e modelli diversi. A seconda delle condizioni di load dei vari calcolatori, e a seconda delle risorse richieste per l'applicazione (che devono essere specificate dall'utente), viene scelto per l'esecuzione il computer più adatto.

L'ambiente tipico su cui LSF opera è quello di un cluster in cui ogni utente ha un account su ogni computer, e dove vi è un file system comune. Se un utente non ha un account su tutti i computer, è possibile definire delle "associazioni" (es. si specifica che l'utente *X* dell'host *Y* corrisponde all'utente *Z* dell'host *W*). Se non c'è un file system comune tra tutte le workstation del cluster, è possibile utilizzare le apposite facility fornite dal sistema LSF per la copia di file da/a una macchina remota, per assicurarsi quindi che questi file siano presenti nel file system della workstation che esegue il job.

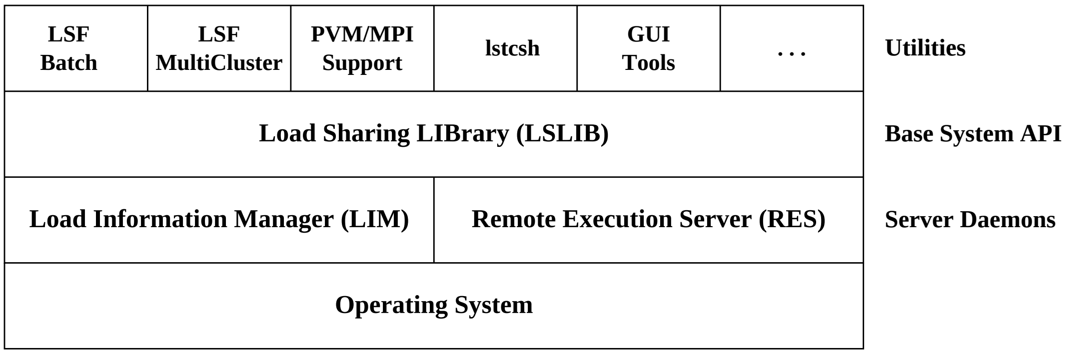

Figura 5.3: Architettura del sistema LSF

L'amministratore del cluster ha il pieno controllo sull'utilizzo dei vari computer: può decidere chi può usare un certo calcolatore e quando, può decidere quanti job di un utente (o di un gruppo di utenti) possono essere in esecuzione simultaneamente, può definire le condizioni per cui un computer può eseguire job, e le condizioni per cui i job devono invece essere sospesi, può imporre dei limiti sull'utilizzo delle risorse (cpu, memoria, ecc...) da parte dei vari job, ecc...

Per definire le condizioni di esecuzione/suspend, si possono considerare sia parametri statici (ad esempio l'architettura dei calcolatori, il loro numero di processori, la memoria presente, ecc...), che dinamici (ad esempio l'utilizzo di cpu, la memoria disponibile, l'attività di paging, l'attività di I/O, lo spazio swap disponibile, il numero di utenti che stanno usando il computer in maniera interattiva, il tempo di inattività interattiva, indici definibili dall'amministratore, ecc...).

Anche in LSF è stato implementato il meccanismo di checkpointing. In alcuni sistemi (come ConxexOS, Cray UNICOS e SPP-UX) questo viene realizzato a livello di kernel: è il sistema operativo che gestisce direttamente il checkpointing, che è del tutto trasparente all'applicazione. Nella maggior parte dei sistemi, invece, il checkpointing è implementato come "user level"[1]: nel linking dell'applicazione che deve poi essere eseguita, si deve utilizzare anche un'apposita libreria.

Non è comunque mai necessario modificare il codice del programma, e vi sono le stesse limitazioni, viste per Condor, per i programmi che utilizzano il checkpointing.

I possibili stati dei job, e le "migrazioni" tra i diversi stati, sono gli stessi di quelli visti per il sistema Condor.

L'architettura del sistema LSF è mostrata in figura 5.3.

In ogni host del cluster sono presenti due daemon: il *Load Information Manager*

[1] Il sistema di checkpointing "user level" di LSF si basa su quello implementato nel sistema Condor

(LIM) e il *Remote Execution Server* (RES): questi due processi forniscono agli utenti un ambiente uniforme, indipendente dall'architettura e dal sistema operativo dei computer.

Il processo LIM si occupa del monitor dei vari parametri del computer, e scambia queste informazioni con gli altri LIM. Tra i vari processi LIM, uno svolge le funzioni di *master*: suo compito è quello di decidere su che computer eseguire i vari job. Se vi è un crash del computer dove è in esecuzione il master, un altro LIM viene "eletto" master: il sistema LSF è quindi nel suo complesso attivo anche se solo un computer del cluster è disponibile. Il processo RES, invece, gestisce le esecuzioni dei job sottomessi da macchine remote.

La *Load Sharing LIBrary* (LSLIB) è l'Application Programming Interface (API) di LSF, che permette alle varie applicazioni di utilizzare i servizi dei processi LIM e RES. È anche possibile utilizzare direttamente questa libreria per scrivere programmi che utilizzano le funzionalità del sistema LSF [45].

Esistono poi varie applicazioni e utility, che operano attraverso chiamate a funzioni LSLIB. Tra queste applicazioni vi è la *lstcsh*, una particolare versione della shell tcsh, che permette di eseguire comandi unix o programmi sugli altri computer del cluster: il sistema LSF sceglie per l'esecuzione il calcolatore più adatto, gestendo l'I/O in maniera del tutto trasparente per l'utente.

Altre utility sono una versione load sharing di Make (che permette la compilazione/linking usando più computer contemporaneamente), tool per la gestione di job che utilizzano i sistemi di message passing PVM o MPI, un'interfaccia grafica per poter accedere facilmente alle varie funzionalità del sistema, ecc...

L'applicazione più importante è comunque *LSF Batch*, un batch system distribuito che fornisce servizi di scheduling di job sui calcolatori del cluster. L'amministratore può definire diverse code, con associati vari parametri: i computer del cluster da utilizzare, gli utenti che possono usare questa coda, la priorità rispetto alle altre code, il valore di nice con cui vengono eseguiti i job, le condizioni che devono essere soddisfatte affinché i job di questa coda possano iniziare l'esecuzione, debbano essere sospesi, debbano essere migrati su altri computer, ecc... Ad esempio, potrebbe essere definita una coda per job ad alta priorità, ma che richiedono un tempo di esecuzione limitato (definendo quindi un tempo massimo di CPU), potrebbe essere creata una coda per job a bassa priorità, che devono essere eseguiti su un computer solo se non ci sono altri job in esecuzione e se non c'è nessun utente che sta lavorando interattivamente su questo calcolatore, ecc...

Quando un utente sottomette un job ad una certa coda, questo resta in attesa finché non si rende disponibile un host adatto per l'esecuzione.

Esiste un'apposita applicazione della "suite" dei prodotti LSF, chiamata *LSF*

MultiCluster, che permette di definire relazioni tra diversi cluster LSF (l'equivalente del Condor flocking).

5.1.3 Conclusioni

I due sistemi considerati hanno caratteristiche e funzionalità diverse. Da questo confronto, LSF appare come un sistema più "evoluto" e completo rispetto a Condor.

In LSF, ad esempio, c'è il supporto per elaborazioni interattive, non previsto invece in Condor. LSF, inoltre, è più sofisticato, e meglio "configurabile": si possono definire code per specifiche esigenze, in cui si possono considerare molti parametri statici e dinamici (mentre in Condor i soli due parametri dinamici a cui si può fare riferimento sono l'utilizzo di CPU, e il tempo di inattività del computer), esistono implementazioni per un numero superiore di piattaforme (in particolare c'è il supporto anche per Windows NT), ecc... Anche per il requisito di fault tolerance, LSF è sicuramente preferibile a Condor. LSF prevede poi la possibilità di poter usare un'API, per poter sviluppare programmi che usano direttamente le funzionalità del load sharing system: questo non è possibile in Condor, a meno che non si considerino applicazioni PVM.

L'unica vera funzionalità implementata in Condor e non presente in LSF, è la possibilità di operare in un ambiente in cui non c'è un file system comune tra i vari calcolatori, senza necessità di spostare i file.

C'è da dire, infine, che LSF, a differenza di Condor, è un prodotto commerciale, con i relativi vantaggi (migliore affidabilità, possibilità di avere supporto in caso di problemi), e svantaggi (il costo).

Per il sistema DEMI si ritengono indispensabili alcune funzionalità proprie di un sistema di load sharing. Innanzittutto è necessario il supporto per la gestione di batch job, mentre non è richiesta la gestione di elaborazioni interattive. Si deve poi poter configurare il sistema (potendo considerare il numero maggiore possibile di parametri) per definire quando un job può iniziare l'esecuzione su un computer, e quando deve invece essere sospeso. Il checkpointing e la migrazione dei job possono essere considerati come funzionalità auspicabili, ma non ad alta priorità, come la possibilità di poter operare in un ambiente dove non c'è un file system comune.

A proposito del checkpointing, bisogna sottolineare che salvare su disco lo stato dei vari job ad intervalli regolari, può richiedere tempi piuttosto elevati. Oltre ad un impatto negativo sulle performance complessive, è poi necessario prevedere uno spazio disco adeguato per poter memorizzare queste immagini dei processi.

Per quel che riguarda le piattaforme supportate, deve essere possibile usare sistemi Unix (l'insieme più grande possibile), e PC Windows NT. È poi indispensabile poter utilizzare un'API, per poter scrivere applicazioni che usano le funzionalità del batch system.

Implementare un nuovo sistema che realizzi tutte queste funzionalità richiederebbe molto tempo e molte risorse (è molto complesso gestire problemi quali il load balancing, il suspend e il resume dei processi, ecc...), e comunque sarebbe un'operazione inutile, visto che esistono già sistemi che soddisfano a questi requisiti in maniera più che adeguata.

Come descritto nel capitolo 8, nella prima implementazione del sistema DEMI si sono usate le funzionalità del sistema LSF, che al momento appare come il load sharing system più completo.