



Openmosix e Beowulf: introduzione e confronto



Giovanni Perbellini



Agenda

- Cluster
 - Introduzione
 - Cluster ESD
- Openmosix
 - Comandi principali
- Beowulf (PVM)
 - Comandi principali
- Libreria PVM
 - API



Introduzione - Cluster

- “In informatica, un **cluster** è un insieme di computer connessi tramite una rete telematica” (Wikipedia)
- Lo scopo di un cluster è quello di distribuire una computazione molto complessa tra i vari computer che compongono il *cluster*
- In sostanza un problema che richiede molte elaborazioni per essere risolto viene scomposto in sottoproblemi separati i quali vengono risolti in parallelo



Requisiti cluster

- Sistema operativo
 - e.g., Linux con OpenMosix
- Hardware di rete
- Un algoritmo il cui calcolo può essere eseguito in parallelo

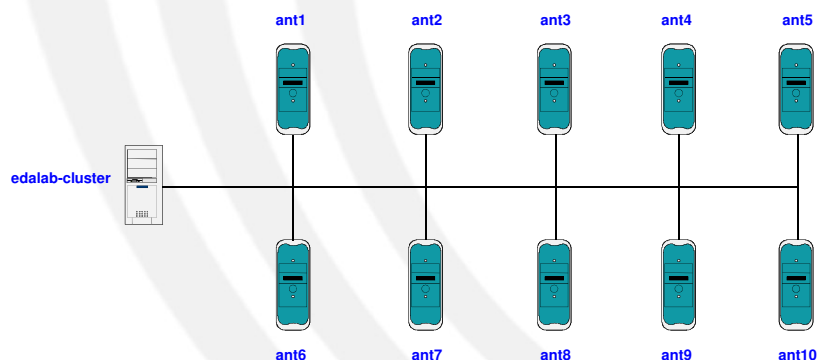


Tipi di cluster

- Failover
 - Monitoraggio dei servizi e migrazione in caso di caduta dei servizi
- Load-balancing
 - Bilanciamento dei processi con migrazione in caso di saturazione delle risorse
 - Esempio: Openmosix
- High Performance Computing (HPC)
 - I processi vengono sparsi sui nodi del *cluster*
 - Esempio: Beowulf



Cluster ESD





Openmosix - Introduzione

- Modifiche al kernel Linux per abilitare la migrazione dei processi
- Web Link:
`http://sourceforge.net/projects/openmosix`
- 2 componenti principali:
 - Patch per il Kernel del O.S.
 - Licenza GPL
 - User-space tools
 - Licenza GPL
 - Funzioni per la comunicazione tra i nodi e interfaccia per le funzioni kernel



Openmosix - Obiettivo

- Migrazione automatica dei **processi** tra i nodi del cluster in modo da bilanciare il carico
- I nodi del cluster notificano agli altri nodi circa il loro stato
 - Esempio: numero di processi eseguiti, uso del processore e della memoria
- Se un nodo del cluster determina che un altro nodo ha più risorse per gestire un processo, il processo viene migrato su un altro nodo
- Controllo centralizzato di tipo master/slave



Openmosix – Vantaggi (I)

- Adattabile: Non è necessario utilizzare nodi identici, ma possono essere usate varie tipologie di CPU
- Economico: Non è necessario usare HW dedicato
- Scalabile: per aumentare le prestazioni è sufficiente aumentare il numero dei nodi (max 65536)



Openmosix – Vantaggi (II)

- Trasparente: Una volta in esecuzione, sarà il sistema stesso a bilanciare il carico di lavoro e a ripartire i processi sui vari nodi.
Nessun cambiamento al codice da parallelizzare (nessuna libreria richiesta).
- Flessibile: E' possibile aggiungere o togliere un nodo senza dover bloccare tutto il *cluster*



Openmosix - Svantaggi

- Esecuzione lenta per i processi che fanno molti I/O runs
 - Tutti gli I/O hanno bisogno di essere comunicati tra i nodi sorgente e destinazione del cluster
- Dispendiosa l'esecuzione di processi paralleli (e.g., applicazioni scientifiche) che hanno bisogno di accedere a grossi file di dati
 - Soluzione: MOPI (MOSIX Parallel I/O)
 - Divisione del file di dati tra i nodi del cluster e quando un processo ha bisogno di accedere a certe porzioni del file viene migrato sull'appropriato nodo



Openmosix – Esempi OK

- Programmi che eseguono “fork” per creare processi figli
- Programmi che creano più istanze dello stesso programma
- Esempi:
 - MatLab 5
 - MJPEG tools (gestione di file audio/video)



Openmosix – Esempi NO

- Programmi che fanno uso di memoria condivisa
- Esempi:
 - Oracle
 - Postgres
- Esistono alcune *patch* per eliminare il problema



Openmosix – user-space tools

- **mosmon** (openMosix monitor)
 - permette di vedere lo stato di tutti i nodi incluso l'utilizzo della cpu , la memoria installata, memoria usata, etc.
- **mtop**
 - versione potenziata di top che mostra su quale nodo viene eseguito un processo



Openmosix – user-space tools

- **mps**
 - versione potenziata di `ps`
 - mostra anche il numero dei nodi
- **mosctl whois**
 - `mosctl whois "nodenumber"` è possibile vedere l'ip o l'hostname di *nodenumber*



Beowulf

- Insieme di SW e librerie per emulare un'architettura concorrente
- Obiettivo: permettere all'insieme di macchine di cooperare per eseguire un calcolo parallelo
- Modalità: Permette al programmatore di dividere i compiti da svolgere su un gruppo di computer collegati in rete e di riunire i risultati dei singoli processi per ottenere la soluzione del problema trattato



Beowulf

- Le più famose
 - PVM (Parallel Virtual Machine)
 - MPI (Message Passing Interface)



PVM – Principi base

- Applicazione è suddivisa in task, responsabile di una parte di computazione dell'intera applicazione
- Task (processo)
 - Unità di lavoro che viene eseguita in parallelo in PVM
 - *Processo* sequenziale e indipendente che alterna tra computazione e comunicazione
- I task dell'applicazione eseguono su un insieme di macchine che sono selezionate dall'utente
 - Nodi selezionati per una determinata esecuzione
 - Aggiunta/rimozione nodi a run-time



PVM – Principi base

- Modello message-passing
 - Invio e ricezione di messaggi tra la collezione di task
- Supporto dell'eterogenietà
 - PVM supporta eterogeneità in termini di nodi (macchine), rete e applicazioni



PVM - Componenti

- Composto da due parti
 - Demone pvmd3
 - Risiede sul nodo che fa da macchina virtuale
 - Libreria di routine PVM
 - Insieme di primitive utili per la cooperazione tra i task di un'applicazione
 - Message passing, spawning processi, coordinazione dei task, etc.



PVM - Parallelismo

- *Functional parallelism*
 - L'applicazione è suddivisa in funzioni che vengono attribuite a diversi task
 - Esempio: input, output, setup, solution, display
- *Data parallelism*
 - I task sono tutti uguali ma ognuno risolve solo una parte dei dati
 - SPMD (Single Program Multiple Data)
- PVM può supportare entrambi i metodi nella stessa esecuzione



PVM – task

- Ogni task è identificato da un TID (*Task Identification*)
 - Unico su tutto il cluster
- Forniti dal demone **pvmd**
 - Non possono essere scelti dall'utente
- PVM contiene diverse funzioni che ritornano il TID per identificare i task



PVM – Funzioni di libreria (I)

- **pvm_spawn()** per generare dei task (che verranno "allocati" automaticamente sulle macchine del *cluster*).
- **pvm_pack()** è una famiglia di funzioni, una per ogni tipo di dato; converte i dati in un formato standard (per permettere lo scambio dei dati tra architetture eterogenee) e li copia nel buffer di invio.
- **pvm_unpack()** è la simmetrica della **pvm_pack**, per la riconversione dei dati dopo la ricezione.
- **pvm_send()** invia dei dati ad un task.
- **pvm_mcast()** invia dei dati a più task (multicast).
- **pvm_recv()** riceve dei dati da un task specifico (o da qualunque task).



PVM – Funzioni di libreria (II)

- **pvm_trecv()** riceve dei dati, entro un certo tempo max, altrimenti ritorna.
- **pvm_nrecv()** riceve dei dati se sono già presenti, altrimenti esce (*receive* non bloccante).
- **pvm_ingroup()** aggiunge il task chiamante al gruppo specificato (se non esiste, lo crea).
- **pvm_leavegroup()** lascia un gruppo.
- **pvm_bcast()** manda un messaggio a tutti i task del gruppo.
- **pvm_barrier()** ferma l'esecuzione del task, finchè tutti i membri del gruppo chiamano **pvm_barrier()** (per la sincronizzazione).



PVM

- Linguaggi: C, C++, Fortran
- Web link
 - <http://www.netlib.org/pvm3/index.html>
 - <ftp://ftp.netlib.org/pvm3>
- Librerie C/C++: **libpvm3.a** e **libgpvm3.a**
- Demone
 - % **pvm**
 - Per accedere al prompt dei comandi di PVM
 - % **xpvm**
 - PVM con interfaccia grafica



PVM – Comandi (I)

- **pvm> add hostname**
 - Aggiunta di **hostname** alla macchina virtuale
- **pvm> delete hostname**
 - Rimozione di **hostname** dalla macchina virtuale
 - I processi in esecuzione su **hostname** vengono persi
- **pvm > conf**
 - Configurazione della macchina virtuale (*hostname* presenti sulla macchina virtuale, task ID, tipo di architetture, etc.)



PVM – Comandi (II)

- **pvm> ps -a**
 - Elenco di tutti i processi in esecuzione sulla macchina virtuale, i loro task ID, etc.
- **pvm> quit**
 - Uscita dal prompt dei comandi, lasciando in esecuzione i demoni e tutti i processi PVM in esecuzione
- **pvm> halt**
 - Uscita dal prompt dei comandi, con terminazione di tutti i processi PVM e di tutti i demoni



PVM – Comandi (III)

- **pvm> spawn**
 - Inizia l'esecuzione di un'applicazione
 - Opzioni:
 - **count**
 - Numero di task (default = 1)
 - **host**
 - Esecuzione su un determinato host (default = any)
 - **?**
 - Abilita il debug
 - **> file**
 - Redirezione dell'output del task su file