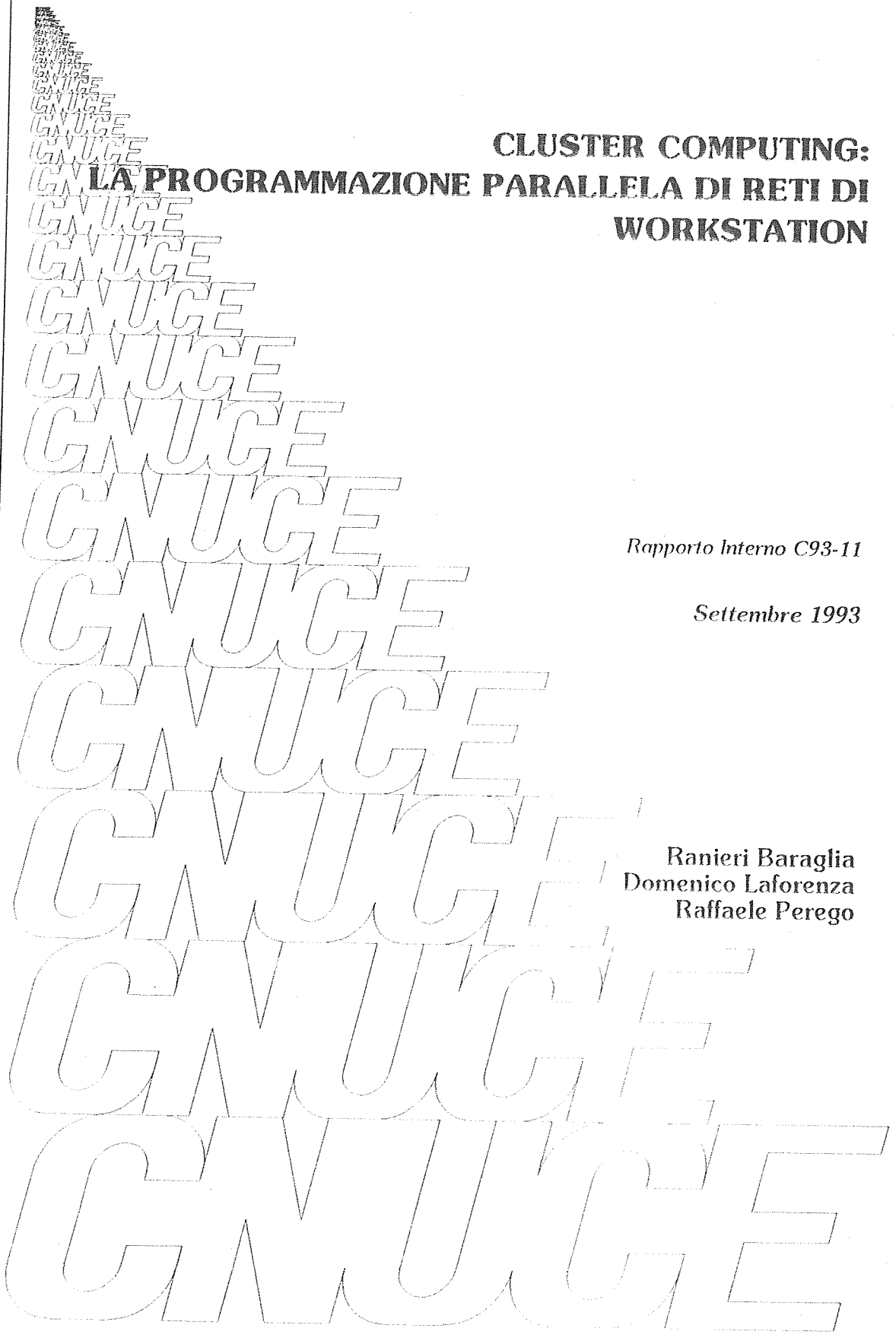


CLUSTER COMPUTING: LA PROGRAMMAZIONE PARALLELA DI RETI DI WORKSTATION

Rapporto Interno C93-11

Settembre 1993

Ranieri Baraglia
Domenico Laforenza
Raffaele Perego

SOMMARIO

Cluster Computing: la programmazione parallela di reti di workstation

(R. Baraglia, D. Laforenza, R. Perego)
CNUCE-Istituto del CNR, Reparto Calcolo Parallelo, Pisa

Introduzione

La disponibilità di workstation sempre piu' potenti[1] ed economiche e i continui miglioramenti in velocità ed affidabilità delle reti di comunicazione hanno reso oggi possibile ed economicamente vantaggioso un nuovo approccio computazionale che considera un insieme di elaboratori eterogenei interconnessi mediante rete come una unica e potente risorsa parallela di calcolo. Tra i vari termini coniati per definire questo nuovo approccio, quelli che maggiormente si incontrano nella letterattura tecnica sono: *Network Computing* e *Cluster Computing*; spesso i due termini sono usati come sinonimi anche se, a voler essere piu' precisi, essi fanno riferimento a due approcci al calcolo distribuito leggermente differenti. In particolare, il temine Network Computing e' piu' generale del primo e definisce l'utilizzo collettivo di una rete composta da piu' elaboratori di potenza e caratteristiche architetturali differenti nonche' da *server* e da periferiche di vario genere. Il termine Cluster Computing, invece, viene generalmente usato quando gli elaboratori in questione sono delle semplici workstation. Il presente articolo riguarda in generale le tecnologie e gli strumenti che hanno reso proponibile questo nuovo utilizzo degli elaboratori e, in particolare, due strumenti per la programmazione parallela di una rete di elaboratori.

---

[1] Le prestazioni dei microprocessori sono cresciute al tasso del 35% annuo negli ultimi 10 anni.

Cluster Computing

Andrew S. Tanenbaum definisce un *sistema di calcolo distribuito* (SCD) come una architettura composta da più nodi di elaborazione, ciascuno in grado di eseguire un proprio flusso di istruzioni, che non condividono alcuna memoria principale e cooperano secondo meccanismi di scambio di messaggi mediante una rete di interconnessione che li connette (tutti o in parte).

Una sotto-classificazione dei sistemi di calcolo distribuito puo' essere effettuata sulla base della rete usata per interconnettere i nodi. Infatti, essa oltre che a definire la topologia di collegamento, determina anche velocità ed affidabilità della comunicazione.

I sistemi di calcolo distribuito si suddividono in:

- *tightly-coupled* o *closely-coupled* (strettamente accoppiati)
- *loosely-coupled* (lascamente accoppiati).

I primi dispongono di una rete di interconnessione veloce ed affidabile. Il costo della comunicazione è relativamente basso, tipicamente variabile dai microsecondi ai millisecondi. Ciascun processore è collegato punto-a-punto a tutti gli altri o ad un loro sottoinsieme e la loro distanza varia da pochi centimetri a qualche metro. Esempi tipici di tali sistemi sono gli elaboratori ad elevato parallelismo, MIMD-DM[1] , ad esempio: Intel iPSC/860, nCUBE 2, TMC CM-5, Meiko Computing Surface, ecc..

I SCD loosely-coupled dispongono invece di una rete di interconnessione piu' lenta e di affidabilità moderata. Il costo della comunicazione puo' variare in questo caso da pochi millisecondi a qualche secondo, e la distanza tra i nodi di elaborazione da qualche metro-chilometro a centinaia-migliaia di chilometri, nel caso in cui il collegamento sia effettuato, rispettivamente, mediante LAN (*Local Area Network*) o WAN (*Wide Area Network*).

---

[1] Multiple-Instruction stream Multiple-Data stream Distributed-Memory

Sulla base di questa classificazione, un cluster di workstation, la cui rete di interconnessione sia una rete locale con prestazioni e costi dipendenti dalla tecnologia utilizzata, può essere visto come un sistema MIMD a memoria distribuita di tipo *loosely-coupled*.

Cluster Computing: Omogeneo o eterogeneo

In generale, si dice che un elaboratore parallelo è *omogeneo* (a livello di codice eseguibile) se un programma, una volta compilato, può essere eseguito su ciascuno dei nodi di elaborazione che compongono il sistema. Esempi di elaboratori paralleli omogenei sono le architetture ad ipercubo e gli array di Transputer. In questi casi, infatti, fatta eccezione per l'eventuale computer "host", tutti i nodi hanno la stessa architettura di base e sono caratterizzati da compatibilità a livello di codice eseguibile. Estendendo questa definizione si ha che se quest'ultima proprietà vale per le workstation all'interno di un cluster, siamo in presenza di un *cluster omogeneo*. Al contrario, un cluster composto da workstation di differenti venditori non garantisce la compatibilità a livello di codice eseguibile tra le diverse macchine e viene per questo chiamato *cluster eterogeneo*

.

Occorre precisare che il termine omogeneo/eterogeneo si pone anche in altre accezioni oltre quella prospettata in precedenza. Ad esempio, un elaboratore parallelo è detto *omogeneo* a livello di prestazioni, se tutti i nodi di elaborazione hanno stessi valori di performance.

Luci ed ombre sul Network Computing

Il Network Computing offre molteplici vantaggi in quanto e' in grado di garantire elevate potenze computazionali a costi contenuti, buona scalabilità ed espandibilità della configurazione, cioe' il numero di nodi di elaborazione utilizzabili da una applicazione può variare a

seconda delle sue necessita' computazionali. Inoltre, nel caso in cui siano presenti in rete elaboratori eterogenei, si ha la possibilita' di eseguire parti diverse di una applicazione complessa su quegli elaboratori che meglio si adattano ad esse, per caratteristiche architetturali o computazionali.

Dal punto di vista economico, il Network Computing rappresenta una buona opportunita' di risparmio in quanto permette di utilizzare al meglio le risorse computazionli gia' presenti nelle varie organizzazioni. Infatti, in questi ultimi anni, nel mondo dell'informatica, si e' assistito ad un interessante fenomeno definito *downsizing*[2].

Con l'avvento delle potenti workstation RISC, caratterizzate da vantaggiosi rapporti prezzo/prestazioni rispetto ai tradizionali mainframe, molti dipartimenti aziendali, universita' e istituti di ricerca hanno acquistato un considerevole numero di stazioni di lavoro. In generale pero', tranne poche eccezioni, queste macchine tendono ad essere parzialmente cariche durante i periodi diurni e praticamente inutilizzate durante la notte. A tal riguardo alcune statistiche indicano che il tempo di *idle*[3] medio di una workstation in rete puo' variare dal 50% al 95%[4] , e questo indica che una notevole potenza di calcolo resta inutilizzata. Per quantizzare tale potenza basti pensare che se si disponesse, ad esempio, di una rete di dodici

---

[2]Donwsizing - Non e' una definizione accettata universalmente; con essa, in generale, si intende la migrazione di applicazioni da mainframe a sistemi piu' piccoli, tipicamente cluster di PC o di workstation.

[3]Tempo di inattivita' durante il quale un elaboratore e' privo di lavoro da eseguire.

[4]Da una indagine condotta negli Stati Uniti presso il Lawrence Livermore National Laboratory e' risultato che un buon numero di workstation rimane in stato di idle per un tempo compreso tra il 95% e il 97% (quasi 22 ore al giorno), con conseguente sperpero di un elevatissimo numero di cicli di macchina.

workstation ciascuna occupata al 50%, questo equivarrebbe ad averne mediamente a disposizione sei. Poiche' una odierna workstation tipicamente puo' raggiungere prestazioni di picco dell'ordine 80-160 MFlop/s[5] pari a 25-50 MFlop/s "sul campo"[6], sei workstation equivarrebbero a una potenza computazionale pari a 150-300 Mflop/s (test eseguiti con Linpack[7] su un supercalcolatore CRAY Y-MP monoprocessore forniscono valori di 150 MFlop/s). Da queste considerazioni discende che la globalita' delle workstation presenti in varie organizzazioni rappresenta una notevolissima capacita' computazionale disponibile a costi praticamente nulli. Cio' dovrebbe spingere molte aziende, soprattutto quelle impegnate in settori tecnico-scientifici a prendere in seria considerazione questo nuovo approccio; infatti, molte di esse, pur necessitando di notevole potenza di calcolo, spesso non dispongono delle risorse (finanziarie o organizzative) necessarie per effettuare grossi investimenti, ne' in costosissimi supercalcolatori, ne' in "esotiche" e sofisticate architetture a parallelismo massiccio.

Il Network Computing solo da poco ha cominciato a prendere piede e, al momento, non esiste ancora un consenso unanime a suo riguardo; in molti ambienti, infatti, l'idea di condividere la propria workstation con altri e' vista con estremo scetticismo che spesso si tramuta in pura ostilita'. Argomenti quasi sempre sfoderati per evitare che la propria

---

[5]Million of Floating-Point Operations per Second. E' una misura delle prestazioni usata per comparare la capacita' computazionale di elaboratori per uso tecnico-scientifico.

[6]Riferim. "Peformance of Various Computers Using Standard Linear Equations Software", J.J. Dongarra, July 22, 1993.

[7]Linpack e' un insieme di sottoprogrammi usati in applicazioni scientifiche per eseguire analisi linerare. Vengono usati come benchmark per valutare le prestazioni di elaboratori tecnico-scientifici e supercalcolatori.

stazione di lavoro sia condivisa da altri sono, ad esempio, la assoluta necessita' di garantire la necessaria sicurezza e *privacy* delle informazioni contenute in essa, la difficolta' di gestione del cluster, ecc.. In effetti, allo stato attuale dell'arte, la sicurezza rappresenta una delle principali difficoltà connesse all'utilizzo di un ambiente basato su rete e la gestione di un insieme di macchine in rete presenta ancor'oggi problemi di natura tecnica e, soprattutto, organizzativa.

Tecnologie per Network Computing

In questo paragrafo si fa il punto delle tecnologie che hanno permesso di considerare realmente praticabile la via del Network Computing.

Workstation

L'introduzione, avvenuta intorno alla meta' degli anni '80, di potenti ad affidabili workstation basate su architettura RISC ha rappresentato una autentica rivoluzione ed e' stata una delle cause principali della cosiddetta "crisi del mainframe". Infatti, le nuove stazioni di lavoro offrono potenze alquanto elevate, certamente competitive rispetto a quelle di un mainframe tanto da cominciare ad "erodere" anche il tradizionale primato spettante ad alcuni supercalcolatori. Con questo non si intende assolutamente affermare che una workstation o un insieme di esse e' oggi equiparabile in tutto per tutto ad un supercomputer, ma solamente che la loro potenza si presta a eseguire applicazioni via via piu' complesse e computazionalmente costose che, solo qualche anno fa, erano di esclusivo dominio dei supercalcolatori. Inoltre, l'estrema competizione del mercato delle workstation ha prodotto costanti riduzioni di costo, garantendo un rapporto prezzo/prestazioni (costo per MFlop/s) alquanto basso.
In Tabella 1 sono riportati alcuni dati tecnici ed i valori di prestazione relativi ad alcune delle piu' diffuse workstation presenti sul mercato.

	HP	DEC	IBM
Modello	9000/735	3000-500	POWERstation 580
Clock (MHz)	99	150	62.5
Memoria Max. (MBytes)	400	1000	1000
Peak Performance (MFlop/s)	198	150	125
Linpack Perf. (n=100)	41	30	38
SPECfp92[8]	150.6	122	133.2

Tab. 1: principali caratteristiche di alcune workstation

Tecnologie di rete

L'odierna tecnologia mette a disposizione svariati strumenti per interconnettere in rete un insieme di elaboratori, e sono possibili diverse scelte ed opzioni in funzione di velocità, affidabilità e costo della struttura di interconnessione che si intende realizzare. Di seguito vengono elencati i principali standard di interconnessione esistenti:

- Ethernet[9] (10 Mbit/sec);
- FDDI[10] - Fiber Distributed Data Interface (100 Mbit/sec);
- HiPPI[11] - High Performance Parallel Interface (1 Gbit/sec);
- Fiber Channel[12] (133 Mbit/sec, 266 Mbit/sec, 1 Gbit/sec);

---

[8]SPECfp92 - benchmark costituito da un insieme di 10 programmi Fortran e C per la valutazione di elaboratori UNIX. Introdotto e gestito dalla societa' statunitense *System Performance Evaluation Cooperative*, in collaborazione con la maggior parte dei fornitori hardware.

[9]Ethernet e' lo standard piu' comune e diffuso per costruire reti locali.

[10]FDDI e' uno standard ANSI (American National Standards Institute) nel campo delle reti locali in fibra ottica.

[11]HiPPI e' uno standard ANSI nel campo delle reti locali per elevare prestazioni, in particolare per connettere supercalcolatori e particolari periferiche di memorizzazione.

[12]Fiber Channel e' uno standard ANSI originariamente introdotto per fornire link ad

Tra parentesi è riportato l'ordine di grandezza dell'ampiezza di banda di ciascuna tecnologia, cioe' la quantità di dati, espressa in bit, che puo' essere inviata da un nodo all'altro nell'unità di tempo. Questa misura rappresenta un valore di picco, generalmente non raggiungibile in pratica, in quanto la trasmissione dati non è esente da errori e necessita quindi della mediazione di appositi protocolli di comunicazione per raggiungere un desiderato grado di affidabilità.

Topologie di rete

Esistono diverse topologie di interconnessione. Nelle reti locali sono tipicamente adottate strutture di interconnessione non dedicate, cosiddette a diffusione. I casi più frequenti sono quelli dell'organizzazione monobus (Fig. 1) e ad anello (Fig. 2). Possono essere presenti più bus/anelli connessi ad ogni unità per ragioni di efficienza e/o affidabilità. L'organizzazione monobus e' adottata da Ethernet.

L'anello è una linea di comunicazione unidirezionale chiusa su se stesso, che attraversa le unità di elaborazione. Uno standard è rappresentato dal Token-Ring. Una rete ad anello presenta un costo alquanto contenuto, ma introduce problemi di latenza[13] se si aumenta il numero dei nodi dell'architettura. Per diminuirne i tempi di latenza, può essere realizzata, ad esempio, mediante FDDI.

E' possibile interconnettere una serie di elaboratori anche secondo una topologia a stella, facendo uso di particolari switch di interconnessione definiti *Hub*(vedi Fig. 3). Ciascun link di comunicazione tra

---

elevate prestazioni tra elaboratori situati a media distanza (sino a 10 km). Per la sua flessibilita' ed elevata capacita' trasmissiva si presta bene anche ad interconnettere elaboratori e periferiche situati a breve distanza.

[13]La latenza è il tempo richiesto affinché un messaggio di lunghezza zero sia inviato da un calcolatore all'altro.

elaboratore e hub puo' essere realizzato in rame (come per HiPPI) o in fibra ottica (come per FCS). L'uso di tali dispositivi, il cui costo al momento e' alquanto elevato rispetto ad altre soluzioni (3000-15.000 dollari per porta), vengono tipicamente usati per interconnettere elaboratori o periferiche situati all'interno di una sala macchine.

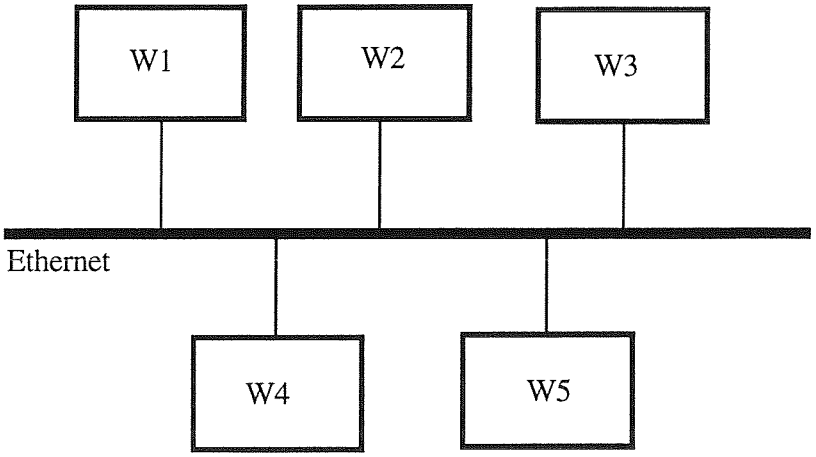


Fig. 1 rete di interconnessione monobus

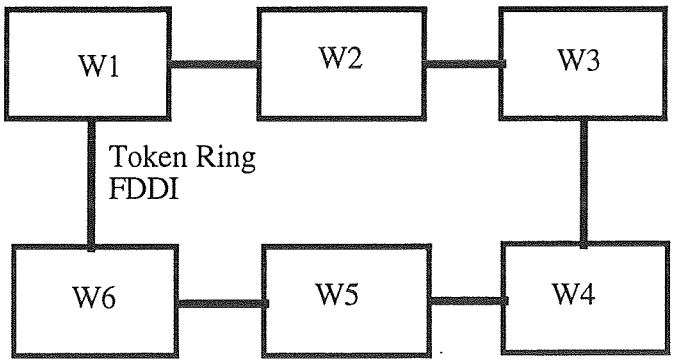


Fig. 2 rete di interconnessione ad anello

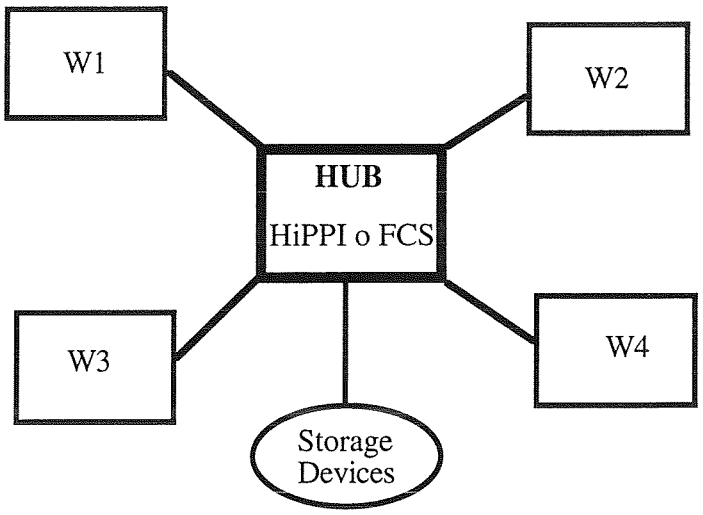


Fig. 3 rete di interconnessione ad Hub

Protocolli di comunicazione

Le modalita' di comunicazione principali, adottati da una serie di protocolli sono:

- *datagram*, adottato dal protocollo UDP/IP (*User Datagram Protocol/Internet Protocol*);
- circuiti virtuali, adottato dal protocollo TCP/IP (*Transmission Control Protocol/Internet Protocol*).

Datagram non effettua alcun controllo sulla correttezza della comunicazione. In particolare, il mittente non verifica se i messaggi/pacchetti sono stati ricevuti correttamente; a destinazione non viene effettuato alcun controllo sulla presenza di messaggi/pacchetti fuori ordine, perduti o duplicati. Nel caso di circuiti virtuali, invece, si garantisce l'esistenza di una comunicazione esente

da errori e inconsistenze. Vengono infatti effettuati vari controlli per la corretta ricezione e sequenzializzazione di ogni messaggio/pacchetto inviato. Per questo, a differenza di datagram, i circuiti virtuali necessitano di connessioni permanenti tra i nodi, con conseguente perdita in scalabilità dell'architettura. E' evidente che i due metodi hanno vantaggi e svantaggi complementari: i datagram offrono una maggiore efficienza e flessibilità di uso, al prezzo di una minore affidabilità. L'utilizzazione di uno o l'altro dei due metodi dipende principalmente dall'applicazione. Tali protocolli, tuttavia, non sono stati progettati per il calcolo distribuito su rete e quindi presentano alcuni problemi. Per esempio TCP/IP offre buone performance per il trasferimento di file, ma non per in applicazioni di calcolo parallelo. Di seguito si riportano indicativamente valori relativi a velocità di trasmissione con protocollo TCP/IP, in condizioni normali di lavoro:

- 3-5 Mbit/sec su LAN Ethernet (10 Mbit/sec);
- 15-20 Mbit/sec su LAN a fibra ottica (440 Mbit/sec).

Network Computing: qualche cenno storico...

Le prime esperienze di network computing effettuate con sistematicita' risalgono agli inizi del 1985 quando presso il Fermi National Laboratory (USA) si comincio' a sperimentare un cluster di elaboratori Digital MicroVax collegati via Ethernet, per analizzare enormi quantita' di dati derivanti da esperimenti di fisica delle alte energie. Il tipo di applicazione (oggi classificabile come "imbarazzantemente parallela[14] ") si prestava molto bene per quel tipo di architettura in quanto ogni MicroVax poteva elaborare parti indipendenti dell'immenso data set, operando in maniera asincrona rispetto agli

---

[14]Termine tradotto dall'inglese *Embarrassingly Parallel*, coniato dal noto ricercatore Geoffrey C. Fox (Syracuse University, USA)

altri e, inoltre, non esisteva la necessita' di un frequente interscambio di dati tra i processori.

Anche nel mondo della finanza furono effettuate significative esperienze di cluster computing tendenti ad utilizzare reti di workstation SUN per analizzare durante la notte i dati finanziari derivanti dalle transazioni diurne.

Intorno agli anni 86-88, la Apollo Computer introdusse un prodotto innovativo chiamato NCS (Network Computing System), che prevedeva sia il concetto di "distribuzione" che di "eterogeneita'" del calcolo; NCS era costituito da un interessante insieme di strumenti software che cercavano di rendere il piu' trasparente possibile all'utente la distribuzione di parti di una applicazione (tipicamente, subroutine), agli elaboratori connessi in rete.

Possiamo dire che l'era del cluster computing comincio' realmente quando la IBM comprese che l'uso di un insieme di workstation RS/6000 poteva costituire una concreta alternativa ai tradizionali mainframe multiprocessore dotati di capacita' vettoriali, che per molte applicazioni tecnico-scientifiche cominciavano ad incontrare qualche difficolta' nei confronti di alcune macchine concorrenti presenti sul mercato dello *High Performance Computing*. Nell'aprile del 1992, la IBM annuncio' il suo primo cluster basato su RS/6000. Da allora, molti sistemi analoghi cominciarono ad apparire, inizialmente presso laboratori e centri di ricerca.

Nell'ottobre 1992, la Convex e la Hewlett-Packard annunciarono la serie Meta*, composta da un minisupercalcolatore Convex serie C strettamente accoppiato ad un cluster di workstation HP 9000/735. Nello stesso periodo anche la HP annunciava una propria soluzione al cluster computing composta da una workstation front-end e da una serie (8-16) di HP 9000/735 e la IBM iniziava la commercializzazione

---

* vedi scheda tecnica allegata

della nuova serie IBM 9076 SP1*, una architettura scalabile composta da 8 sino a 64 processori RISC con sistema operativo AIX/6000 Vers. 3, in grado di fornire prestazioni di picco variabili da 1 a 8 GFlop/s.

Convex Meta Series	
Sistema	costituito da 1-8 CPU Convex C + 1-64 nodi HP PA-RISC 200 MFlop/s - 14 GFlop/s peak performance sino a 30 GBytes di memoria RAM
Nodo di elaborazione	CPU HP PA-7100 (99 MHz) 198 MFlop/s (32 bits) peak performance Sistema Operativo : ConvexOS 32-400 Mbytes di memoria RAM (Adattatori opzionali per Ethernet, FDDI e SMI (Shared Memory Interconnect)
Comunicazione	Con SMI latenza 500 microsecondi ampiezza di banda 14 MByte/s

IBM Scalable POWERparallel System (IBM 9076 SP1)	
Sistema	8-64 nodi 1-8 GFlop/s peak performance 0.5-16 GBytes di memoria RAM
Nodo di elaborazione	CPU RISC System/6000 (62.5 MHz) 125 MFlop/s peak performance Sistema Operativo: AIX/6000 Vers. 3 64-256 MBytes di memoria RAM 2 adattatori Ethernet Adattatori opzionali per FDDI, FCS, S/390 e High Performance Switch
Comunicazione	High PerformanceSwitch ampiezza di banda di 40Mbyte/s - latenza 500 nsec

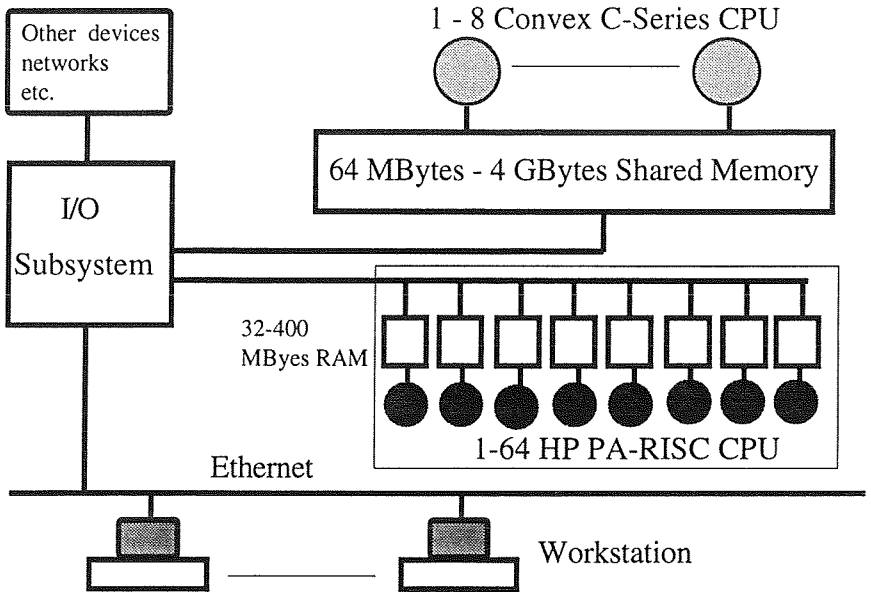


Fig. 4: Schema di un sistema Convex Meta Series

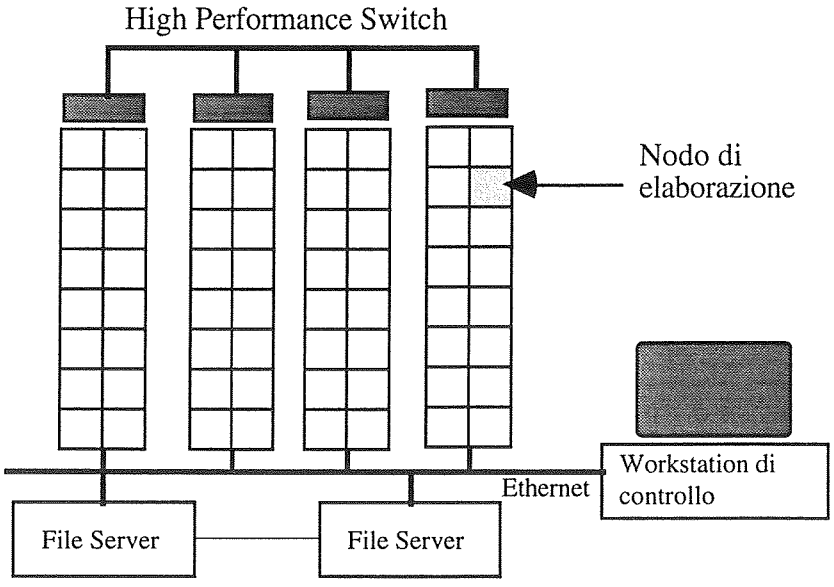


Fig. 5: Schema di un sistema parallelo IBM 9076 SP1

Digital Alpha AXP Farm	
Sistema	costituito da workstation (server) Alpha AXP
Nodo di elaborazione	DEC3000 Model 400-500 (133-160 MHz) 133-160 MFlop/s (64bit) peak performance Sistema Operativo: DEC OSF/1 64-512 Mbytes di memoria RAM Adattatore opzionale per Ethernet, FDDI
Comunicazione	GIGAswitch - crossbar switch che permette il collegamento di max 22 porte FDDI Ampiezza di banda di 3.6 Gigabit/s

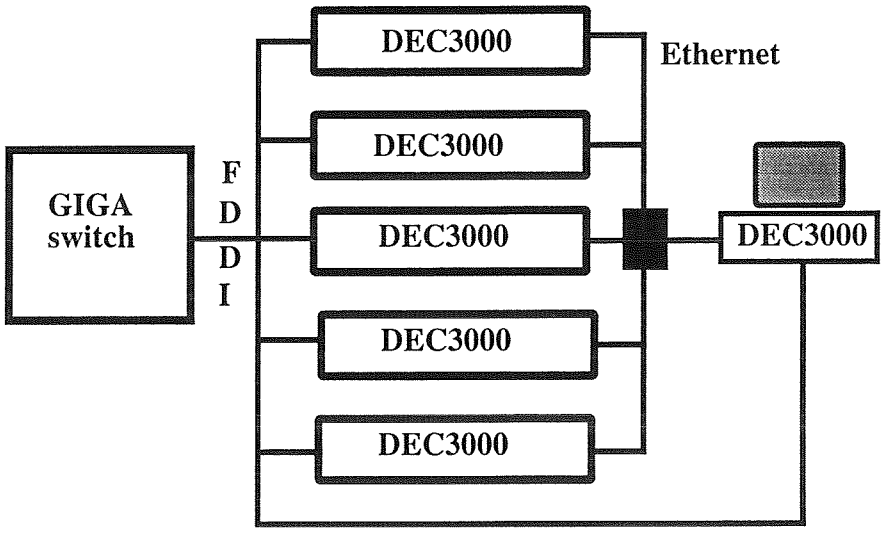


Fig. 6: Schema di cluster di workstation basato su Digital Alpha AXP

Agli inizi del 1993 anche la Digital ha proposto una propria soluzione di cluster computing, introducendo un "farm" di workstation basate sul nuovo chip Alpha AXP*. Il Farm, che e' sinonimo di "cluster di workstation", e' in grado di offrire potenze di picco dell'ordine dei GFlop/s. Il principale elemento di innovazione della proposta Digital e' rappresentato dal GIGAswitch, una apparecchiatura di interconnessione ad alta velocita' avente topologia *crossbar*. Le workstation del farm possono essere collegate al GIGAswitch mediante collegamenti FDDI e questo rende alquanto bassa la latenza di comunicazione tra i processori e consente una ampiezza di banda di banda complessiva di 3.6 Gbits/sec.

Software per Cluster di workstation

Per la gestione di un cluster di workstation esiste sul mercato una serie di prodotti software, alcuni venduti dai fornitori hardware, altri da societa' terze parti, altri di pubblico dominio. Tali software si dividono principalmente in due categorie di prodotti specializzati per la gestione del cluster e l'elaborazione parallela.

Classe	**Funzioni principali**
Gestione del Cluster	autorizzazione ed accesso al cluster, gestione delle code di elaborazione batch, bilanciamento dinamico del carico, monitoring del cluster, ecc.
Elaborazione Parallela	ambienti per la parallelizzazione, performance monitoring, profiling, debugging, librerie tecnico-scientifiche, ecc..

*vedi scheda tecnica allegata

Le Tabelle 2 e 3 mostrano la lista dei principali prodotti disponibili, rispettivamene, nella prima e seconda classe.

Software	**Produttore/Distributore (•) = Public-domain**
NQS e NQS/Exec	Sterling Software
ConvexNQS+	Software basato su NQS di proprieta' Convex
CODINE	Genias Software, Gmbh Germany
Task Broker	Software prodotto dalla Hewlett-Packard (per HP Serie 700 e 800) e dalla SAIC (per Sun SPARCstation).
UTOPIA	Platform Computing
NCLOGIN	The Cummings Group
Distributed Queuing System (DQS)	(•) Florida State University
Dynamical Network Queuing System	(•) McGill University
Load Balancer	Freedman Sharp and Associates
IBM LoadLever	IBM Corp.

Tab. 2: Software per la gestione del cluster

Ambienti di programmazione parallela per cluster

Negli ultimi anni sono stati realizzati vari strumenti per sviluppo, debugging e analisi delle prestazioni di applicazioni parallele su cluster di workstation. Essi forniscono un insieme di funzionalità per l'esplicitazione del parallelismo, il controllo dei processi, la comunicazione e la sincronizzazione. La differenza sostanziale tra un ambiente e l'altro è data dai paradigmi computazionali sui quali si basano, dal livello di funzionalità offerto e dalla portabilità che

assicurano. Alcuni di questi ambienti sono di pubblico dominio (per esempio PVM, P4), mentre altri, quali EXPRESS, Linda e ISIS sono prodotti commerciali.

Software	**Produttore/Distributore (•) = Public-domain**
PVM-HeNCE	(•) Oak Ridge National Labs, University of Tennessee, Emory University
ConvexPVM	PVM distribuito dalla Convex
Express	ParaSoft Corporation
Network Linda	Scientific Computing Associates
ISIS	ISIS Distributed Systems, Inc.
P4	(•) Argonne National Labs
PARMACS	(•) GMD, Bonn

Tab. 3: Software per la elaborazione parallela su cluster

In questo articolo viene posta particolare attenzione al software per l'elaborazione parallela ed in particolare ai due ambienti di programmazione PVM e Linda. La scelta di focalizzare l'attenzione e descrivere in dettaglio questi due ambienti deriva principalmente dal fatto che si tratta di due prodoti che, oltre ad essere rappresentativi di due differenti paradigmi computazionali, risultano, l'uno di pubblico dominio e l'altro disponibile commercialmente.

PVM

PVM (*Parallel Virtual Machine*) è uno strumento di pubblico dominio per la programmazione parallela di reti loosely-coupled di calcolatori eterogenei. Sviluppato a Oak Ridge National Laboratory dal gruppo *"Heterogeneous Network Project"* coordinato da Al Geist (ORNL) e Vaidy Sunderam (Emory University), PVM e' disponibile in formato

di codice sorgente.
PVM e' composto da una libreria di primitive per l'attivazione, la comunicazione e la sincronizzazione di processi paralleli e da un processo daemon[15]. Le primitive costituiscono l'interfaccia utente e debbono essere inserite all'interno di un programma Fortran o C per esplicitarne il parallelismo secondo il modello *message-passing*.

Il processo deamon, che costituisce il supporto a tempo d'esecuzione di PVM, deve essere attivo su tutti gli elaboratori della rete che formano la "macchina virtuale", cioe' l'insieme dei calcolatori collegati in rete che PVM tratta come un unico elaboratore a memoria distribuita. Gli elaboratori costituenti la macchina virtuale PVM possono avere caratteristiche architetturali differenti e possono essere sistemi uniprocessore, workstation, elaboratori vettoriali, multiprocessori e multicomputer.

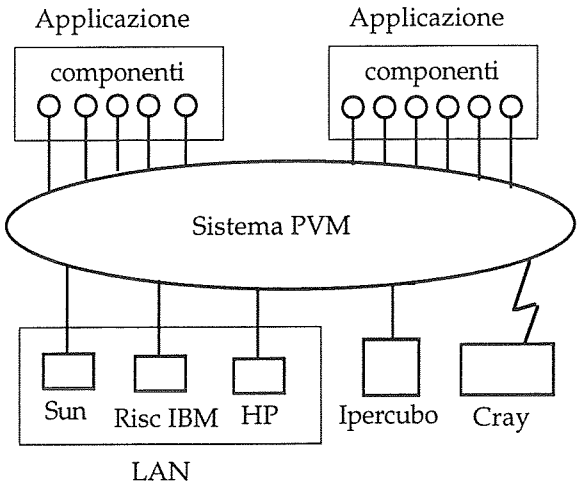


Fig. 7: Modello architetturale del sistema PVM.

---

[15] I procesi daemon rappresentano una classe di processi Unix che svolgono attivita' comuni di sistema come, ad esempio, la contabilita' dei login, la gestione delle code di stampa, i server di rete, ecc..

In figura 7 ed in Tab. 4 sono mostrati, rispettivamente, un esempio di macchina virtuale PVM e l'elenco degli elaboratori su cui PVM e' implemento. I vari elementi di elaborazione presenti nella macchina virtuale possono essere interconnessi tramite una o piu' reti anche differenti (*Ethernet, Token-ring*, ecc.).

Nome PVM	**Elaboratore**
AFX8	Alliant FX/8
ALFA	DEC Alpha
BAL	Sequent Balance
BFLY	BBN Butterfly TC2000
BSD386	80386/486 Unix box
CM2	Thinking MachinesCM2 Sun front-end
CM5	Thinking Machines CM-5
CNVX	Convex C-series
CNVXN	Convex C-series
CRAY	C-90,YMP, Cray-2
CRAYSMP	Cray-S-MP
DGAV	Data General Avion
HP300	HP-9000 model 300
HPPA	HP-9000 PA-RISC
KSR1	Kendall Square KSR-1
I860	Intel iPSC/860
IPSC2	Intel IPSC/2 386 host
NEXT	NeXT
PGON	Intel Paragon
PMAX	Dec/station 3100, 5000
RS6K	IBM/RS6000
RT	IBM RT
SGI	Silicon Graphics IRIS
SUN3	Sun 3
SUN4	Sun 4, SPARCstation
SYMM	Sequent Symmetry
TITN	Stardent Titan
UVAX	Dec MicroVAX

Tab 4: Elenco elaboratori su cui PVM e' eseguibile

Applicazione PVM

Una applicazione parallela PVM e' costituita da *componenti*. Una componente è, in generale, la versione compilata di un programma C o Fortran che contiene chiamate a primitive PVM. Il partizionamento dell'applicazione in componenti è compito del programmatore.

Ciscuna componente, a tempo d'esecuzione, verra' caricata ed eseguita su uno o piu' elaboratori della macchina virtuale. Poiche' PVM supporta un vasto insieme di architetture, ciascuna componente potra' essere progettata in modo da essere eseguita sull'elaboratore che meglio si adatta alle sue caratteristiche computazionali.

La Fig. 8 mostra un esempio di applicazione strutturata secondo quattro differenti componenti (A, B, C e D) cooperanti e illustra graficamente le loro interazioni. Tutte le componenti sono state attivate in una sola istanza, ad eccezione della D.

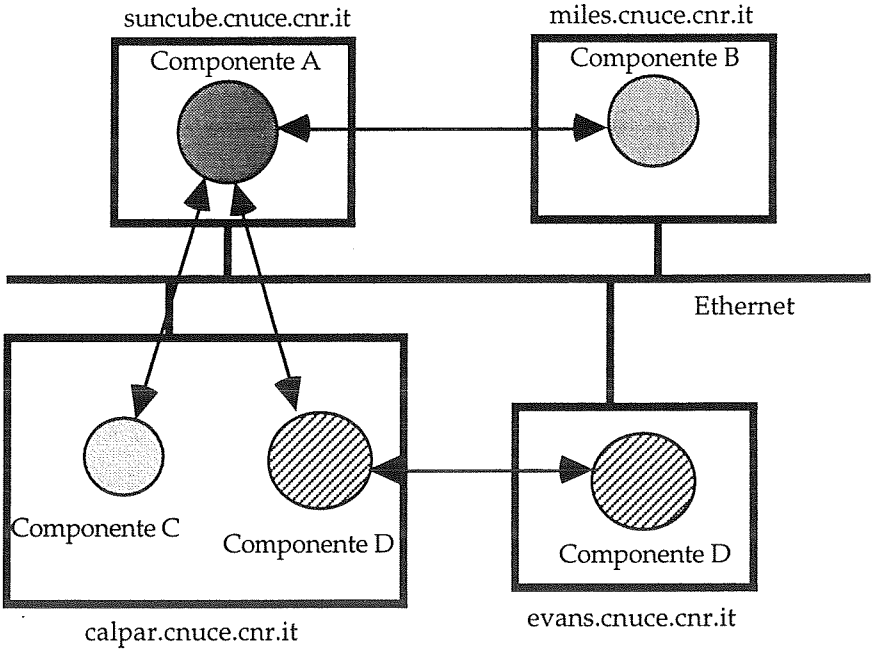


Fig. 8: Componenti di una applicazione.

L'attivazione di una componente sulle risorse della macchina virtuale puo' avvenire secondo tre differenti modalità: indipende dall'architettura, dipendente da una particolare classe architetturale e direttamente su una specifica risorsa della macchina virtuale.
La modalità "indipendente dall'architettura" prevede che l'istanza di una componente sia allocata automaticamente sul calcolatore che PVM ritiene più idoneo, in base ad opportuni algoritmi di selezione. La modalità "dipendente dall'architettura" permette all'utente di indicare specifiche classi architetturali sulle quali una componente dovra' essere eseguita. Secondo l'ultima modalita', l'utente dovra' indicare specificatamente l'indirizzo dell'elaboratore su cui eseguire la componente.

Definizione della macchina virtuale

PVM è attivato mandando in esecuzione il processo daemon da una qualsiasi risorsa di calcolo della macchina virtuale che assumera' funzione di *master*. Detto daemon, accedendo ad uno specifico file di configurazione (vedi Fig. 8), rileva i nomi e gli indirizzi degli elaboratori costituenti la macchina virtuale e su ciascuno di essi attiva un ulteriore processo daemon (deamon locale). Quest'ultimo è responsabile della creazione e gestione di tutti i processi PVM presenti sull'elaboratore stesso ed e' in grado di comunicare con tutti gli altri deamon. Il controllo è così completamente distribuito, cioe' i processi daemon hanno conoscenza "globale" di tutti processi attivi nella macchina virtuale, evitando cosi' eventuali colli di bottiglia ed aumentando la tolleranza ai guasti.
In Fig. 8 è mostrato un esempio di file di configurazione usato per creare un macchina virtuale. Ciascuna riga contiene, il nome della workstation che fara' parte della macchina virtuale PVM (suncube, evans, miles e calpar) ed eventuali opzioni ad essa associate. La

macchina virtuale sara' inizialmente composta da quattro workstation (*suncube, miles, calpar* e *evans*) connesse sulla rete cnuce.cnr.it; succesivamente, una ulteriore workstation (*sun4*) potra' essere inserita nel cluster.

```
suncube.cnuce.cnr.it   dx=~/pvmd
miles.cnuce.cnr.it
calpar.cnuce.cnr.it   ep=~/chol:~/traje
evans.cnuce.cnr.it
&  sun4.cnuce.cnr.it
```

Fig. 8: Esempio di file di configurazione

Ogni utente può installare il deamon PVM su un qualsiasi elaboratore della rete, purché vi abbia un *account* (*userid*). Macchine virtuali differenti possono condividere lo stesso cluster di workstation, e cio' permette quindi di avere in esecuzione contemporaneamente più applicazioni PVM.

E' responsabilita' del programmatore progettare applicazioni tolleranti ai guasti. PVM, prevede funzionalita' sia per rilevare automaticamente il malfunzionamento degli elaboratori della macchina virtuale che per modificarne dinamicamente la configurazione, ma non prevede alcun meccanismo di tolleranza ai guasti che automaticamente ripristini su altri elaboratori l'esecuzione dei processi interrotti a causa del malfuzionamento. E' comunque possibile per un'applicazione richiedere informazioni sullo stato degli elaboratori ed eventualmente, se necessario, aggiungerne di nuovi sia in sostituzione di quelli malfunzionanti che per aumentare la potenza elaborativa della macchina virtuale.

Creazione, schedulazione e controllo dei processi PVM

Un processo PVM è un'istanza di esecuzione di una componente, identificata mediante un valore numerico univoco (tid) che viene assegnato automaticamente dal daemon locale all'elaboratore sul quale il processo e' attivato. In Fig. 10 e' mostrato un esempio di attivazione di processi PVM. Ogni processo Unix per poter utilizzare una macchina virtuale deve essere registrato come processo PVM tramite la funzione **pvm_mytid** che deve precedere l'esecuzione di qualsiasi altra funzione PVM. Routine sono inoltre previste per la creazione (**pvm_spawn**) e terminazione (**pvm_kill**) dei processi.

Come gia' detto, l'attivazione di un processo puo' essere richiesta su uno specifico elaboratore della macchina virtuale, su un elaboratore di una determinata una classe architetturale oppure, si puo' lasciare la scelta dell'elaboratore all'arbitrio di PVM. Nel primo caso, nella funzione **pvm_spawn** deve essere specificato l'indirizzo di rete dell'elaboratore target, nel secondo caso, il nome PVM (vedi Tab. 4) prescelto e, nell'ultimo caso, nessun valore deve essere specificato. Nel secondo e terzo caso, la scelta della risorsa su cui il processo viene attivato è fatta da PVM in base ad euristiche che tengono conto del carico e delle prestazioni degli elaboratori presenti nella macchina virtuale.

Una volta identificato l'elaboratore *target*, il deamon locale invia la richiesta di attivazione al processo deamon dell'elaboratore remoto dove la componente deve essere attivata. Successivamente, il deamon remoto invia, in *broadcast*, la notifica dell'evento a tutti gli altri deamon, inserendo nel messaggio il *tid* del processo attivato.

I processi possono essere organizzati in entita' logiche dette *gruppi* la cui composizione puo' variare dinamicamente a tempo di esecuzione. Un gruppo e' visto come entita' atomica avente un proprio nome che deve essere specificato in ciascuna funzione PVM quale, ad esempio, la funzioni di broadcast.

```
main()
{       ........
/* Registrazione di un processo UNIX come processo PVM */
        my_tid = pvm_mytid();
        ........
/* Attivazione della componente "progr01", secondo la modalita' indipendente
dall'architettura */
        pvm_spawn ("progr01",NULL,0,"",1,tid);

/* Attivazione della componente "progr02", su una specifica classe di architetture
(RIOS= IBM RISC/6000 ) */
          pvm_spawn ("progr02",NULL,2,"RIOS",1,tid);

/* Attivazione della componente "progr03", su uno specifico elaboratore */
        pvm_spawn("progr03",NULL,1,"suncube.cunuce.cnr.it",1,tid);
        ........
/* Terminazione del processo PVM che ritorna processo UNIX */
        pvm_exit();
}
```

Fig. 10: Registrazione, attivazione e terminazione di processi PVM.

Cooperazione tra processi

Il modello di comunicazione tra processi supportato da PVM e' quello a scambio di messaggi con comunicazioni asincrone. Il modello prevede comunicazioni punto-a-punto e comunicazioni uno-a-molti (*broadcast*) e non prevede limitazioni sulla dimensione e numero dei messaggi scambiati. PVM prevede funzioni per l'invio e ricezione dei

messaggi. Nel caso di invio, il controllo ritorna al processo chiamante appena il buffer dei messaggi e' disponibile per la memorizzazione di un successivo messaggio senza tener conto dello stato del ricevente. La ricezione di un messaggio puo' essere "bloccante" o "non bloccante": nel primo caso, il controllo dell'esecuzione ritorna al processo chiamante solo dopo che il messaggio e' stato ricevuto, altrimenti esso viene sospeso in attesa dell'evento; nel secondo, il controllo ritorna immediatamente al processo chiamante con la segnalazione che il messaggio e' stato ricevuto oppure no.
Come mostrato nell'esempio di Fig. 11, l'invio di un messaggio prevede tre fasi: inizializzazione del buffer (**pvm_initsend**), "impacchettamento"[16] del messaggio (**pvm_pk***) e suo invio (**pvm_send**).
La funzione di invio prevede due parametri che indicano, rispettivamente, il processo destinatario ed il tipo associato al messaggio da inviare. La ricezione di un messaggio avviene in due fasi: ricevimento (**pvm_recv**) e "spacchettamento" (**pvm_upk***).

Controllo e debugging di applicazioni PVM
PVM prevede l'uso di una *console*, attiva sull'elaboratore *master*, sulla quale vengono inviati tutti i messaggi prodotti durante l'esecuzione dell'applicazione. Sono disponibili una serie di comandi PVM per monitorare lo stato della macchina virtuale e quello dei processi applicativi in esecuzione. Sulla console possono essere indirizzati anche messaggi usati dall'utente in fase di *debugging* "manuale" delle proprie applicazioni.

---

[16]L'elaborazione parallela in ambiente eterogeneo presenta, in generale, problemi di rappresentazione dei dati. PVM prevede routine di conversione di formato necessarie ogni qualvolta due elaboratori utilizzano differente rappresentazione dei dati.

```
main() /* processo mittente */
{
        .....
        msg_tipo = 11;
/*      inizializzazione del buffer dei messaggi */
        pvm_initsend();

/*      "impacchettamento" del messaggio nel buffer */
        pvm_pkstr ("stringa di dati");
        pvm_pkint (&var_int,1,1);
        pvm_pkfloat (&var_float,1,1);

/*      invio del messaggio */
        pvm_send (tid,msg_tipo);

        .....
}
main() /* processo ricevente */
{
        .....
        char *stringa;
/*      ricezione del messaggio nel buffer dei messaggi */
        pvm_recv(tid,11);

/*      "spacchettamento" del messaggio dal buffer */
        pvm_upkstr (stringa);
        pvm_upkint (&var_int,1,1);
        pvm_upkfloat (&var_float,1,1);

        .....
}
```

Fig. 11: Invio e ricezione di un messaggio in PVM.

HeNCE: un'interfaccia grafica per l'utente PVM

Un modo alternativo per la scrittura di applicazioni PVM è quelo di utilizzare il package HeNCE (Heterogeneous Network Concurrent Environment). HeNCE è costruito sopra PVM e prevede un'interfaccia

grafica per: specificare le componenti di una applicazione e le loro interazioni, configurare una macchina virtuale, compilare e distribuire il codice parallelo sulle varie risorse di calcolo, eseguire l'applicazione bilanciando dinamicamente il carico, raccogliere e visualizzare dati di traccia per il controllo della esecuzione.

Linda

La semplicita' e la pulizia formale sono le caratteristiche principali di Linda, il secondo linguaggio per la programmazione distribuita esaminato in questo articolo. Ideato presso la Yale University (New Haven, Connecticut, U.S.) da Nicholas Carriero e David Gelernter, Linda e' definito da sole quattro semplici operazioni che permettono di arricchire qualsiasi linguaggio di programmazione tradizionale quale, ad esempio, C o Fortran, con dei meccanismi completi e generali per la creazione e la cooperazione di processi paralleli.

Come spesso avviene negli ambienti accademici statunitensi, in seguito al successo del linguaggio, gli ideatori di Linda hanno costituito una societa', chiamata *Scientific Computing Associates* (SCA), incaricata della sua commercializzazione[17]. La SCA fornisce attualmente versioni C e Fortran di Linda per workstation IBM, HP, SUN, Digital e Silicon Graphics, per multicomputer Intel iPSC e nCUBE, e per molte architetture multiprocessore con memoria condivisa.

Il modello computazionale di Linda e' basato su uno spazio di memoria condivisa *(shared-memory model)*, chiamato spazio delle *n-uple (tuple space)* che contiene oggetti tipati *(n-uple)* acceduti non per indirizzo ma bensi' per associazione di contenuto tramite un procedimento di *pattern matching* tra tutte le *n-uple* contenute nello

---

[17]Linda in Italia e' commercializzato da LogPar s.r.l., via Paracelso, 18, Agrate Brianza, MI.

spazio e un *template* fornito dal processo che desidera accedervi. La presenza di uno spazio comune di memorizzazione, emulato abbastanza efficientemente a software nel caso in cui Linda sia implementato su una piattaforma hardware distribuita quale una rete di workstation o un multicomputer, rende molto flessibile e facile da utilizzare il linguaggio Linda. Buoni risultati possono infatti essere ottenuti tanto con applicazioni parallele basate su scambio di messaggi (lo scambio di un messaggio tra due o piu' processi puo' essere facilmente emulato con poche operazioni sullo spazio delle *n-uple*), che con applicazioni parallele che sfruttano lo spazio delle *n-uple* per la memorizzazione di dati condivisi acceduti se necessario in mutua esclusione (secondo il modello di programmazione tipico delle architetture multiprocessore con un unico spazio di indirizzamento).

Le n-uple e le regole di pattern maching di Linda

Una *n-upla* e' un insieme ordinato di un numero qualsiasi[18] di campi distinti, ognuno dei quali contiene un valore di un tipo ammesso dal linguaggio sequenziale utilizzato (un valore intero o reale, una stringa di caratteri, un vettore, una struttura, ecc.). In Linda una n-upla viene specificata separando con virgole una sequenza di espressioni. Ad esempio, la scrittura C:

("una n-upla", 10, 24.71, sqrt(4), v)

indica una n-upla di cinque campi aventi rispettivamente tipo char[10], int, float, double e, ad esempio, int[5] nel caso in cui la variabile *v* sia stata dichiarata come vettore di cinque elementi di tipo intero. Come accennato precedentemente, lo spazio delle *n-uple* di Linda e' una

---

[18] L'implementazione di Linda della Scientific Computing Associates restringe ad un massimo di sedici il numero di campi di una n-upla.

memoria associativa a cui i processi accedono mediante procedimento di *pattern matching* sul tipo e sul contenuto delle *n-uple* contenute in essa. Le operazioni Linda che cercano una n-upla all'interno del *tuple space* hanno come parametro un *template*, ovvero una sequenza di campi tipati ognuno dei quali puo' essere un campo "attuale" o "formale". Un campo "attuale" e' un valore (o meglio, una espressione la cui valutazione restituisce un valore) mentre un campo "formale" viene indicato con un identificatore di variabile preceduto dal carattere '**?**'. La scrittura:

(str10, 10, ?x, sqrt(4), ?w)

definisce un *template* in cui il primo, secondo e quarto campo sono campi "valore" mentre il terzo e quinto sono campi "formali". Le regole di *pattern matching* di Linda stabiliscono che si ha *matching* tra un template **s** e una n-upla **t** se e solo se:

1) il numero di campi di **t** e **s** e' lo stesso;

2) il tipo di ogni campo di **t** e il tipo di ogni campo ("valore" o "formale") nella stessa posizione di **s** e' lo stesso;

3) i campi "valore" di **s** sono uguali a quelli corrispondenti di **t**.

Dall'applicazione di queste tre regole alla *n-upla* e al *template* usati come esempio, si puo' facilmente dedurre che, avendo lo stesso numero di campi, si ha *matching* se e solo se: *str10* e' un vettore di caratteri con 10 posizioni a cui e' stata precedentemente assegnata la stringa "una n-upla", x e' una variabile di tipo reale e w e' un vettore di cinque elementi di tipo intero. Come vedremo piu' avanti l'effetto di un procedimento di *pattern matching* tra un template **s** e una *n-upla* **t** e' che i valori "attuali" di **t** vengono assegnati alle variabili

indicate nei campi "formali" di **s**.

All'interno dello spazio delle *n-uple* non sono presenti solamente *n-uple* contenenti dati (chiamate *n-uple* "passive") ma, dal punto di vista logico, vi sono anche i processi del programma parallelo (*n-uple* "attive"), che cooperano tra di loro depositando in esso e prelevando da esso *n-uple* di dati . Quando un processo termina l'esecuzione, esso si "fissa" infatti nel *tuple space* divenendo una *n-upla* passiva indistinguibile dalle altre, avente per valore di uno dei campi il risultato della valutazione della funzione eseguita dal processo terminato.

Da questa prima breve descrizione il funzionamento di Linda potra' forse apparire al lettore profano complesso e misterioso, ma garantiamo che qualsiasi buon programmatore, dopo aver visto la semantica delle operazioni Linda ed aver minimamente compreso il modello computazionale sottostante, sara' in grado di scrivere il suo primo programma parallelo Linda senza commettere gravi errori.

Le operazioni sullo spazio delle *n-uple*

Tutte le operazioni Linda agiscono sul *tuple space* permettendo di inserire in esso una nuova n-upla o di ricercarne una con determinate caratteristiche leggendone il contenuto e eventualmente eliminandola. Le due operazioni di generazione sono:

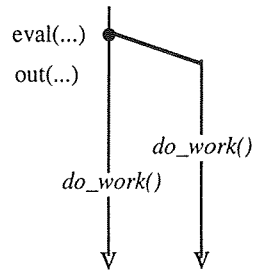
- **out (t)**: la *n-upla* **t** passata come parametro viene valutata all'interno del processo invocante e aggiunta allo spazio delle *n-uple*; il processo che la ha generata continua l'esecuzione;

- **eval (t)**: ha lo stesso funzionamento di **out**() con la differenza che **eval(t)** crea implicitamente un nuovo processo per valutare

ogni campo della *n-upla* **t**; una volta terminata la valutazione di tutti i campi di **t**, **t** viene depositata nello spazio delle *n-uple*; il processo che ha invocato la **eval**() continua immediatamente l'esecuzione senza attendere la terminazione della valutazione della *n-upla*.

Out fornisce quindi i meccanismi per mettere a disposizione di altri processi informazioni calcolate dal processo chiamante mentre **eval**, oltre a questo, permette di generare nuovi processi e di assegnare ad essi dei compiti che verranno svolti in parallelo. La differenza semantica tra le due operazioni e' sottile anche se sostanziale. Il frammento di codice C seguente invoca entrambe le operazioni con una *n-upla* analoga: nel primo caso (**eval**) la funzione *do_work()* viene eseguita da un nuovo processo appositamente creato, nel secondo (**out**) la stessa funzione e' invece valutata dal processo chiamante. L'effetto complessivo delle due istruzioni e' che due istanze della funzione C *do_work()* , il cui risultato sara' memorizzato all'interno dello spazio delle *n-uple*, vengono eseguite in parallelo da due processi distinti.

```
int do_work(), data[2];
.....
eval ("compute", do_work(data[0]), 0);
out  ("compute", do_work(data[1]), 1);
.....
```



Analizziamo ora le restanti operazioni Linda di lettura ed eliminazione di *n-uple* :

in (**s**) : una *n-upla* **t** che verifica il procedimento di *pattern matching* con il *template* **s** viene eliminata dallo spazio

delle *n-uple*; ai campi formali di **s** vengono assegnati i valori corrispondenti della *n-upla* **t**; se, al momento della chiamata, nello spazio delle *n-uple* non esiste alcuna *n-upla* che si accoppia ad **s** il processo si sospende, se invece esistono piu' *n-uple* ne viene scelta arbitrariamente ed eliminata una;

inp (s): e' la versione non bloccante di **in**(); se esiste una *n-upla* **t** che verifica il procedimento di *pattern matching* con il *template* **s**, **inp**() ha lo stesso funzionamento di **in**() e ritorna il valore **1** (*.true.* in Fortran), se invece non esiste alcuna *n-upla* che si accoppia ad **s**, **inp**() termina immediatamente restituendo il valore **0** (*.false.* in Fortran);

rd (s) : ha lo stesso funzionamento di **in**() con la differenza che la *n-upla* **t** selezionata non viene eliminata dal *tuple space;*

rdp (s): e' la versione non bloccante di **rd**().

Allo scopo di comprendere meglio il funzionamento delle operazioni Linda, consideriamo un esempio concreto quale la sincronizzazione di un insieme di processi. Desideriamo cioe' che un insieme di processi si pongano vicendevolmente in attesa uno dell'altro proseguendo contemporaneamente solo quando tutti hanno raggiunto il medesimo punto all'interno del loro programma.

```
barrier_init (int n)          wait_on_barrier()
{                             {
                              int val;
out("barrier", n-1);          in("barrier", ?val);
rd("barrier",0);              out("barrier", val-1);
                              rd("barrier", 0);
}                             }
```

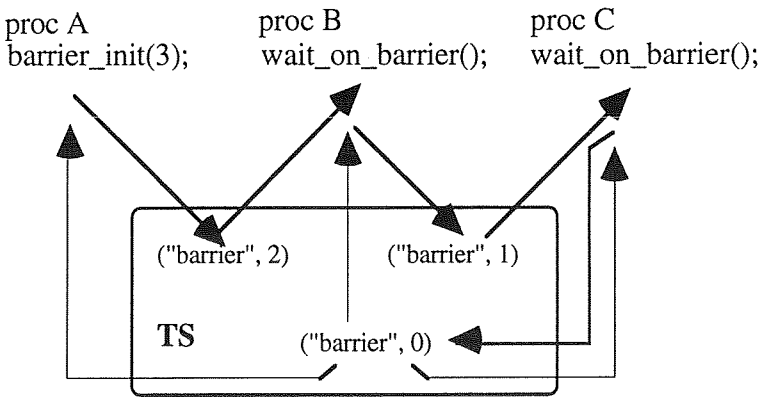
Questo tipo di sincronizzazione viene chiamata "barriera" e puo' essere realizzata in Linda con le due semplici funzioni riportate in seguito. La chiamata alla funzione *barrier_init()* deve essere effettuata da un solo processo che conosce il numero di processi tra cui la sincronizzazione verra' effettuata. La funzione deposita all'interno del *tuple space* una *n-upla* contenente la stringa di caratteri "barrier" e un valore intero pari al numero di processi che devono sincronizzarsi. Una volta depositata tale n-upla, il processo attende che nello spazio delle *n-uple* venga inserita da un altro processo una n-upla analoga il cui valore intero sia uguale a 0.
In maniera corrispondente tutti gli altri processi che devono effettuare la sincronizzazione invocano la funzione *wait_on_barrier()* che sostituisce nel *tuple space* l'unica *n-upla* "di tipo barriera" di volta in volta presente con una analoga avente il valore intero decrementato di uno. A questo punto i processi attendono anch'essi che nel *tuple space* venga inserita la *n-upla* "di tipo barriera" il cui valore intero sia uguale a 0 ad indicare che anche l'ultimo processo si e' sincronizzato con gli altri.
E' semplice convincersi, eventualmente con l'aiuto di carta e penna, che l'effetto del funzionamento descritto e' la sincronizzazione tra tutti i processi partecipanti indipendentemente dall'ordine temporale di esecuzione delle due funzioni. Supponiamo ad esempio che la "barriera" sia impiegata per sincronizzare i tre processi A, B e C, e che A, avente il compito di inizializzare la barriera invocando la funzione *barrier_init(3)*, giunga per primo al punto di sincronizzazione, seguito da B ed infine dal processo C. In questo caso il comportamento dei processi illustrato graficamente in figura e' il seguente:

1) A deposita la *n-upla* ("barrier", 2) e si sospende sulla operazione di lettura **rd**("barrier", 0);

2) B sostituisce la *n-upla* ("barrier", 2) con la *n-upla* ("barrier", 1) e si sospende a sua volta sulla operazione di lettura **rd**("barrier", 0);

3) C sostituisce la *n-upla* ("barrier", 1) con la *n-upla* ("barrier", 0) e di seguito esegue l'operazione di lettura **rd**("barrier", 0) immediatamente soddisfatta; contemporaneamente i processi A e B vengono risvegliati dalla presenza della *n-upla* desiderata e completano l'operazione di lettura;

4) tutti i processi escono contemporaneamente dalle funzioni invocate e proseguono la loro esecuzione.



Una versione parallela di "hello_world"

Chiunque abbia intrapreso lo studio del linguaggio C, si e' imbattuto in "hello_world", sicuramente il piu' classico e semplice esempio di programma C:

```
main()
{
printf("Hello, world. \n");
}
```

Vediamo quindi una versione parallela di "hello_world" implementata mediante Linda. Nell'esempio ogni processo in esecuzione su una differente workstation esegue il programma "hello_world" leggermente modificato (vedi la funzione C *hello()*) allo scopo di stampare sul terminale da cui il programma e' stato attivato, oltre al messaggio "Hello, world", anche il nome della workstation da cui tale messaggio proviene.

```
real_main(argc,argv)
int argc;
char *argv[];
{
int n_worker, j, hello();
n_worker = atoi(argv[1]);

for (j=1; j <= n_worker; j++)
    eval("worker", hello(j));

for (j=1; j <= n_worker; j++)
    in("worker", 0);

printf("hello_world is finished.\n");
}
```

```
hello(j)
int j;
{
char name[20];
gethostname(name, 20);
printf("Hello, world from process %d
running on workstation %s\n", j, name);
return(0);
}
```

Il programma richiede che il numero di processi attivati sia specificato nella linea di comando. Nel primo loop, mediante la funzione **eval**(), vengono attivati i processi Linda e ognuno di essi viene incaricato di eseguire la funzione *hello()*. Il programma principale (si noti che per

motivi implementativi la routine principale di un programma C-Linda deve avere nome *real_main* anziche' l'usuale *main*), attende poi nel secondo loop che tutti i processi attivati giungano a terminazione diventando *n-uple* passive del tipo ("worker", 0) che vengono immediatamente rimosse dallo spazio delle *n-uple*.

Il programma puo' essere innanzitutto provato su una singola workstation per verificarne il funzionamento: il pacchetto software della SCA comprende infatti un emulatore dell'ambiente Linda su singola workstation e uno strumento grafico per la visualizzazione dello spazio delle *n-uple* e l'analisi dei processi (*tuplescope*). Una volta assicurato il corretto funzionamento su singola workstation, per eseguire il programma realmente in parallelo su rete di workstation e' sufficiente ricompilare il sorgente con il compilatore network Linda **clc** e invocare il comando **ntsnet** che gestisce l'esecuzione di un programma Linda su rete.
Avremo quindi:

```
$ clc -o hello_world hello_world.cl
CLC (V2.5.2 Network version)
$ ntsnet hello_world 5
Hello, world from process 3 running on workstation mammolo
Hello, world from process 4 running on workstation sting
Hello, world from process 5 running on workstation mafalda
Hello, world from process 1 running on workstation miles
Hello, world from process 2 running on workstation seal1
Hello_world is finished.
$
```

Il comando **ntsnet**, per conoscere la configurazione della rete utilizzata dall'utente utilizza informazioni specificate in un apposito file. Tra queste informazioni devono essere indicati, ad esempio, i *login* e i *pathname* dell'utente sulle varie workstation, le soglie di carico entro le quali le workstation possono essere utilizzate dai programmi Linda e, nel caso in cui non tutte le workstation abbiano la stessa potenza, si

possono indicare le prestazioni relative di ogni singola macchina. La flessibilita' offerta dallo strumento **ntsnet** e' elevata ed assicura, se il file di configurazione e' stato ben costruito, che i processi Linda verranno eseguiti sulle workstation piu' potenti e meno cariche di lavoro, minimizzando cosi' il tempo totale di esecuzione.

V.S. Sunderam and G.A. Geist.
Network--Based Concurrent Computing on the PVM System.
Concurrency: Practice and Experience, 4(4):293--311, July 1992.

R. Baraglia, D. Laforenza, R. Perego
Programming a workstation cluster with PVM and Linda: a qualitative and quantitative comparison
Proc. AICA93 International Section, Parallel and distributed architectues and algorithms, pp. 101-114, September 1993, Gallipoli (LE).