

Sesión 2: Middleware y Programación paralela de clusters

Indice

1. Middleware
2. OpenMosix
3. Computación Paralela
4. Modelos de Computación paralela
5. Modelos de cómputo
6. Procesos
7. Métricas de rendimiento
8. Metodología de computación paralela
9. Programación Paralela con MPI

1. Middleware

- El middleware es la diferencia clave entre una red de PCs y un cluster.
- Es el nivel de software que ejecuta por encima del sistema operativo y que proporciona al usuario una imagen de sistema único.
- Permite un acceso uniforme a los distintos nodos de cómputo independientemente del sistema operativo que implemente cada uno de ellos.
- Gestiona los fallos de los nodos individuales y el equilibrio de carga de trabajo.

1. Middleware

- Los objetivos que debe cubrir el middleware idealmente son los siguientes:

1. Punto de entrada único
 2. Jerarquía de ficheros única
 3. Punto de control único
 4. Red virtual única
 5. Espacio de memoria único
 6. Sistema de gestión de procesos único
 7. Interface de usuario único
 8. Espacio de Entrada/salida único
 9. Espacio de proceso único
 10. Gestión de puntos de chequeo
 11. Migración de procesos
- } Imagen de sistema único
- } Disponibilidad del cluster

2. OpenMosix

- Mosix es una herramienta de código abierto que proporciona equilibrio de carga de trabajo adaptativo entre computadores con arquitectura x86 y sistema operativo Linux.
- Utiliza migración de procesos para reasignar los procesos entre los nodos de forma que optimiza el uso de los recursos del sistema.
- Utiliza un algoritmo de asignación basado en el nodo más descargado.

URL

<http://www.mosix.org>

2. OpenMosix

- OpenMosix es un parche sobre el kernel de Linux.
- Trabaja con granularidad de proceso (no puede migrar threads)
- Su meta principal es optimizar el rendimiento del cluster y crear un entorno multiusuario de tiempo compartido que permita la ejecución tanto de trabajos secuenciales como paralelos en el cluster.

2. OpenMosix

■ Transparencia de red:

- Oculta tanto el interface de usuario como los programas de nivel de aplicación la existencia de la red de comunicación, presentando el sistema como un único computador.
- Por ejemplo el acceso remoto a ficheros del servidor es completamente transparente a los programas.

■ Migración de procesos con interrupción:

- Los procesos de usuario se pueden migrar de forma transparente y en cualquier instante de tiempo a un nodo con mayor capacidad de cómputo.
- El proceso migrado se divide en dos contextos:
 - Sistema: que no puede migrarse.
 - Usuario: que puede migrarse a un nodo sin disco.

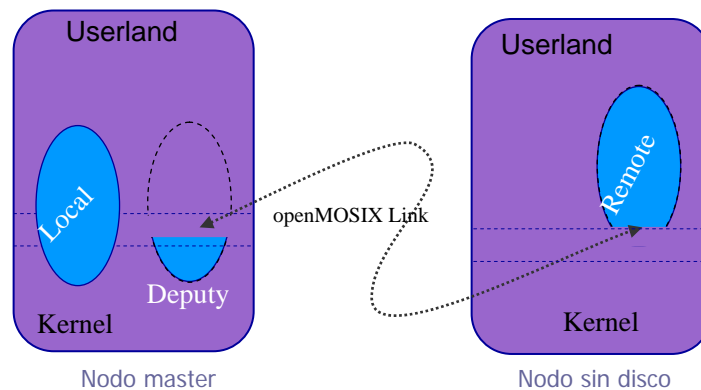
10 de mayo de 2006

Jose Luis Bosque

7

2. OpenMosix

Migración de procesos:



10 de mayo de 2006

Jose Luis Bosque

8

2. OpenMosix

■ Equilibrio de carga de trabajo dinámico:

- La migración de procesos se realiza para equilibrar la carga de trabajo de los nodos del cluster.
- Al dinámico permite responder a cambios en tiempo de ejecución de la carga de los nodos, de las características de los procesos, del número de nodos y de su capacidad de cómputo.
- OpenMosix constantemente intenta reducir la diferencia de carga entre pares de nodos a base de migración de procesos.
- El algoritmo es simétrico y distribuido: todos los nodos ejecutan el mismo algoritmo y la reducción en la diferencia de carga se realiza independientemente para cada par de nodos.

2. OpenMosix

■ Gestión de memoria:

- Entre los objetivos de la migración está maximizar el uso de la memoria física de los nodos del cluster.
- No hace una compartición real pero intenta que se use la mayor cantidad de memoria física posible.
- Minimiza el número de páginas de memoria que están en la zona de swap.
- En la decisión de la migración de procesos también se tiene en cuenta las necesidades de memoria del proceso y la memoria libre del nodo al que se migra.

■ Comunicación eficiente entre núcleos:

- OpenMosix ha sido especialmente desarrollado para minimizar las comunicaciones internas del núcleo.
- Ha desarrollado un protocolo propio rápido y fiable con baja latencia y gran ancho de banda.

2. OpenMosix

■ Algoritmos de diseminación de información probabilísticos:

- Se encargan de proporcionar a cada nodo información suficiente sobre el estado del resto de los nodos sin necesidad de hacer encuestas.
- Mide la cantidad de recursos disponibles en cada nodo.
- Recibe "índices de los recursos disponibles que cada nodo envía a intervalos fijos de tiempo a un subconjunto aleatoriamente seleccionado de nodos.
- Esto permite una reconfiguración dinámica, facilita la escalabilidad de la aplicación y tolerancia a fallos.

■ Control descentralizado y autonomía:

- Nada nodo toma sus propias decisiones de control de forma independiente evitando relaciones maestro-escalvo entre nodos.
- Cada nodo es capaz de operar como un sistema independiente: esto permite una configuración dinámica en la que los nodos se agregan o abandonan el sistema sin problemas

10 de mayo de 2006

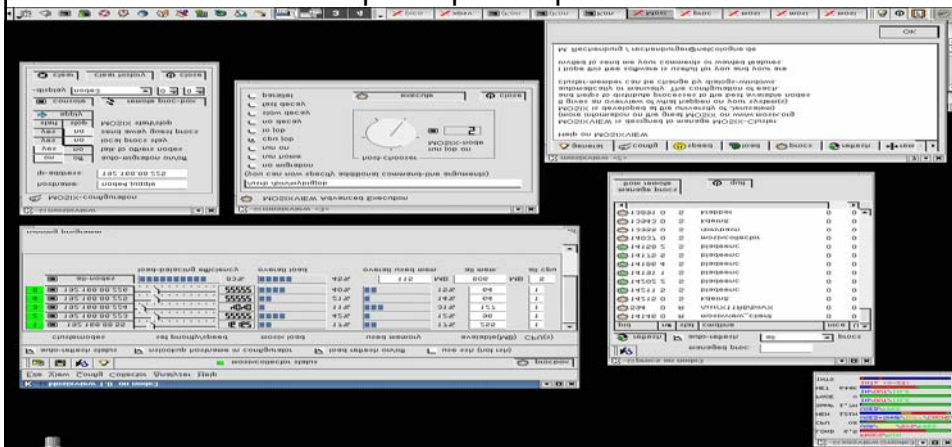
Jose Luis Bosque

11

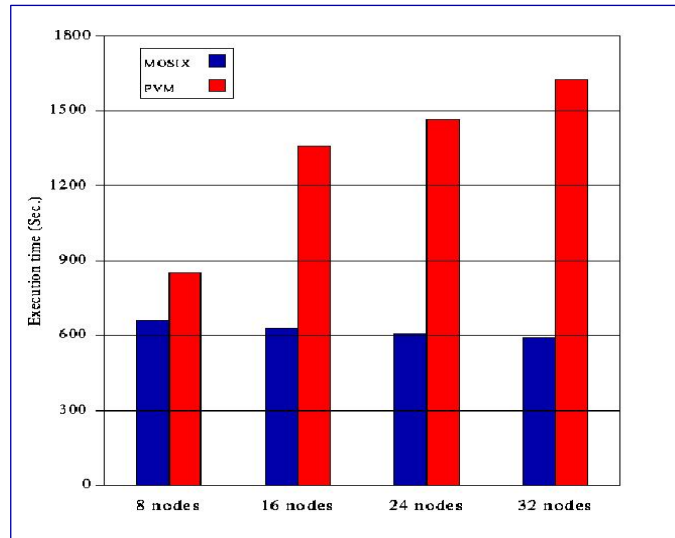
2. OpenMosix

■ Mosixview:

- Visualiza la carga, velocidad y la utilización de memoria a lo largo de todos los nodos del cluster.
- Utiliza el interface /proc/hpc/info para recabar la información.



2. OpenMosix



10 de mayo de 2006

Jose Luis Bosque

13

3. Computación paralela

- **Computación paralela**: forma eficaz de proceso de la información, que favorece la explotación de sucesos concurrentes en el proceso de cómputo.
- Objetivo fundamental: aumentar el rendimiento
- Computación paralela:
 - Memoria Compartida.
 - Paso de mensajes.

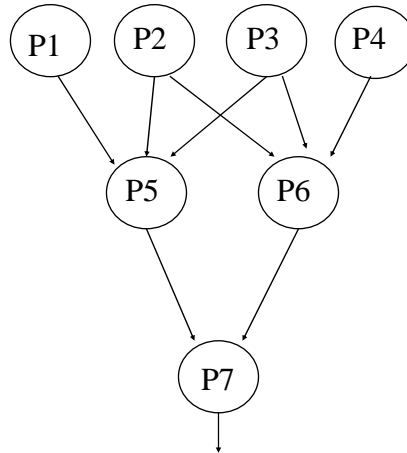
10 de mayo de 2006

Jose Luis Bosque

14

3. Modelos de Computación

- Aplicación: conjunto de procesos independientes que se comunican y sincronizan mediante paso de mensajes.
- Computación de grano grueso: gran importancia de la relación tiempo de comunicación y tiempo de cómputo.
- Herramientas de desarrollo:
 - MPI.
 - PVM.



4. Modelos de cómputo

- Paralelismo funcional:
 - Tareas diferentes que se pueden ejecutar en paralelo.
 - Inherente en todas las aplicaciones.
 - Generalmente tienen un grado de paralelismo bajo.
- Paralelismo de datos:
 - Una misma tarea se ejecuta en paralelo sobre un conjunto de datos.
 - Se tiene réplicas del mismo programa trabajando sobre partes distintas de los datos.
 - Control centralizado o distribuido.
 - Grado de paralelismo muy alto, pero no está presente en todas las aplicaciones.

5. Procesos

- Proceso: programa en ejecución junto con un conjunto de recursos propios.
- Ciclo de vida:
 - Creación: preparar el programa para su ejecución.
 1. Cargar en memoria el programa: la información se almacena en la tabla de procesos.
 2. Asignarle un espacio de direcciones: depende del planificador de memoria.
 3. Pasar el descriptor de proceso al planificador: añadir a la cola de procesos listos para ejecutar.

5. Procesos

- Ejecución: depende del planificador.
 - Declaración de estados.
 - Diagrama de transición de estados.
 - Política de planificación => afecta mucho al rendimiento.
 - Realiza tareas de cómputo y de comunicación con otros procesos, locales o remotos.
- Finalización: se termina el proceso y se liberan los recursos previamente asignados.

6. Métricas de Rendimiento

- Objetivo de paralelismo: aumentar el rendimiento de los programas.
- Depende de las capacidades del sistema y del comportamiento del programa.
 - Tiempo de ejecución: tiempo desde que se inicia el programa hasta que finaliza.
 - Tiempo de CPU: tiempo que realmente está ejecutando, es decir el tiempo que tiene asignado la CPU.
 - Tiempo de Comunicación: tiempo invertido por la aplicación en funciones de comunicación entre procesos.
 - Sobrecarga: Tiempo total en el que la aplicación paralela no hace trabajo útil.

6. Métricas de Rendimiento

- Speedup: es la ganancia del sistema paralelo, frente al secuencial.

$$S = \frac{T_1}{T_N}$$

- Eficiencia: % de tiempo empleado en proceso efectivo.

$$E = \frac{S}{N} = \frac{T_1}{T_N \cdot N}$$

6. Métricas de Rendimiento

- Fracción serie: es el % del programa que no se puede paralelizar.
- Ley de Amdahl: Si la fracción serie de un programa es “s” el speedup máximo que se puede alcanzar es $1/s$.

$$Speedup = \frac{s + p}{s + \frac{p}{N}} = \frac{1}{s + \frac{p}{N}} = \frac{1}{s}$$

6. Métricas de Rendimiento

- Ley de Gustafson:
 - Tiempo constante incrementando el volumen de datos.
 - Aumentar el volumen de datos aumenta el % de paralelismo.

$$\begin{aligned} \text{Scaled Speedup} &= \frac{s' + p' \cdot N}{s' + p'} = \\ &= s' + p' \cdot N = N + (1 - N) \cdot s' \end{aligned}$$

7. Metodología de Programación Paralela

- Fases de la programación paralela:
 - **Descomposición funcional:** identificar las funciones que debe realizar la aplicación y las relaciones entre ellas.
 - **Partición:**
 - Distribución de las funciones en procesos y esquema de comunicación entre procesos.
 - Objetivos: maximizar el grado de paralelismo y minimizar la comunicación entre procesos.

7. Metodología de Programación Paralela

- Fases de la programación paralela:
 - **Localización:**
 - Los procesos se asignan a procesadores del sistema.
 - Objetivo: ajustar los patrones de comunicación a la topología del sistema.
 - **Escalado:** Determina el tamaño óptimo del sistema en función de algún parámetro de entrada.

8. Programación con MPI

- Especificación de una librería para paso de mensajes, propuesto como un estándar por un amplio comité de vendedores, implementadores y usuarios.
- Colección de funciones que oculten detalles de bajo nivel tanto HW como SW.
- Diseñado para obtener un alto rendimiento tanto en máquinas paralelas como en clusters.

8. Programación con MPI

- Objetivos del Forum MPI:
 - Definir un entorno de programación único que garantice la portabilidad de las aplicaciones paralelas.
 - Definir totalmente el interfaz de programación, sin especificar la implementación.

8. Programación con MPI

- Objetivos de MPI Forum (II):
 - Ofrecer implementaciones de calidad de dominio público para favorecer la extensión del estándar.

 - Convencer a los fabricantes de computadores paralelos para que ofrezcan versiones MPI optimizadas para sus máquinas

8. Programación con MPI

- Elementos comunes a toda implementación:
 - Biblioteca de funciones C, C++ (FORTRAN), más fichero de cabecera `mpi.h` con la definición de funciones, constantes y macros.
 - Comandos de compilación que incorporan automáticamente las librerías MPI.
 - Comandos para ejecución de aplicaciones.
 - Herramientas de monitorización y depuración.

8. Programación con MPI

■ Limitaciones:

- Entrada/salida: no establece un mecanismo estandarizado de E/S paralela.
- Creación dinámica de procesos. MPI asume un número de procesos constante al arrancar la aplicación.
- Variables compartidas: el modelo de comunicación estandarizado por MPI sólo tiene en cuenta el paso de mensajes.

8. Programación con MPI

■ Limitaciones (II):

- Enlaces con otros lenguajes como C++ y Ada.
- Soporte para aplicaciones en tiempo real. No recoge en ningún punto restricciones relacionadas con tiempo real.
- Interface gráficos: No se define ningún aspecto relacionado con la interacción con la aplicación paralela.

8. Programación con MPI

- La unidad básica de paralelismo son los procesos independientes.
- Tienen espacios de memoria independientes.
- Intercambio de información y sincronización mediante paso de mensajes.
- Comunicadores: grupos de procesos, que definen el ámbito de las operaciones colectivas.
- A cada proceso se le asigna un identificador interno propio de MPI.

8. Programación con MPI

```
#include "mpi.h"
main (int argc, char **argv)
{
    int nproc, myid;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &nproc);
    MPI_Comm_Rank (MPI_COMM_WORLD, &myid);
    /* CUERPO DEL PROGRAMA */
    MPI_Finalize ();
}
```

8. Programación con MPI

- Transferencia de información entre dos procesos.
- Dos modelos:
 - Bloqueante: mantiene al proceso bloqueado hasta que la operación finalice.
 - No bloqueante: se recupera el control inmediatamente.
- ¿ Cuándo finaliza una operación ?
 - Recepción: cuando tengamos un mensaje completo en el buffer designado.
 - Envío: cuando el emisor puede reutilizar sin interferencias el buffer de emisión.

8. Programación con MPI

- Se definen 4 modos de envío:
 - Básico: no especifica la forma en que se completa la operación; depende de la implementación.
 - Con buffer: se copia el mensaje a un buffer local y se da por finalizada la operación.
 - Síncrono: la operación finaliza sólo cuando el mensaje ha sido recibido en el destino
 - Listo: sólo se puede hacer si antes el otro extremo está preparado para una recepción inmediata.

8. Programación con MPI

- Funciones básicas bloqueantes:
- `int MPI_Send (void *buf, int count, MPI_DataType datatype, int dest, int tag, MPI_Comm comm);`
- `int MPI_Recv (void *buf, int count, MPI_DataType datatype, int source, int tag, MPI_Comm comm, MPI_status *status);`
- `int MPI_Get_count (MPI_status *status, MPI_Datatype datatype, int *count);`

8. Programación con MPI

- Funciones básicas no bloqueantes:
 - `MPI_Isend` y `MPI_Irecv`: añaden un parámetro “recibo” `MPI_request` para comprobar si la operación ha terminado.
 - `MPI_Wait`: toma un recibo y bloquea el proceso hasta finalizar la operación asociada.
 - `MPI_Test`: informa si ha finalizado o no una operación.
 - `MPI_Cancel`: cancela operaciones de comunicación pendientes.

8. Programación con MPI

- Comunicación con buffer:
 - Se tiene mayor control sobre la duración de la comunicación.
 - MPI_Buffer_attach, MPI_Buffer_detach: asigna o libera una zona de memoria como buffer de salida.
 - MPI_Bsend: envío con buffer.
 - Estos envío fracasan si el buffer no tiene suficiente sitio para almacenar el mensaje.

10 de mayo de 2006

Jose Luis Bosque

37

8. Programación con MPI

- Recepción por encuesta:
 - Evitan la sincronización del receptor con el emisor.
 - MPI_Probe, MPI_Iprobe: permiten saber si tenemos un mensaje recibido y listo para leer pero sin leerlo.

10 de mayo de 2006

Jose Luis Bosque

38

8. Programación con MPI

- Etiquetas y comunicadores:
 - Las etiquetas permiten diferenciar distintas clases de información (contexto).
 - Comunicador (entorno de comunicación): conjunto de procesos.
 - MPI garantiza la entrega ordenada de mensajes dentro de un mismo comunicador.
 - Permiten a distintos grupos de procesos intercambiar información sin interferirse.

8. Programación con MPI

- Permiten comunicación entre más de dos procesos.
 - `int MPI_Barrier (MPI_Comm comm)`: Función de sincronización; bloquea a todos los procesos hasta que todos hayan pasado por esa barrera.
 - `MPI_Bcast`: el proceso raíz envía un mensaje a todos los procesos del comunicador.
 - `MPI_Gather`: Recolección de datos en el proceso raíz. Todos los procesos del comunicador contribuyen con la misma cantidad de datos y estos se almacenan de forma consecutiva.

8. Programación con MPI

- MPI_Scatter: envía a cada proceso una parte de un vector de datos almacenado en el raíz.
- MPI_Alltoall: comunicación todos con todos.
- MPI_Reduce: operación cooperativa realizada entre todos los miembros del comunicador y se obtiene un único resultado final.
- MPI_Op_create (MPI_Op_free): permite crear funciones que se pueden utilizar en MPI_Reduce.

8. Programación con MPI

- Grupo: conjunto de procesos + espacio de direcciones.
- Comunicador: grupo de procesos + contexto.
 - MPI_Comm_dup, MPI_Comm_free: crea/destruye un nuevo comunicador con el mismo conjunto de procesos pero diferente contexto. Operación colectiva.
 - MPI_Comm_split: crea varios comunicadores con distintos procesos.
 - MPI_Intercomm_create: crea un intercomunicador para permitir comunicación entre procesos de distinto grupo.

8. Programación con MPI

- Agrupar datos heterogéneos en un solo mensaje.
 - MPI_Pack: almacena de forma explícita datos heterogéneos en un buffer con posiciones de memoria contiguas.
 - MPI_Unpack: copia los datos recibidos en una o varias variables locales.

8. Programación con MPI

- “MPI: A Message Passing-Interface standard”, MPI Forum. 1995
- “MPI-2: extensions to the message-Passing Interface”, MPI Forum. 1997.
- “A user’s Guide to MPI”, Peter Pacheco. Univerty of San Francisco.
- “MPI Primer / Developing with LAM”. Ohio Supercomputer Center, Ohio State University.

8. Programación con MPI

- “Programación de aplicaciones paralelas con MPI”, José Miguel Alonso. UPV.
- [Http://www.mpi-forum.org](http://www.mpi-forum.org)
- <http://www.mcs.anl.gov/mpi>