openMosix

Linux nel calcolo distribuito

Dino Del Favero, Micky Del Favero

dino@delfavero.it, micky@delfavero.it

BLUG - Belluno Linux User Group

Linux Day 2004 - Belluno 27 novembre

Cos'è openMosix

openMosix è una patch per il kernel di Linux per trasformare un gruppo di computer in un *cluster*.

Cos'è openMosix

openMosix è una patch per il kernel di Linux per trasformare un gruppo di computer in un *cluster*.

Un *cluster* è un gruppo di computer interconnessi al fine di permettere l'esecuzione di uno o più processi migliorando le prestazioni ottenibili facendo girare gli stessi processi sui singoli nodi componenti il cluster.

Tipi di cluster

Fail Over (HA): due o più macchine connesse tra di loro con più link *hearbeat*, se una macchina si guasta l'altra ne prende automaticamente il posto garantendo la continuità del servizio.

Tipi di cluster

Fail Over (HA): due o più macchine connesse tra di loro con più link *hearbeat*, se una macchina si guasta l'altra ne prende automaticamente il posto garantendo la continuità del servizio.

Load Balancing: due o più macchine connesse ad un apparato che dirige le richieste in modo da rendere il carico uniforme su tutte le macchine del cluster.

Tipi di cluster

Fail Over (HA): due o più macchine connesse tra di loro con più link *hearbeat*, se una macchina si guasta l'altra ne prende automaticamente il posto garantendo la continuità del servizio.

Load Balancing: due o più macchine connesse ad un apparato che dirige le richieste in modo da rendere il carico uniforme su tutte le macchine del cluster.

High Performance Computing (HPC): due o più macchine connesse tra di loro al fine di ottenere una singola unità logica ottimizzata per il calcolo, a questa categoria appartengono *Beowulf* e *openMosix*.

openMosix vs Beowulf (1)

openMosix

- Nodi anche di potenze diverse

openMosix vs Beowulf (1)

openMosix

- Nodi anche di potenze diverse
- Software standard

openMosix vs Beowulf (1)

openMosix

- Nodi anche di potenze diverse
- Software standard
- Scalabilità

openMosix vs Beowulf (1)

openMosix

- Nodi anche di potenze diverse
- Software standard
- Scalabilità
- Un processo, un processore

openMosix vs Beowulf (1)

openMosix

- Nodi anche di potenze diverse
- Software standard
- Scalabilità
- Un processo, un processore
- Solo Linux/x86

openMosix vs Beowulf (2)

Beowulf

- Nodi della stessa potenza

openMosix vs Beowulf (2)

Beowulf

- Nodi della stessa potenza
- Software sviluppato ad hoc

openMosix vs Beowulf (2)

Beowulf

- Nodi della stessa potenza
- Software sviluppato ad hoc
- Un processo tutti i processori

openMosix vs Beowulf (2)

Beowulf

- Nodi della stessa potenza
- Software sviluppato ad hoc
- Un processo tutti i processori
- Non limitato a x86

Caratteristiche principali di openMosix

- Economico

Caratteristiche principali di openMosix

- Economico
  Non è necessario hardware dedicato; normali PC connessi in rete possono essere trasformati in cluster.

Caratteristiche principali di openMosix

- Economico
- Scalabile

Caratteristiche principali di openMosix

- Economico
- Scalabile
  Per migliorare le prestazioni è, spesso, sufficiente aggiungere nodi al cluster.

Caratteristiche principali di openMosix

- Economico
- Scalabile
- Trasparente

Caratteristiche principali di openMosix

- Economico
- Scalabile
- Trasparente
  Una volta in esecuzione sarà il sistema operativo stesso a bilanciare il carico sui vari nodi.

Caratteristiche principali di openMosix

- Economico
- Scalabile
- Trasparente
- Adattabile

Caratteristiche principali di openMosix

- Economico
- Scalabile
- Trasparente
- Adattabile
  Non sono necessari nodi identici.
  L' unico limite imposto da openMosix, attualmente, è che ogni macchina appartenga all'architettura x86.
  Un altro limite è che l' hardware sia supportato da GNU/Linux.

Caratteristiche principali di openMosix

- Economico
- Scalabile
- Trasparente
- Adattabile
- Flessibile

Caratteristiche principali di openMosix

- Economico
- Scalabile
- Trasparente
- Adattabile
- Flessibile
  E’ possibile aggiungere o togliere nodi senza dover fermare il cluster.

Come funziona openMosix (1)

openMosix rende possibile far migrare i processi in esecuzione tra i nodi componenti il cluster.

Come funziona openMosix (1)

openMosix rende possibile far migrare i processi in esecuzione tra i nodi componenti il cluster.

La tecnologia openMosix è costituita da due parti:

1. Un meccanismo di *Preempitive Process Migration (PPM)*.
2. Un insieme di algoritmi per la condivisione adattativa delle risorse.

Come funziona openMosix (1)

openMosix rende possibile far migrare i processi in esecuzione tra i nodi componenti il cluster.

La tecnologia openMosix è costituita da due parti:

1. Un meccanismo di *Preempitive Process Migration (PPM)*.
2. Un insieme di algoritmi per la condivisione adattativa delle risorse.

Gli algoritmi di condivisione forniscono al PPM le informazioni necessarie per decidere quale processo deve essere migrato verso quale dei nodi componenti il cluster.

Come funziona openMosix (2)

Ogni processo ha un *Unique Home-Node (UHN)*: il nodo su cui il processo è stato generato.

Come funziona openMosix (2)

Ogni processo ha un *Unique Home-Node (UHN)*: il nodo su cui il processo è stato generato.

Il modello *SSI (Single System Image)* di openMosix è un *Cache Coerent Cluster* in cui ogni processo sembra girare sul proprio UHN e dove ogni processo di una singola sessione utente condivide l'ambiente di esecuzione dell'UHN.

Come funziona openMosix (2)

Ogni processo ha un *Unique Home-Node (UHN)*: il nodo su cui il processo è stato generato.

Il modello *SSI (Single System Image)* di openMosix è un *Cache Coerent Cluster* in cui ogni processo sembra girare sul proprio UHN e dove ogni processo di una singola sessione utente condivide l'ambiente di esecuzione dell'UHN.

I processi migrati verso nodi remoti useranno le risorse locali (del nodo remoto) fintantoché ciò è possibile, ma interagiranno con l'ambiente utente attaverso l'UHN.

Quando migrano i processi

Un processo migra quando il carico del nodo (cpu, ram, ecc.) che esegue il processo supera una certa soglia.

Quando migrano i processi

Un processo migra quando il carico del nodo (cpu, ram, ecc.) che esegue il processo supera una certa soglia.

I processi in esecuzione vengono quindi assegnati dinamicamente ai nodi del cluster in modo da massimizzare lo sfruttamento delle risorse disponibili mantenendo il carico medio di ogni singolo nodo sotto una certa soglia critica.

Come migrano i processi (1)

Per implementare la PPM il processo da migrare è diviso in due contesti: *user context* che può migrare e *system context* che è dipendente dall'UHN.

Come migrano i processi (1)

Per implementare la PPM il processo da migrare è diviso in due contesti: *user context* che può migrare e *system context* che è dipendente dall'UHN.

Lo user context, detto *remote*, contiene *text, data, stack, memory maps, register* del processo. Il remote incaplusa il processo quando questo gira a livello user.

Come migrano i processi (1)

Per implementare la PPM il processo da migrare è diviso in due contesti: *user context* che può migrare e *system context* che è dipendente dall'UHN.

Lo user context, detto *remote*, contiene *text, data, stack, memory maps, register* del processo. Il remote incaplusa il processo quando questo gira a livello user.

Il system context, detto *deputy*, contiene una descrizione delle risorse a cui il processo è attaccato e un kernel-stack per l'esecuzione del codice a livello kernel. Il deputy incapsula il processo quando questo gira nel kernel, a lui appartiene la parte dipendente dal nodo del processo e quindi non può migrare.

Come migrano i processi (2)

L’interfaccia di comunicazione fra user e system context è ben definita e può essere intercettata, ogni interazione fra i due contesti e inviata attraverso la rete a nodi diversi.

Come migrano i processi (2)

L’interfaccia di comunicazione fra user e system context è ben definita e può essere intercettata, ogni interazione fra i due contesti e inviata attraverso la rete a nodi diversi.

Il tempo necessario alla migrazione ha una componente fissa necessaria ad istanziare un nuovo process frame nel nodo remoto e una lineare al numero di pagine di memoria che devono essere trasferite. Per minimizzare l’overhead solo la tabella delle pagine e le pagine usate sono trasferite.

Come migrano i processi (2)

L’interfaccia di comunicazione fra user e system context è ben definita e può essere intercettata, ogni interazione fra i due contesti e inviata attraverso la rete a nodi diversi.

Il tempo necessario alla migrazione ha una componente fissa necessaria ad istanziare un nuovo process frame nel nodo remoto e una lineare al numero di pagine di memoria che devono essere trasferite. Per minimizzare l’overhead solo la tabella delle pagine e le pagine usate sono trasferite.

Durante l’esecuzione di un processo la trasparenza al nodo viene garantita inviando le chiamate a sistema dipendenti dal nodo al deputy presso l’UHN, mentre quelle indipendenti dal nodo vengono eseguite localmente.

Il nostro cluster (1)

Il cluster dimostrativo che abbiamo portato è composto da 4 macchine di cui 2 diskless, su tutte le macchine gira Debian GNU/Linux “Sarge” e in particolare sono stati installati:

- bysybox
- initscripts
- net-tools
- debianutils
- bash
- coreutils
- modutils
- ifupdown
- sysvinit
- findutils
- udhcpc
- openmosix-tools

Il nostro cluster (2)

Sul cluster gireranno più istanze dello script di test tratto dall'*openMosix HOWTO* reperibile presso http://howto.ipng.be/openMosix-HOWTO/

```
for i in $(seq 1 10); do

  awk 'BEGIN {
    for(i=0;i<10000;i++)
      for(j=0;j<10000;j++);
  }' &

done
```

Una demo.

Domande?

Grazie per l’attenzione.