WAMM (Wide Area Metacomputer Manager): una interfaccia visuale per la gestione di un metacalcolatore
Versione 1.0

Ranieri Baraglia, Gianluca Faieta, Marcello Formica, Domenico Laforenza

CNUCE - Institute of the Italian National Research Council
Via S.Maria, 36 - I56100 Pisa, Italy
Tel. +39-50-593111 - Fax +39-50-904052
email: R.Baraglia@cnuce.cnr.it, D.Laforenza@cnuce.cnr.it
meta@calpar.cnuce.cnr.it

Indice

1 Introduzione

Negli ultimi venti anni abbiamo assistito ad un notevole incremento delle prestazioni dei calcolatori, dovuto principalmente alla disponibilità di hardware più veloce e software più sofisticato. Nonostante ciò, in numerose aree della scienza e dell'ingegneria esistono problemi di dimensione e complessità tali da richiedere potenze computazionali sempre maggiori, almeno di svariati ordini di grandezza superiori a quella del più potente dei supercalcolatori esistenti. Per tentare di affrontare tali problemi, occorrerebbe concentrare nello stesso sito svariati supercalcolatori in modo da disporre della potenza complessiva necessaria. Questa soluzione è ovviamente improponibile sia per ragioni di praticità che di economicità.

Da qualche tempo a questa parte, in alcuni importanti ambienti di ricerca (perlopiù statunitensi) sono state avviate esperienze intese a sperimentare l'utilizzo cooperativo via rete di risorse computazionali distribuite geograficamente. Per indicare questo emergente approccio, sono apparsi in letteratura termini quali: Metacomputing [SC92], Heterogeneous Computing [KPSW93], Distributed Heterogeneous Supercomputing [FC90], Network Computing, ecc... Una delle principali ragioni alla base dell'introduzione delle reti di calcolatori è stata la necessità di rendere irrilevante il dove il ricercatore svolge fisicamente la propria attività, permettendogli di accedere agli strumenti di elaborazione, distribuiti geograficamente in vari siti, in maniera veloce e trasparente. L'evoluzione tecnologica e la sempre più capillare diffusione delle reti telematiche, nate inizialmente per permettere principalmente operazioni di *file transfer*, posta elettronica e poi anche di *remote login*, rendono oggi possibile il concretizzarsi di un ulteriore ed affascinante obiettivo: considerare le molteplici risorse in rete come un unico calcolatore, cioè, un metacalcolatore.

1.1 Metacalcolatore

Un metacalcolatore differisce notevolmente da un tipico elaboratore parallelo MIMD a memoria distribuita (esempio, Thinking Machine CM-5, nCUBE 2, IBM SP2, ecc.). Infatti, generalmente, quest'ultimo si compone di nodi di elaborazione *tightly-coupled* dello stesso tipo, dimensione o potenza, mentre in un metacalcolatore le risorse sono *loosely-coupled* ed eterogenee. Ciascuna di esse può svolgere una precisa funzione (calcolo, memorizzazione, rendering, ecc.) e, in questa visione, ciascuna risorsa potrebbe prendersi carico di un opportuno pezzo dell'applicazione, in funzione del paradigma computazionale al quale l'algoritmo si ispira e delle caratteristiche dell'architettura. Ad esempio, in un metacalcolatore comprendente, tra l'altro, una Connection Machine (CM-2) ed un cluster di workstation, una applicazione partizionabile, ad esempio, in due componenti, l'una di tipo *data parallel* e la seconda *coarse-grain task farm* sfrutterebbe naturalmente le caratteristiche degli elaboratori citati. L'obiettivo di realizzare un metacalcolatore risulta oggi concretamente perseguibile e può rappresentare una soluzione economicamente interessante per affrontare complessi problemi computazionali (non solo di natura tecnico-scientifica), in alternativa all'uso dei costosissimi supercalcolatori tradizionali. Lo studio delle problematiche e lo sviluppo delle tecnologie necessarie al consolidamento del metacomputing, visto come strumento di calcolo per uso generale, sono ancora allo stadio iniziale e richiedono ancora molti anni di ricerca e sperimentazione prima di diventare di dominio comune. Affinché il metacomputing possa diffondersi ed affermarsi è necessario proseguire nelle ricerche in diverse aree quali, ad esempio:

1. metodologie e strumenti per l'analisi, la parallelizzazione e distribuzione su metacomputer di una applicazione;
2. algoritmi di mapping e di bilanciamento del carico in ambiente eterogeneo;
3. sviluppo di interfacce user-friendly per la gestione e la programmazione di metacalcolatori;
4. fault-tolerance e sicurezza degli ambienti di metacomputing;
5. reti ad elevate prestazioni.

Che esista una certa convergenza nel considerare un metacalcolatore come una soluzione estremamente interessante è anche confermato dal notevole numero di lavori sull'argomento apparsi recentemente in letteratura, tanto che alcuni ricercatori concordano sul fatto che i prossimi anni costituiranno quelli del metacomputing.

2 Ambienti per metacomputing

La realizzazione di ambienti di metacomputing richiede la risoluzione di alcuni problemi di tipo hardware (reti di interconnessione, memorie di massa ad accesso parallelo) e software (linguaggi, ambienti di sviluppo, strumenti per la gestione delle risorse). Nonostante molti dei problemi hardware si avviino ad una soluzione, le questioni rimaste aperte sul fronte del software sono ancora molteplici.

Gli ambienti di sviluppo oggi disponibili prevedono generalmente dei meccanismi di gestione delle risorse del metacomputer, ma spesso non forniscono strumenti adeguati per la progettazione e la scrittura dei programmi. In mancanza di tali strumenti, il ciclo di progettazione del software per un metacomputer può comportare notevoli difficoltà. Tipicamente si possono individuare le seguenti fasi:

1. l'utente sviluppa i sorgenti su un nodo locale;
2. i sorgenti vengono trasferiti su *ogni* nodo costituente il metacomputer;
3. si esegue la compilazione su *tutti* i nodi;
4. se si verificano errori o si rendono necessarie modifiche, si rieseguono tutti i passi precedenti.

La compilazione su tutti i nodi è necessaria in quanto a priori non è possibile determinare su quali macchine saranno mandati in esecuzione i moduli che compongono l'applicazione, quando essa sarà eseguita sul metacalcolatore. Se non sono disponibili strumenti opportuni, l'utente deve effettuare "a mano" il trasferimento dei sorgenti e la relativa compilazione su tutti i nodi, e ripetere le operazioni ogni volta che occorre correggere anche il più piccolo errore. Avere a disposizione un insieme omogeneo di macchine non semplifica molto il problema: in questo caso i sorgenti possono essere compilati anche su un solo nodo, ma i file eseguibili prodotti devono sempre essere trasferiti su tutte le macchine. Perciò, già con pochi nodi, un metodo per automatizzare queste attività diventa utile, se non indispensabile.

Alcuni ambienti per il metacomputing mettono a disposizione programmi per controllare certi aspetti di configurazione e gestione della macchina virtuale, quali l'attivazione, l'inserimento o la rimozione dei nodi, ecc. In certi casi, tuttavia, si sente la necessità di strumenti di gestione più semplici da usare, soprattutto quando si deve lavorare con metacomputer "grandi", con decine di nodi.

Per alleviare questi problemi, abbiamo realizzato WAMM (Wide Area Metacomputer Manager), una interfaccia grafica basata su PVM [Sun90, BDG+93, BDG+94a, Sun92, BDGS93, GBD+94] e OSF/Motif [Bra92] che consente di semplificare le operazioni normalmente svolte per programmare e utilizzare applicazioni per metacomputer, nonché per la gestione della macchina virtuale parallela.

3 Caratteristiche generali

In questa sezione vengono descritte le "linee guida" che abbiamo seguito nel progetto dell'interfaccia e le principali caratteristiche del pacchetto sviluppato.

3.1 Facilità d'uso

L'obiettivo principale dell'interfaccia deve essere la semplificazione dell'uso di un metacomputer. Per ottenere questo risultato è importante fornire all'utente una visione d'insieme del sistema, specialmente se si compone di molti nodi situati in laboratori diversi. Al tempo stesso dovrebbe essere possibile identificare facilmente le singole risorse. Anche se una semplice lista di indirizzi di rete delle macchine costituisce forse il metodo più veloce per individuare un particolare nodo ed accedervi, è bene raggruppare le macchine secondo un preciso criterio, in modo da facilitare all'utente l'*esplorazione* delle risorse disponibili in rete.

È fondamentale, inoltre, fare in modo che l'utente possa lavorare principalmente sul nodo locale; le operazioni da compiere su quelli remoti dovrebbero essere eseguite automaticamente. In questo modo lo sviluppo di software per metacomputer non comporterà l'uso di strumenti molto diversi da quelli "classici" (editor, `make`) usati per la scrittura e la messa a punto dei programmi sequenziali; l'impatto con un nuovo ambiente di programmazione risulterà meno difficoltoso.

La semplificazione dell'uso del metacomputer non può avvenire se lo strumento da impiegare non è a sua volta facile e intuitivo. È ormai noto come le interfacce di tipo grafico (*GUI - Graphical User Interface*) abbiano conquistato il favore degli utenti di elaboratori. Pertanto abbiamo ritenuto opportuno realizzare la nostra interfaccia come programma X11, lasciando che l'utente acceda alle sue funzionalità attraverso finestre, menu, icone. Ciò impone l'uso di terminali grafici, ma risparmia all'utente la necessità di imparare nuovi comandi, combinazioni di tasti, ecc.

3.2 Controllo del sistema

Lavorando con un metacomputer, specialmente se si fa uso di un ambiente di programmazione a basso livello come PVM, può essere difficile controllare le operazioni che avvengono sui nodi remoti; la sensazione di precarietà che spesso si ricava può scoraggiare gli utenti meno esperti.

Un'interfaccia per la programmazione e l'uso di un metacalcolatore dovrebbe offrire all'utente il maggior numero possibile di informazioni e il pieno controllo su quanto accade nel sistema. Ad esempio, l'utente non dovrebbe mai rimanere nella situazione di non sapere cosa sta accadendo su un determinato nodo. Nel caso in cui si verifichino problemi, questi devono essere comunicati con messaggi completi e non con arcani codici di errore. Se il problema è tale da impedire l'ulteriore uso dell'interfaccia, si dovrebbe effettuare una chiusura ordinata del programma.

3.3 Gestione della macchina virtuale

L'interfaccia deve prevedere un'insieme di funzioni "base" per la gestione della macchina virtuale (aggiunta/rimozione di nodi; controllo dello stato di uno o più nodi; creazione e distruzione della macchina virtuale). In sostanza si tratta di implementare le principali funzioni della console PVM.

3.4 Gestione dei processi

Anche in questo caso le primitive da implementare sono quelle messe a disposizione dalla console di PVM. Deve essere possibile avviare processi; l'interfaccia deve consentire l'uso di tutti i parametri e i flag di attivazione utilizzabili in PVM. Inoltre, l'utente deve poter ridirigire l'output dei task verso l'interfaccia, in modo da controllare in "tempo reale" il comportamento dei programmi.

3.5 Comandi remoti

Avendo a disposizione più macchine, capita molto spesso di dover effettuare sessioni di lavoro su host remoti; anche quando non è necessario aprire una sessione si devono poter eseguire comandi remoti (per esempio, `uptime` o `xload` per conoscere il carico della macchina). L'utilizzo di strumenti UNIX quali `rsh` per eseguire un comando su un host remoto è abbastanza scomodo, dunque l'interfaccia dovrebbe consentire una sua semplificazione. Un'ulteriore agevolazione si deve avere per i programmi X11. Quando si avvia un programma X11 su un host remoto, le finestre devono essere visualizzate sul terminale grafico locale, che deve essere "autorizzato" ad accettarle. L'abilitazione si ottiene con il comando `xhost`, ma dovrebbe essere automatizzata per i programmi X11 avviati dall'interfaccia.

3.6 Compilazione remota

Una delle funzionalità più importanti dell'interfaccia deve essere la possibilità di effettuare la compilazione di un programma su più macchine remote. Una volta definiti la directory locale contenente i sorgenti, il `Makefile` da impiegare e gli host su cui il programma deve essere compilato, il resto dovrebbe essere gestito completamente dall'interfaccia. Sia il trasferimento dei sorgenti su tutte le macchine che la compilazione vera e propria non dovrebbero richiedere interventi manuali che, oltre a risultare costosi, sono spesso fonte di errori.

La compilazione remota è un'operazione abbastanza complessa: è importante che chi usa l'interfaccia possa seguirne l'andamento passo dopo passo, ed eventualmente bloccarla in qualsiasi momento. Affinché l'utente possa sentirsi a proprio agio con uno strumento automatico, è opportuno eseguire tutte le operazioni in modo "ordinato": per esempio, i file temporanei o vecchi, creati sui file system delle macchine remote per effetto della compilazione, devono essere cancellati in modo trasparente.

3.7 Configurabilità

L'interfaccia deve costituire uno strumento configurabile, in grado di adattarsi a qualsiasi numero e dislocazione di macchine. Pertanto la configurazione del metacomputer non deve essere *cablata* nel programma, ma va specificata attraverso un file esterno.

Per modificare un qualsiasi elemento grafico dell'interfaccia (colori, dimensioni delle finestre, font, ecc.) si dovrebbe usare la tecnica standard dei *resource file*, usata per tutti i programmi X11. Questo tipo di modifiche non richiede una nuova compilazione dell'interfaccia; inoltre, usando il programma X11 `editres` è possibile modificare "al volo" un elemento anche senza scrivere un resource file.

Infine, l'interfaccia non dovrebbe imporre alcun vincolo sul numero di nodi e reti presenti nel sistema, nè sul loro tipo o sulla loro dislocazione.

4 WAMM

In base agli obiettivi descritti in precedenza è stato sviluppato WAMM, un prototipo di interfaccia per il metacomputing. In questa sezione viene presentata una descrizione generale del programma; successivamente saranno discussi i dettagli implementativi.

4.1 Configurazione

Per poter utilizzare l'interfaccia, l'utente deve preparare un file di configurazione contenente la descrizione dei nodi che possono essere inseriti nella macchina virtuale. In sostanza si tratta di tutte le macchine che sono accessibili all'utente e hanno PVM installato. Questa operazione deve essere compiuta solo la prima volta che si usa l'interfaccia.

Il file di configurazione, del quale viene ora mostrato un esempio, è scritto in un semplice linguaggio dichiarativo:

```
WAN toscana {
    TITLE "WAN Toscana"
    PICT toscana.xpm
    LAN arcetri 235 130
    MAN pisa 70 155
}

LAN arcetri {
    TITLE "Arcetri"
    HOST theseus
}

HOST theseus {
    ADDRESS theseus.arcetri.astro.it
    PICT Indigo2.xpm
    ARCH HPPA
    INFO "HP 9000/735"
```

Figura 1: WAMM

```
}

MAN pisa {
    TITLE "MAN Pisa"
    PICT pisa.xpm
    LAN cnuce 200 100
    LAN sns 280 55
}

LAN cnuce {
    TITLE Cnuce
    HOST calpar
    HOST miles
}

LAN sns {
    TITLE Normale
    HOST cibs
}

HOST calpar {
    ADDRESS calpar
    PICT HP715.xpm
    ARCH HPPA
}

HOST miles {
    ADDRESS miles.cnuce.cnr.it
    PICT Sun12.xpm
    ARCH SUN4
}

HOST cibs {
    ADDRESS cibs.sns.it
    PICT IBM41T.xpm
    ARCH AIX
}
```

Il file descrive una rete geografica di nome `toscana`. La rete si compone di una piccola LAN ad Arcetri, contenente il solo host `theseus`, e di una MAN a Pisa, comprendente le reti locali `cnuce` e `sns`. La prima include due host, `calpar` e `miles`, la seconda la macchina `cibs`.

Come si può notare, la rete viene descritta seguendo una struttura ad albero. La radice è la WAN, la rete geografica che ingloba tutti gli host. La WAN ha per figli reti metropolitane (MAN) o locali (LAN); una rete MAN può contenere solo reti locali, mentre le LAN racchiudono solo gli host, le foglie dell'albero.

Per ogni struttura dichiarata si possono usare varie direttive, molte delle quali sono opzionali e prevedono valori di default. Ogni nodo dell'albero (rete oppure host) può avere una direttiva `PICT`, usata per associare alla struttura un disegno; tipicamente per le reti vengono usate piantine geografiche che indicano la dislocazione delle risorse, per gli host icone che rappresentano il tipo di architettura.

Un esempio di descrizione "ricca" di un host è il seguente:

```
HOST theseus {
    ADDRESS theseus.arcetri.astro.it   # indirizzo dell'host
    PICT HP.xpm                        # icona
    ARCH HPPA                          # tipo di architettura
    INFO "HP 9000/735"                 # informazioni varie
    OPTIONS "& so=pw"                  # opzioni PVM
    XCMD "XTerm" "xterm"
    XCMD "VUEPad" "vuepad"             # comandi remoti
    XCMD "XLoad" "xload"
    CMD "Uptime" "uptime"
}
```

In questo caso si tratta di una workstation `HP 9000/735`; il tipo di architettura è `HPPA` (vengono usati gli stessi nomi adoperati da PVM). Per l'inserimento del nodo in PVM si devono seguire particolari modalità (`& so=pw` sono opzioni proprie del PVM). L'utente può eseguire sul nodo, direttamente dall'interfaccia, i comandi `xterm`, `vuepad`, `xload` e `uptime`. Per esempio, usando `xterm` può collegarsi direttamente alla macchina: il programma xterm gira sul nodo remoto ma l'utente riceve la finestra sul terminale locale e può usarla per inserire i comandi.

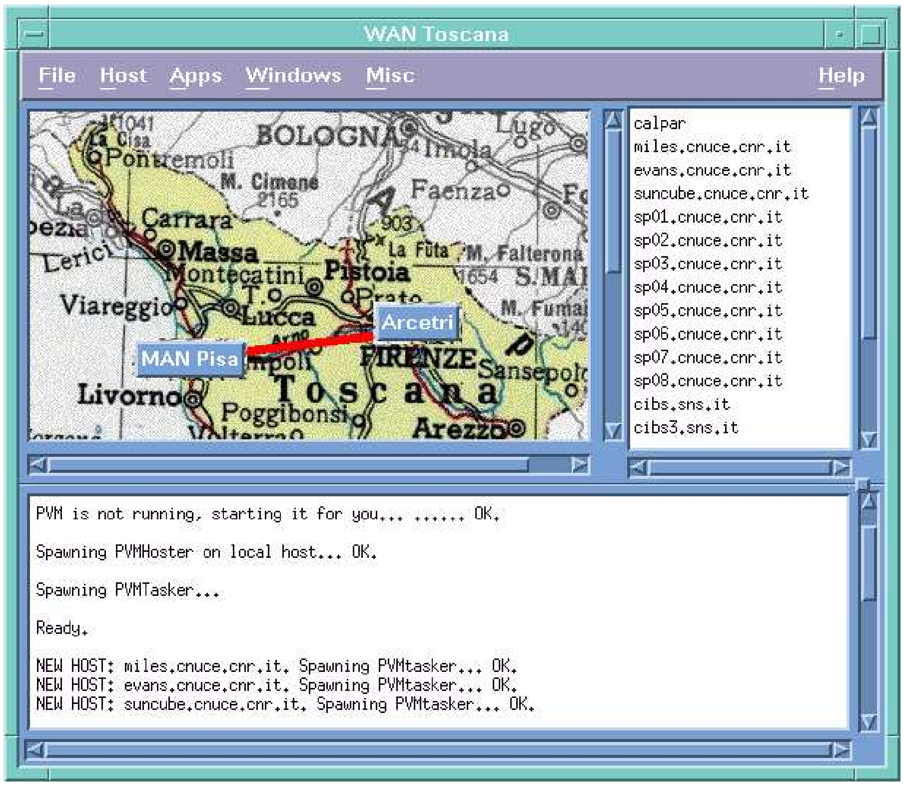

Figura 2: WAMM, finestra WAN iniziale

4.2 Attivazione

L'interfaccia viene mandata in esecuzione con il comando:

```
wamm <file di configurazione>
```

Non è necessario che il PVM sia già stato attivato: se la macchina virtuale non esiste ancora, il programma provvede a crearla in base al contenuto del file di configurazione. In particolare, vengono inseriti nel metacomputer tutti i nodi dichiarati nel file e privi dell'opzione **&**, usata anche negli *hostfile* standard di PVM per indicare che un host non deve essere immediatamente aggiunto alla macchina virtuale. All'utente viene mostrata la finestra "base" corrispondente alla WAN (fig. 2).

4.3 Finestre

WAMM visualizza le informazioni relative alle reti (WAN, MAN o LAN) in finestre separate, una per rete. Gli host vengono mostrati all'interno della finestra della LAN a cui appartengono.

La finestra per la WAN è suddivisa in tre parti (fig. 2). Nel riquadro in alto a sinistra viene mostrata la cartina indicata nel file di configurazione. Per ogni sottorete è presente un *bottone*: selezionandolo, l'utente può aprire le finestre corrispondenti (fig. 1).

A destra sono elencati tutti gli host dichiarati nel file di configurazione. La lista ha vari scopi: l'utente può accedere rapidamente ad un host facendo un doppio click con il mouse sul nome della macchina, senza dover passare prima per le varie sottoreti. Selezionando invece un gruppo di host, si possono richiamare da menu varie operazioni:

- inserimento degli host in PVM;
- rimozione degli host dal PVM;
- controllo dello stato degli host;
- compilazione.

In basso vengono mostrati tutti i messaggi prodotti da WAMM. In figura 2 si possono notare le informazioni scritte al momento dell'attivazione dell'interfaccia: in questo caso il PVM è stato avviato automaticamente e sono stati inseriti nella macchina virtuale vari nodi, tra cui `miles.cnuce.cnr.it`, `evans.cnuce.cnr.it` e `suncube.cnuce.cnr.it`.

Le sottoreti MAN vengono mostrate usando lo stesso tipo di finestra; l'unica differenza riguarda la lista degli host, che in questo caso comprende solo i nodi appartenenti alla MAN.

Per le reti locali si usa un'organizzazione della finestra differente (cfr. fig. 3). La finestra "riproduce" un segmento di rete Ethernet con relativi host. Per ogni nodo sono mostrati l'icona, lo stato attuale (`PVM` indica che il nodo appartiene alla macchina virtuale), il tipo di architettura e le altre informazioni specificate nel file di configurazione. Ogni icona possiede un menu *popup*, attivabile con il tasto destro del mouse, attraverso il quale si può cambiare lo stato del nodo (aggiungere o togliere da PVM), far partire una compilazione o eseguire uno dei comandi remoti indicati nel file. È ancora possibile effettuare le operazioni base su gruppi di host, selezionandone più

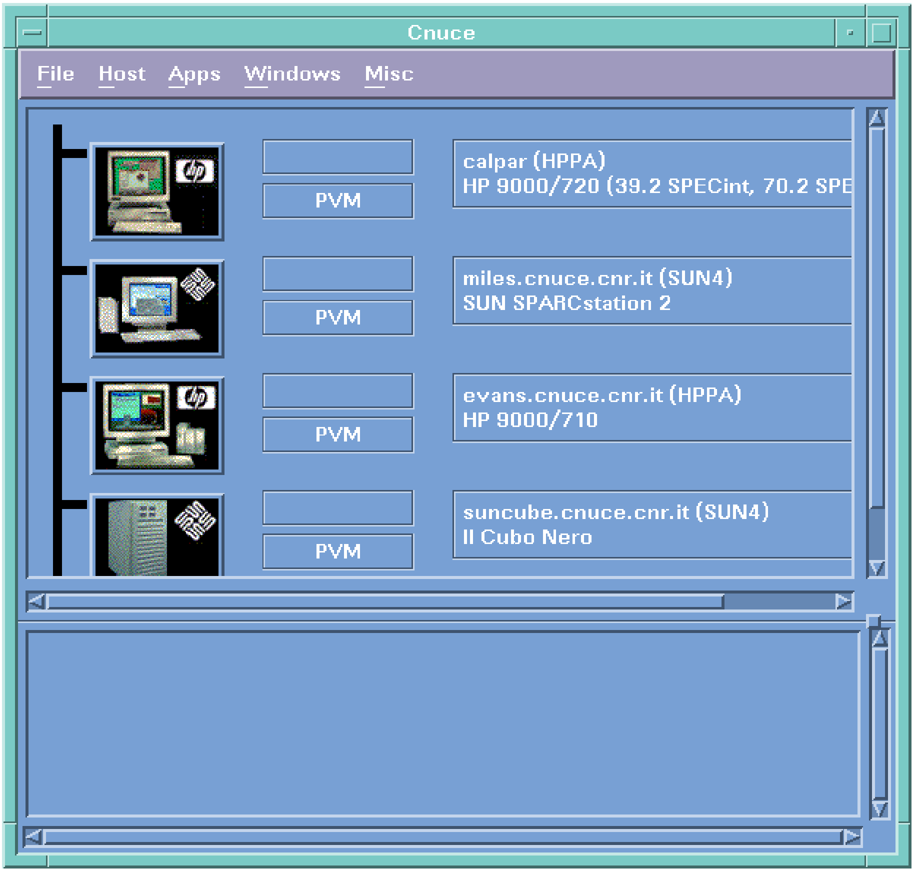

Figura 3: WAMM, finestra LAN

di uno e richiamando l'operazione dal menu della finestra. In tutti i casi i risultati sono stampati nell'apposita area per i messaggi, in basso nella finestra.

4.4 Compilazione

La compilazione di un programma viene gestita interamente dall'interfaccia: l'utente deve solo selezionare gli host su cui vuole effettuare la compilazione e richiamare la voce `Make` dal menu `Apps` (fig. 2). Attraverso una finestra di dialogo si possono specificare la directory *locale* che contiene i sorgenti e il `Makefile`, nonché eventuali parametri per il comando `make`. Non viene fatta alcuna ipotesi sul tipo dei file sorgenti da compilare: non è necessario che facciano uso delle funzioni PVM e possono essere scritti in un qualsiasi linguaggio.

La compilazione di un'applicazione richiede alcune fasi preparatorie, svolte dall'interfaccia. Nell'ordine:

1. tutti i file sorgenti vengono racchiusi in un unico file, nel formato standard `tar` usato su UNIX;

2. il file prodotto viene compresso usando il comando `compress`;

3. su ogni nodo selezionato viene fatto partire un task PVM che si occuperà della compilazione (`PVMMaker`); a ogni task viene inviato il file compresso.

A questo punto il lavoro dell'interfaccia termina; il resto viene eseguito contemporaneamente da tutti i PVMMaker, ciascuno sul proprio nodo:

1. il task crea una directory di lavoro temporanea, all'interno della home directory dell'utente;

2. il file compresso viene ricevuto e salvato all'interno della nuova directory; viene quindi espanso. I sorgenti vengono estratti;

3. viene eseguito il comando `make`.

Al termine della compilazione la directory di lavoro *non* viene distrutta (sarà fatto in caso di una successiva compilazione). In questo modo l'utente, se lo desidera, potrà collegarsi all'host, modificare eventualmente il codice sorgente ed eseguire manualmente una nuova compilazione sugli stessi file. Ogni PVMMaker informa l'interfaccia di tutte le operazioni compiute. I messaggi ricevuti vengono stampati in una finestra, utile per controllare l'andamento della compilazione. Nell'esempio di figura 4 la compilazione è stata avviata su sette macchine. Per ciascuna si può individuare il passo in corso nel momento in cui è stata "fissata" l'immagine. Per esempio, `astro.sns.it` ha ricevuto la propria copia della directory e la sta espandendo, mentre `calpar` ha già concluso con successo la compilazione.

Selezionando uno o più host nella finestra è possibile vedere i messaggi di output e gli eventuali errori prodotti dal `make` e dal compilatore, anche prima

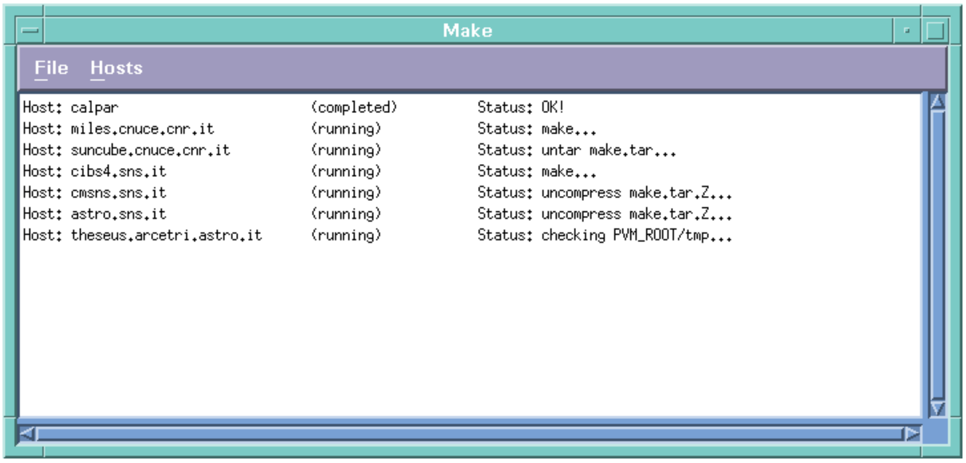

Figura 4: WAMM, finestra di controllo per la compilazione

che la compilazione finisca. Si può fermare il `make` in qualsiasi momento, usando un comando del menu. È opportuno notare che l'eventuale fallimento di un nodo (per esempio per errori nei sorgenti) non influenza in alcun modo l'attività in corso sugli altri.

Se la compilazione termina con successo, lo stesso `Makefile` usato per guidarla può copiare il file eseguibile prodotto all'interno della directory `$PVM_ROOT/bin/$PVM_ARCH`, usata dal PVM come "deposito" per i programmi da eseguire. Questa operazione sarà eseguita su ciascun nodo.

4.5 Task

L'interfaccia consente di avviare e controllare task PVM. Per eseguire un programma si seleziona la voce `Spawn` dal menu `Apps` (fig. 2). Viene aperta una finestra di dialogo che consente l'inserimento dei parametri usati da PVM per l'esecuzione dei task (fig. 5). Si possono specificare:

- il nome del programma;
- eventuali argomenti da passare al programma;
- il numero di copie da attivare;

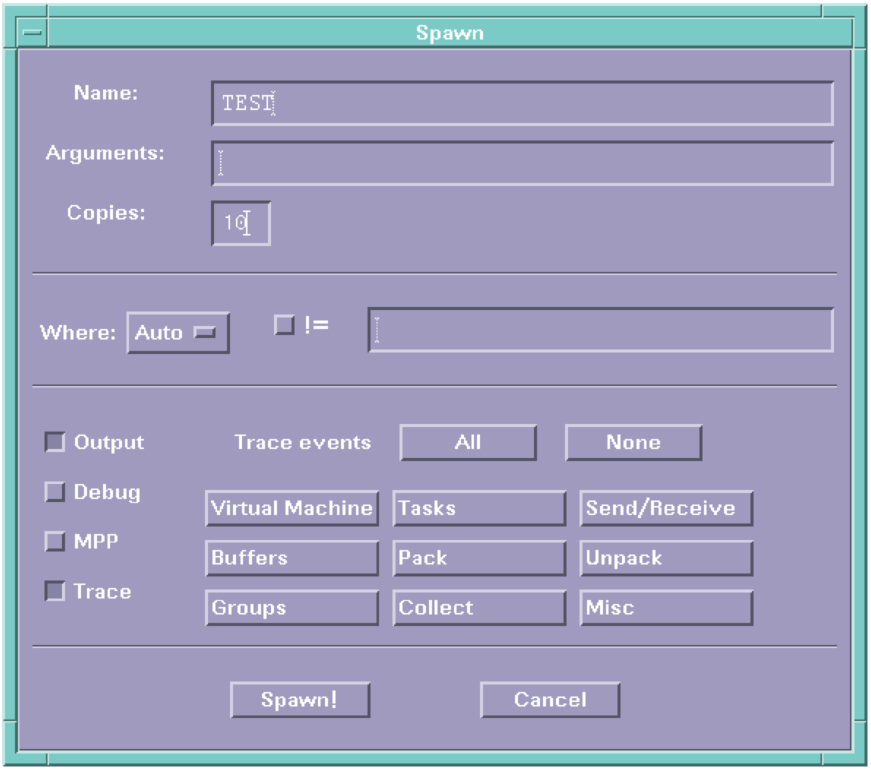

Figura 5: WAMM, richiesta di esecuzione di un programma PVM

- le modalità da seguire per avviare l'esecuzione delle copie: specificando `Host` tutti i task vengono attivati su un'unica macchina (di cui va indicato l'indirizzo); indicando `Arch` PVM sceglie solo macchine con una particolare architettura; infine con `Auto` la scelta dei nodi su cui far partire le copie viene lasciata interamente al PVM;
- vari flag propri del PVM (`Debug`, `Trace`, `MPP`);
- l'eventuale reindirizzamento dell'output del programma verso l'interfaccia (flag `Output`);
- gli eventi da registrare nel caso in cui venga abilitata l'opzione `Trace`.

Selezionando `Spawn` i nuovi task vengono avviati. Dal menu `Windows` (fig. 2) si può aprire una finestra di controllo contenente i dati di tutti i task PVM

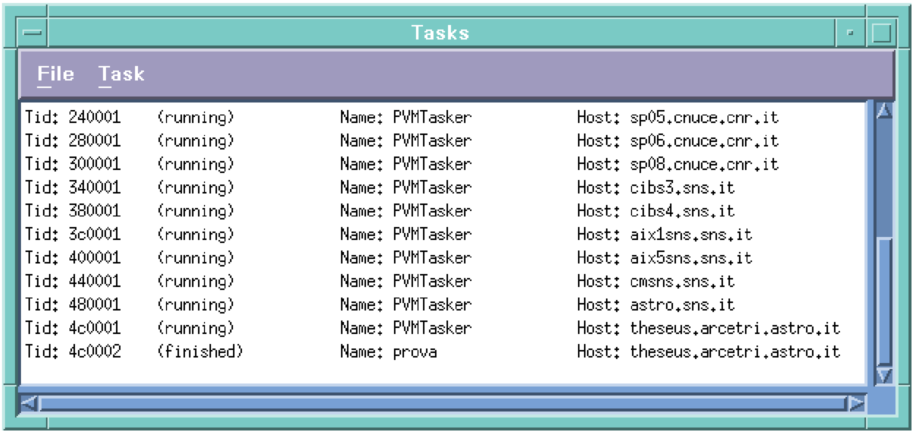

Figura 6: WAMM, finestra di controllo dei task PVM

in esecuzione nella macchina virtuale (fig. 6). I dati sui task sono aggiornati automaticamente: se un task termina, il suo stato viene cambiato; i nuovi task che compaiono nel sistema vengono aggiunti alla lista, anche se non sono stati attivati dall'interfaccia. Per i processi avviati con l'opzione `Output` è possibile vedere, in finestre separate, i messaggi di output prodotti; si possono anche salvare su file. Se le finestre di output sono aperte, i nuovi messaggi provenienti dai task vengono immediatamente visualizzati.

Per tutti i task, compresi quelli non mandati in esecuzione dal programma, sono possibili le operazioni di `Kill` (distruzione task) e `Signal` (invio di segnali specifici), usando il menu `Task` (fig. 6).

5 Implementazione

In questa sezione vengono discussi gli aspetti più importanti dell'implementazione e le scelte di progetto effettuate. In alcuni punti si farà riferimento a concetti e funzionalità propri del sistema operativo UNIX e dell'ambiente PVM; per una loro descrizione approfondita si possono consultare [Hah93] e [BDG+94a].

5.1 Struttura del programma

WAMM è scritto in C; per la compilazione sono richieste, oltre a PVM, le librerie OSF/Motif e xpm[1].

Ogni funzione complessa di WAMM è realizzata da un modulo indipendente; i moduli sono agganciati in fase di compilazione. Questo tipo di struttura, utile per tutti i programmi complessi, facilita le modifiche al codice e l'inserimento di nuove funzionalità. I moduli si possono suddividere in tre livelli:

1. Moduli applicativi. Sono moduli ad "alto livello" che implementano l'attivazione dei task, il loro controllo, la compilazione;

2. Moduli grafici. Comprendono tutte le funzioni necessarie per realizzare l'interfaccia grafica del programma;

3. Moduli di rete. Sono moduli di controllo che fanno da interfaccia tra l'applicazione e la macchina virtuale sottostante.

Il programma è completamente guidato da eventi: terminata la fase di inizializzazione dei propri moduli interni e delle relative strutture dati, si sospende, in attesa di messaggi provenienti dall'ambiente PVM o dall'utente (per esempio, la terminazione di un task attivo o la selezione di un bottone in una finestra). È una modalità di funzionamento tipica dei programmi X11.

5.2 Controllo degli host

Durante la fase di inizializzazione il programma si configura come task PVM, in modo da poter sfruttare tutte le funzioni di controllo della macchina virtuale offerte dall'ambiente. In particolare, l'inserimento e la rimozione di nuovi host sono controllati usando la funzione `pvm_notify`: qualsiasi cambiamento di configurazione nel metacomputer viene segnalato all'interfaccia e mostrato all'utente. Il meccanismo di notifica consente di controllare anche eventuali variazioni prodotte da programmi esterni: per esempio, se si aggiungono o tolgono host dalla console PVM, la modifica viene rilevata anche da WAMM.

[1] *xpm* è una libreria liberamente distribuibile che consente di semplificare l'uso delle pixmap sotto X11; è disponibile via anonymous FTP presso `avahi.inria.fr`.

5.3 Controllo dei task

Purtroppo PVM non comprende dei meccanismi di notifica completi anche per i task: usando `pvm_notify` è possibile sapere quando un dato task termina, ma non quando un nuovo task compare nel sistema. Per avere un controllo completo anche sui task, pertanto, l'interfaccia fa uso di "processi satellite" denominati `PVMTasker`. In fase di inizializzazione viene lanciato un PVMTasker su ogni nodo presente nella macchina virtuale. Ogni PVMTasker interroga periodicamente il proprio demone PVM per ottenere l'elenco dei task attivi sul nodo remoto. Quando vengono rilevate variazioni rispetto al controllo precedente, vengono inviate le opportune informazioni all'interfaccia.

L'uso dei PVMTasker è solo uno dei possibili metodi per emulare una `pvm_notify` più completa; presenta lo svantaggio di dover installare il programma di controllo su ogni nodo indicato nel file di configurazione. Un metodo alternativo consiste nel lasciare che sia l'interfaccia stessa a richiedere, di tanto in tanto, l'elenco completo dei task (si può usare la funzione PVM `pvm_tasks`). Questo sistema non richiede satelliti ma presenta degli svantaggi; in particolare:

- è necessaria la trasmissione di dati da *tutti* i demoni al programma, indipendentemente dal fatto che vi siano state variazioni nel numero di task rispetto al controllo precedente. La prima soluzione prevede l'invio di messaggi solo quando è necessario;
- se uno dei nodi fallisce, la funzione `pvm_tasks` rimane sospesa fino al raggiungimento di un timeout; possono passare anche dei minuti prima che prosegua con i nodi successivi e restituisca al programma l'elenco dei task. La prima soluzione, basata su task indipendenti, non presenta questo problema: se un nodo fallisce, saranno solo i suoi task a non essere più aggiornati.

Esiste una terza soluzione che fa uso del concetto di *tasker*, introdotto per la prima volta nella versione 3.3 di PVM. Un tasker è un programma PVM abilitato a ricevere i messaggi di controllo che normalmente, nella macchina virtuale, sono usati per richiedere l'attivazione di un task. In sostanza, se su un nodo è presente un tasker, il demone locale non fa partire il programma, ma passa la richiesta al tasker. Il tasker la esegue; quando il processo che ha fatto partire termina, informa il demone.

È possibile realizzare un tasker che, oltre a dialogare con il demone, segnali l'attivazione e la terminazione dei propri task all'interfaccia. Questa soluzione risulta la meno onerosa in termini di comunicazioni sulla rete (vengono eliminati anche i messaggi periodici tra il task di controllo e il demone del suo nodo), ma non è esente da svantaggi:

- è ancora necessaria l'installazione di un programma su ogni nodo;
- non è possibile avere gli aggiornamenti sui processi PVM creati prima della registrazione dei tasker.

In fase di sviluppo dell'interfaccia sono state provate tutte e tre le soluzioni; è stata adottata la prima, in quanto è risultata di gran lunga la migliore, sia in termini di utilizzo della rete che di possibilità di controllo.

5.4 Comandi remoti

I processi PVMTasker descritti in precedenza vengono usati anche per eseguire programmi sugli host remoti: il task riceve il nome del programma e gli argomenti della linea di comando ed esegue una `fork`; il figlio esegue il programma; l'output viene trasmesso all'interfaccia.

Anche in questo caso sono possibili soluzioni alternative a quella descritta, che ha come difetto l'impossibilità di eseguire comandi remoti su host su cui non stia girando un PVMTasker.

Il metodo "classico" consiste nell'usare comandi quali `rsh` o `rexec`. Possono essere usati per qualsiasi host, anche se non è in PVM; per esempio, per conoscere il carico del nodo `evans.cnuce.cnr.it`, un utente collegato a `calpar.cnuce.cnr.it` può scrivere:

```
rsh evans.cnuce.cnr.it uptime
```

Il comando `uptime` viene eseguito su `evans` e la risposta viene mostrata su `calpar`. Non è necessario che i nodi appartengano alla macchina virtuale (né, in realtà, che abbiano PVM installato).

Il problema nasce dal fatto che `rsh` e `rexec` possono essere considerati alternativi:

- per poter usare `rsh` l'utente deve autorizzare gli host remoti ad accettare le richieste di esecuzione, creando un file `.rhosts` su ogni nodo utilizzato;

- per usare rexec non servono file, ma a differenza di rsh viene chiesta la password dell'account sull'host remoto.

Nessuno dei due metodi è veramente soddisfacente: i file .rhosts costituiscono un problema per la sicurezza e spesso sono vietati; la richiesta di password è inaccettabile quando gli account o i comandi da eseguire sono tanti. Per questo PVM consente l'uso di entrambi: nell'hostfile l'utente specifica, per ogni nodo, che cosa va usato (le stesse opzioni sono ammesse nel file di configurazione usato da WAMM). Queste informazioni sono necessarie in quanto PVM deve attivare il demone pvmd su tutti i nodi remoti che vengono inseriti nella macchina virtuale: usando, appunto, rsh o rexec.

Il metodo alternativo per eseguire un comando remoto richiederebbe, dunque, l'esame del file di configurazione per stabilire se, per il nodo interessato, sia richiesto rsh o rexec. Tuttavia è più comodo far lanciare il comando dal PVMTasker presente sul nodo remoto interessato: rispetto al PVMTasker, il comando viene eseguito localmente, dunque non viene usato né rsh né rexec.

5.5 Compilazione remota

Come descritto nella sezione precedente, la compilazione di un programma su un host remoto è affidata ad un task PVM chiamato PVMMaker. Il task viene eseguito, su ogni nodo richiesto, solo al momento della compilazione e termina subito dopo la conclusione del make. Ai PVMMaker vengono inviati la directory compressa e i parametri per il make; ciascuno espande la directory e avvia la compilazione sul proprio nodo, trasmettendo all'interfaccia i messaggi sull'attività in corso. Anche i messaggi di output prodotti dal compilatore vengono indirizzati all'interfaccia, sotto forma di normali messaggi PVM.

Molte delle considerazioni fatte per l'esecuzione dei comandi remoti sono ancora applicabili al caso della compilazione. Per trasferire un file su un host si può usare il comando UNIX rcp, ma richiede, al pari di rsh, la presenza di un opportuno file .rhosts sul nodo destinazione. L'unica alternativa valida è data dal protocollo ftp per il trasferimento dei file, ma la sua gestione è notevolmente più complessa di un semplice trasferimento di dati tra task ottenuto con le primitive PVM.

Per quanto riguarda l'esecuzione dei comandi necessari alla compilazione, in linea teorica l'interfaccia potrebbe usare anche in questo caso rsh o rexec, ma questa scelta, oltre a presentare i problemi descritti in precedenza, non sarebbe efficiente quanto la soluzione basata sui PVMMaker: il programma dovrebbe pilotare e controllare gli esiti di *tutte* le operazioni su *tutti* gli host. Usando i PVMMaker, il programma si limita a lanciare i task, inviare la directory con i sorgenti e mostrare all'utente i messaggi ricevuti via via dai vari PVMMaker. La compilazione avviene in parallelo su tutti i nodi.

Gli svantaggi nell'uso dei PVMMaker sono simili a quelli descritti per i PVMTasker: è necessario installare un PVMMaker su ogni nodo a cui l'utente ha accesso; inoltre non è possibile compilare su un nodo che non faccia parte della macchina virtuale (non sarebbe possibile attivare il PVMMaker). Per quanto riguarda il primo punto, si potrebbe pensare di riunire le funzionalità di PVMTasker e PVMMaker in un solo task, semplificando così l'installazione dell'interfaccia. Tuttavia la separazione in due task distinti offre una maggiore modularità.

6 Sviluppi futuri

La versione corrente di WAMM dovrebbe essere considerata solo come un punto di partenza per la costruzione di un ambiente completo per il metacomputing, basato su PVM. Certe scelte implementative (es: la suddivisione in moduli dalla struttura simile) sono state adottate in modo da semplificare il più possibile l'inserimento di nuove funzionalità.

Gestione delle risorse. La gestione delle risorse di un metacomputer (nodi, reti, ecc.) costituisce uno degli aspetti più importanti di ogni ambiente per il metacomputing. Attualmente WAMM mette a disposizione dell'utente solo i semplici meccanismi usati da PVM per il controllo dei nodi e l'avvio di task. Per esempio, per lanciare un task l'utente può unicamente scegliere se usare un host particolare, una classe di architetture o un nodo qualsiasi scelto da PVM in base ad un semplice criterio round-robin. Per applicazioni complesse è necessario l'uso di algoritmi di mapping più sofisticati. Tali algoritmi potrebbero essere implementati nel modulo che si occupa dell'attivazione dei task.

Studio delle prestazioni. La versione corrente dell'interfaccia consente l'attivazione di un task in modalità *trace*: ogni volta che il task richiama una funzione PVM, un messaggio opportuno viene inviato all'interfaccia. Registrando e organizzando in modo appropriato questi dati si possono ottenere tutte le informazioni necessarie per lo studio delle prestazioni dei programmi. In particolare si possono determinare i tempi spesi per il calcolo, per le comunicazioni, per fattori di overhead vari. Attualmente WAMM non fa alcun uso dei messaggi di traccia che riceve; una versione futura dovrebbe prevedere un modulo in grado di collezionare questi dati, mostrarli all'utente sotto forma di grafici o tabelle e salvarli in un formato “standard” utile per l'esame successivo mediante tool esterni, come, ad esempio, ParaGraph [HF94].

Comandi remoti. La possibilità di eseguire comandi remoti potrebbe essere sfruttata per compiere la stessa operazione contemporaneamente su più host. Questo tipo di funzionalità, attualmente non implementata, consentirebbe di risolvere alcuni problemi riguardanti lo sviluppo e la manutenzione dei programmi PVM. Per esempio, con un solo comando si potrebbe cancellare un vecchio file eseguibile PVM da un gruppo di nodi. L'ambiente si aspetta di trovare tali file sempre all'interno della directory `$PVM_ROOT/bin/$PVM_ARCH` di ciascuna macchina, dove tendono ad accumularsi. La cancellazione manuale non è proponibile se i nodi sono numerosi, dunque potrebbe essere vantaggioso sfruttare l'interfaccia.

7 Lavori correlati

Le funzionalità di WAMM possono essere suddivise in due categorie: da una parte vengono forniti all'utente vari servizi per il controllo e la configurazione del metacalcolatore; dall'altra, l'interfaccia può essere considerata anche come uno strumento di sviluppo per il software. Esistono ambienti per il metacomputing che offrono funzionalità, appartenenti ad una delle due categorie, confrontabili con quelle di WAMM.

XPVM è una console grafica per PVM dotata di strumenti per la gestione della macchina virtuale e dei suoi processi. L'utente è in grado di cambiare la configurazione del metacalcolatore (aggiungendo o rimuovendo nodi) e avviare processi, in maniera analoga a quanto avviene con WAMM. Rispetto a quest'ultimo, XPVM non permette però la stessa visione “geografica” della macchina virtuale, ed è probabilmente più indicato per sistemi con pochi nodi. XPVM è del tutto privo di facilitazioni per la distribuzione dei sorgenti, la compilazione e l'esecuzione di comandi remoti; tuttavia comprende una sezione dedicata all'analisi delle informazioni di traccia, ancora non implementata in WAMM.

HeNCE [BDG+91, BDG+92, BDG+94b] è un ambiente per metacomputing, basato su PVM, che consente una notevole semplificazione del ciclo di sviluppo del software. In particolare, HeNCE prevede un sistema per la distribuzione dei sorgenti dei programmi e per la loro compilazione sui vari nodi, molto simile a quello utilizzato in WAMM: i sorgenti possono essere compilati in parallelo su più macchine e l'operazione è controllata da processi del tutto paragonabili ai PVMMaker. In HeNCE mancano però tutte le funzionalità di gestione previste da WAMM; in molti casi è necessario, per queste, l'uso della console standard di PVM. Va notato il fatto che HeNCE è stato ideato con obiettivi diversi rispetto a quelli di WAMM: la semplificazione nello sviluppo delle applicazioni è dovuta principalmente all'impiego di un differente modello di programmazione, con astrazione delle comunicazioni, piuttosto che all'uso di strumenti automatici per la compilazione.

References

[BDG+91] Adam Louis Beguelin, Jack J. Dongarra, G. A. Geist, Robert Mancheck, and Vaidy S. Sunderam. Graphical development tools for network-based concurrent supercomputing. In *Proceedings of Supercomputing 91*, 1991.

[BDG+92] Adam Louis Beguelin, Jack J. Dongarra, G. A. Geist, Robert Mancheck, Vaidy S. Sunderam, Keith Moore, Reed Wade, and Jim Plank. *HeNCE: A Users' Guide. Version 1.2*, December 1992.

[BDG+93] Adam Louis Beguelin, Jack J. Dongarra, G. A. Geist, Robert Mancheck, Vaidy S. Sunderam, and Wicheng Jiang. PVM3 users' guide and reference manual. Technical Report ORNL/TM-12187, Oak Ridge National Lab, May 1993.

[BDG+94a] Adam Louis Beguelin, Jack J. Dongarra, G. A. Geist, Robert Mancheck, Vaidy S. Sunderam, and Wicheng Jiang. PVM3 users' guide and reference manual. Technical Report ORNL/TM-12187, Oak Ridge National Lab, May 1994.

[BDG+94b] Adam Louis Beguelin, Jack J. Dongarra, G. A. Geist, Robert Mancheck, Vaidy S. Sunderam, Keith Moore, and Peter Newton. *HeNCE: A Users' Guide. Version 2.0*, June 1994.

[BDGS93] Adam Louis Beguelin, Jack J. Dongarra, G. A. Geist, and Vaidy S. Sunderam. Visualization and debugging in a heterogeneous environment. *IEEE Computer*, 26(6):88–95, June 1993.

[Bra92] Marshall Brain. *Motif programming: the essentials... and more*. Digital Press, 1992.

[FC90] Richard R. Freund and Sunny Conwell. Superconcurrency: A form of distributed heterogeneous supercomputing. *Supercomputing Review*, pages 47–51, October 1990.

[GBD+94] Al Geist, Adam Beguelin, Jack J. Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press, Cambridge, Massachusetts, 1994. Disponibile via FTP: netlib2.cs.utk.edu, pvm3/book.

[Hah93] Harley Hahn. *A Student's Guide to UNIX*. Mc Graw-Hill, Inc., 1993.

[HF94] M. T. Heath and J. E. Finger. *ParaGraph: a tool for visualizing performance of parallel programs*. Oak Ridge National Lab, Oak Ridge, TN, USA, 1994.

[KPSW93] Ashfaq A. Khokhar, Viktor K. Prasanna, Muhammad E. Shaaban, and Cho-Li Wang. Heterogeneous computing: Challenges and opportunities. *IEEE Computer*, 26(6):18–27, June 1993.

[SC92] Larry Smarr and Charles E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):45–52, June 1992.

[Sun90] Vaidy S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4):315–339, December 1990.

[Sun92] Vaidy S. Sunderam. *Heterogeneous network-based concurrent computing environments*, volume 8, pages 191–203. Elsevier Science Publishers B. V., 1992.