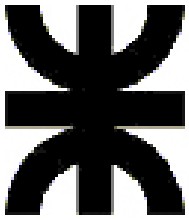


Universidad Tecnológica Nacional
Facultad Regional Santa Fe
Ingeniería en Sistemas de Información

Sistemas Operativos Avanzados

Trabajo Práctico Final

Procesamiento paralelo: qué tener en cuenta para aprovecharlo. Conceptos y alternativas en Linux.

Smerling, Leonardo
Tschanz, Raúl

Año: 2001

Procesamiento paralelo: qué tener en cuenta para aprovecharlo
Conceptos y alternativas en Linux.

Contenido del documento

Introducción y objetivos

Este trabajo pretende introducir al lector en los fundamentos del procesamiento en paralelo, empezando por una breve historia y entrando rápidamente en los conceptos del paralelismo en la computación. Luego se describen las arquitecturas comunes del procesamiento paralelo, clasificación y características de cada una.

Cabe destacar en este punto, que si bien este documento presenta una serie de conceptos teóricos, no se pretende ahondar en todas las alternativas de procesamiento paralelo descritas en el punto relativo a las arquitecturas. Se ha trabajado mucho en el procesamiento simétrico y las maquinas con multiprocesadores a nivel hardware y kernels de sistemas operativos. Es bueno conocer estas alternativas antes de decidir la implementación de un sistema con paralelismo de cómputo, pero el objeto principal de este trabajo es tratar de explicar mejor una modalidad relativamente incipiente que se viene dando en el mundo de la informática y que se orienta al aprovechamiento de los recursos y tecnología de hardware y software existentes para el armado de "equipos" capaces de alcanzar un alto rendimiento si se los agrupa y se aprovecha la posibilidad de paralelismo en la operatoria: nos referimos a los clusters.

Si bien los clusters pueden venir pre-ensamblados y optimizados con distintos tipos de mejoras de hardware y software propias de cada proveedor, existe con la tecnología y conocimientos actuales la posibilidad de que con unos pocos recursos –llámese unas cuantas PC estándar, comunicadas por una red Ethernet por mencionar una popular—se pueda aprovechar las características del procesamiento paralelo. Los clusters de tipo Beowulf, especialmente los de Clase I, están orientados a aprovechar hardware y software disponible masivamente, así como también los conocimientos de sistemas operativos tan populares como Linux y redes Ethernet con protocolos TCP/IP para el armado relativamente económico y la implementación exitosa del procesamiento paralelo. También son tan flexibles como para incorporar tecnologías propias que mejoren el rendimiento de operaciones aprovechando las características propias del cluster.

No es nuestro objetivo tampoco en este trabajo adentrarnos en especificaciones de hardware, ni administración y configuración de Linux como sistema de base para clusters Beowulf, pero sí describiremos el software adicional necesario y a la vez más comúnmente usado para la configuración de clusters y la programación paralela. También hemos tratado de realizar comparaciones entre las tecnologías descritas ya que, como se verá, es muy difícil elegir un solo componente de software que satisfaga todos los requerimientos y a la vez aproveche al máximo las posibilidades de la programación paralela.

Esperamos este trabajo sirva de guía para quienes buscan una alternativa a la adquisición de costosos sistemas de procesamiento paralelo y a la vez anime a muchos a emprender la implementación de un cluster propio con el objetivo de mejorar la performance y la economía de procesos que pueden valerse del paralelismo.

Procesamiento paralelo

Historia e introducción al procesamiento paralelo

La industria informática es una de las industrias de mayor crecimiento y es impulsada por los rápidos desarrollos que se producen en las áreas del hardware y del software. Los avances tecnológicos del hardware incluyen el desarrollo de chips y nuevas tecnologías de fabricación, microprocesadores más rápidos y baratos, así como también mayor ancho de banda y menor latencia en las interconexiones de redes. También en el área del software se producen avances rápidos. Software maduro, como sistemas operativos, lenguajes de programación, metodologías de desarrollo y herramientas complementarias están ahora disponibles.

Sin embargo, hay muchas áreas dentro de la informática que necesitan de una gran inversión en equipos, y esta inversión rara vez llega o es suficiente. Algunas de estas áreas son la meteorología (una simulación de la atmósfera puede llegar a ser tremendamente pesada), la física de la materia condensada con modelos muy densos (fundamentalmente, cuánticos o, para problemas de dinámica molecular, semiempíricos), el estudio de proteínas, análisis de terremotos, la generación de imágenes por computador con modelos realísticos o el secuenciamiento del genoma humano.

El procesamiento paralelo consiste en acelerar la ejecución de un programa mediante la descomposición en fragmentos que puedan ejecutarse de forma paralela, cada uno en una unidad de proceso diferente.

Una forma de clasificar la computación, es a través de dos eras de desarrollo:

- Era de computación secuencial
- Era de computación paralela

Cada era de la computación comienza con un desarrollo en la arquitectura del hardware, seguida por sistemas de software (particularmente en las áreas de compiladores y sistemas operativos), aplicaciones, y finalmente alcanzando el punto máximo en los ambientes de resolución de problemas (Problem Solving Environments – PSE). Cada componente de un sistema computacional abarca tres fases: Investigación y Desarrollo (Research and Development –R&D), comercialización, y disponibilidad masiva. La tecnología detrás del desarrollo de componentes de sistemas computacionales en la era secuencial ha alcanzado su madurez, y desarrollos similares están a punto de producirse en la era paralela. Esto significa que la tecnología de la computación paralela necesita avanzar, ya que aún no está lo suficientemente madura como para ser explotado como una tecnología de disponibilidad masiva.

La razón principal para crear y utilizar computación paralela es que el paralelismo es una de las mejores formas de salvar el problema del cuello de botella que significa la velocidad de un único procesador.

La razón de ser del procesamiento en paralelo es acelerar la resolución de un problema. La aceleración (speedup) que puede alcanzarse depende tanto del problema en sí como de la arquitectura del ordenador paralelo. Las aplicaciones que se benefician de una aceleración más significativa son aquellas que describen procesos intrínsicamente paralelos. Las simulaciones de modelos moleculares, climáticos o económicos tienen todas una amplia componente paralela, como los sistemas que representan. El hardware de la máquina entra en juego ya que es preciso maximizar la relación entre el tiempo de cálculo útil y el perdido en el paso de mensajes, parámetros que dependen de la capacidad de proceso de las CPUs y de la velocidad de la red de comunicaciones.

La clave es descomponer el problema de tal forma que cada procesador pueda operar el mayor tiempo posible sobre su fragmento de datos, sin tener que recurrir a los de los demás procesadores.

Programación en paralelo

Para el desarrollo de aplicaciones, y por consiguiente para diseñar la estructura y topología adecuadas para aprovechar el procesamiento paralelo, se debe obtener la independencia de algunas partes del programa. Hay 2 formas básicas de obtener partes independientes en un programa paralelo: descomposición funcional o descomposición de datos, que describiremos a continuación.

Descomposición de datos

Un ejemplo de aplicación completamente paralelizable es el cálculo del área bajo una curva por integración numérica. Basta con dividir el intervalo de integración entre todos los procesadores disponibles y que cada uno resuelva su fragmento sin preocuparse de qué hacen los demás. Al final, los resultados parciales se recolectan y se suman convenientemente. Con n procesadores es posible resolver el problema n veces más rápido que haciendo uso de uno sólo (salvo por el mínimo retraso que supone el reparto de trabajo inicial y la recolección de datos final), consiguiendo una aceleración lineal con el número de procesadores. Si las condiciones son muy favorables es incluso posible alcanzar la aceleración superlineal, consistente en que el programa se ejecuta aún más rápido que en régimen lineal. La aparente paradoja se da debido a que cada procesador cuenta con su propia memoria ordinaria y caché, que pueden ser usadas de forma más eficiente con un subconjunto de datos. De hecho, es posible que el problema no se pueda resolver en un único procesador pero sí sobre un conjunto de ordenadores debidamente configurados, simplemente por cuestión de tamaño de los datos.

Descomposición funcional

Un modelo computacional se basa por empezar, en que una aplicación consiste en varias tareas. Cada tarea es responsable de una parte de la carga de procesamiento de la aplicación en general y a su vez, cada tarea realiza una operación independiente de las otras tareas. Los algoritmos de cada tarea son

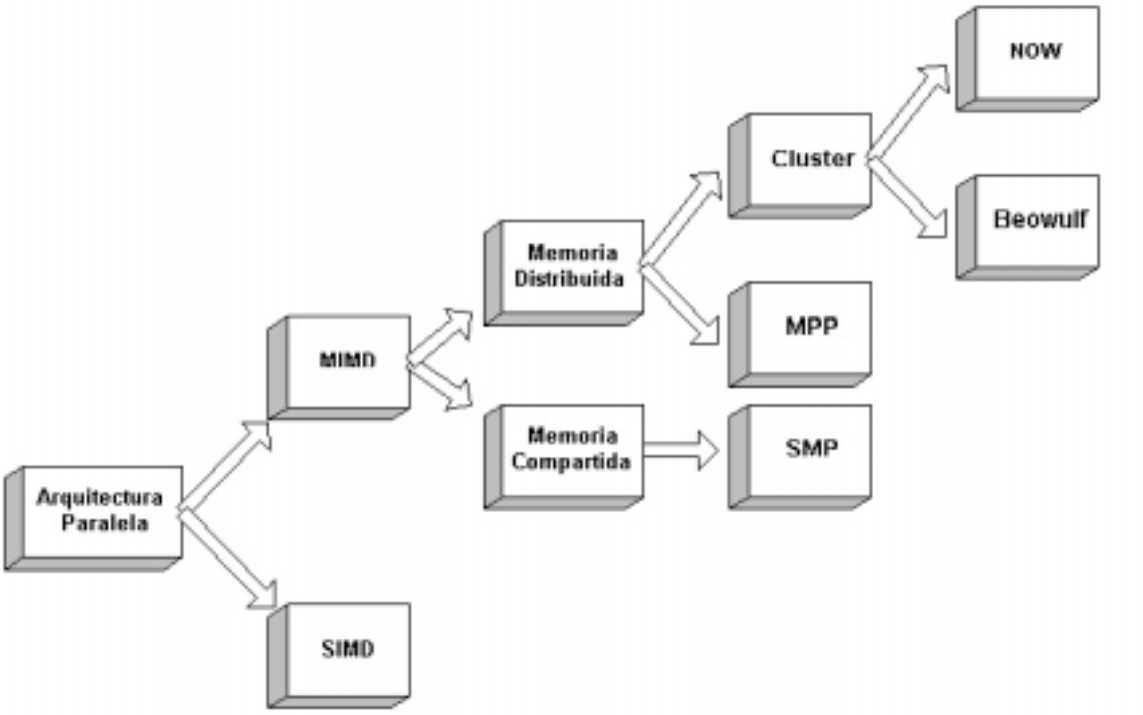
diferentes. Este modelo se denomina descomposición funcional y se puede aprovechar las características particulares de cada tipo de tarea para ejecutarlas en la maquina que sea mas conveniente a tal efecto.

Arquitecturas Paralelas

A diferencia de un procesador serial, en el cual se ejecuta una instrucción por vez sobre un único dato, las arquitecturas paralelas operan en forma simultánea sobre un conjunto de datos y puede ejecutarse una misma instrucción en cada procesador como también el conjunto de CPUs ejecutar distintas instrucciones.

El procesamiento paralelo se clasifica en dos tipos:

- SIMD (Single Instruction, Múltiple Data)
- MIMD (Multiple Instruction, Multiple Data)



Taxonomía de las Arquitecturas Paralelas

Arquitectura SIMD (Simple Instruction Multiple Data)

En el modelo SIMD cada procesador ejecuta la misma operación que los demás procesadores, pero sobre un conjunto particular de datos propios. El conjunto de los procesadores están siempre sincronizados. Con respecto a la memoria, ésta no

se comparte, por lo que si un procesador necesita conocer algún dato generado por otro procesador, debe efectuarse una comunicación explícita. En este esquema, la implementación más frecuente es la presencia de una gran cantidad de procesadores con poca capacidad, cada uno de los cuales dispone de una cantidad limitada de memoria local y canales de comunicación para interconectarse con el resto de los procesadores. En las implementaciones de arquitecturas SIMD siempre se requiere una estación de trabajo que actúe como controlador.

Arquitectura MIMD (Múltiple Instruction Múltiple Data)

En el modelo MIMD cada procesador actúa en forma independiente, realizando cada uno de ellos un trabajo específico sobre una parte de la tarea completa. Esto es lo que se denomina descomposición de control, donde cada procesador puede estar ejecutando un programa distinto. Mucho más habitual es la descomposición de datos en la que los datos del problema se reparten entre los distintos procesadores y todos ejecutan el mismo programa sobre su fragmento de datos. Esta versión restringida del modelo MIMD se denomina SPMD (Single Program, Múltiple Data), donde la diferencia con el modelo SIMD reside en que cada procesador que ejecuta un programa SPMD puede fluir a través de ramas de código diferentes, resultando un modelo mucho más flexible. Para lograr este efecto, se debe recurrir a la programación de hebras (threads). Esencialmente en el modelo MIMD los datos se intercambian entre procesadores mediante memoria compartida o por medio del paso de mensajes. Los procesadores de los sistemas con memoria compartida comparten físicamente la memoria, accediendo todos al mismo espacio de direcciones. A la combinación de MIMD y memoria compartida se le denomina SMP (Symmetric Multi-Processing). En el caso de utilizar el paso de mensajes, los sistemas están compuestos por procesadores que disponen cada uno de su propia memoria independiente de la del resto. Existen herramientas de software que permiten programar una arquitectura de memoria distribuida como si fuera de memoria compartida, sin preocuparse de en qué procesador reside cada dato. A este modelo se lo denomina de memoria compartida virtual. Los sistemas con memoria distribuida pueden ser un único “ordenador” con múltiples CPUs comunicadas por un bus de datos o bien múltiples ordenadores, cado uno con su procesador, enlazados por una red de datos. Más comúnmente se implementan los sistemas constituidos por varios ordenadores conectados entre sí, lo que de forma genérica se conoce como cluster (grupo) de estaciones de trabajo. Las implementaciones de las arquitecturas MIMD suelen constituirse por procesadores potentes y grandes cantidades de memoria.

Memoria compartida

El concepto de memoria compartida implica la existencia de múltiples procesadores accediendo a un espacio de direcciones de memoria único. Ese espacio de memoria puede estar compuesto de hecho por diferentes memorias físicas o por una única memoria centralizada. La clave de esta arquitectura –por oposición a la memoria distribuida pura basada en el paso de mensajes– es la transparencia para el acceso y uso de la memoria.

SMP (Symmetric Multiprocessors)

Un sistema de procesamiento paralelo SMP es aquel que resulta de la combinación de la arquitectura MIMD y memoria compartida. Esta implementación es considerada como una arquitectura compartida (shared-everithing). En estos sistemas, todos los procesadores comparten todos los recursos disponibles (bus, memoria, sistemas de entrada/salida), siendo la principal característica que cada procesador tenga una vista global de toda la memoria. En estos sistemas, se ejecuta una única copia del sistema operativo.

Memoria distribuida

El concepto de memoria distribuida es representado técnicamente por el paso de mensajes. Las librerías de paso de mensajes permiten desarrollar eficientes programas paralelos para sistemas de memoria distribuida. Estas librerías proveen rutinas para inicializar y configurar un ambiente de mensajes así como también proveen las facilidades para el intercambio de paquetes de datos. Actualmente, los dos sistemas de alto nivel más populares para el paso de mensajes para aplicaciones científicas y de ingeniería son PVM (Parallel Virtual Machine) y MPI (Message Passing Interface).

MPP (Massively Parallel Processor)

Un sistema de procesamiento paralelo MPP es generalmente un gran sistema de procesamiento paralelo con una arquitectura no compartida (nothing-share). Típicamente consiste de varios cientos de procesadores (nodos), los cuales están interconectados mediante una red de conmutación de alta velocidad. Cada nodo puede tener una variedad de componentes de hardware, pero comúnmente consisten de una memoria principal y de uno o más procesadores. Nodos especiales pueden además tener periféricos conectados, tales como discos o sistemas de resguardo (backup). Además, cada nodo ejecuta una copia individual de un sistema operativo.

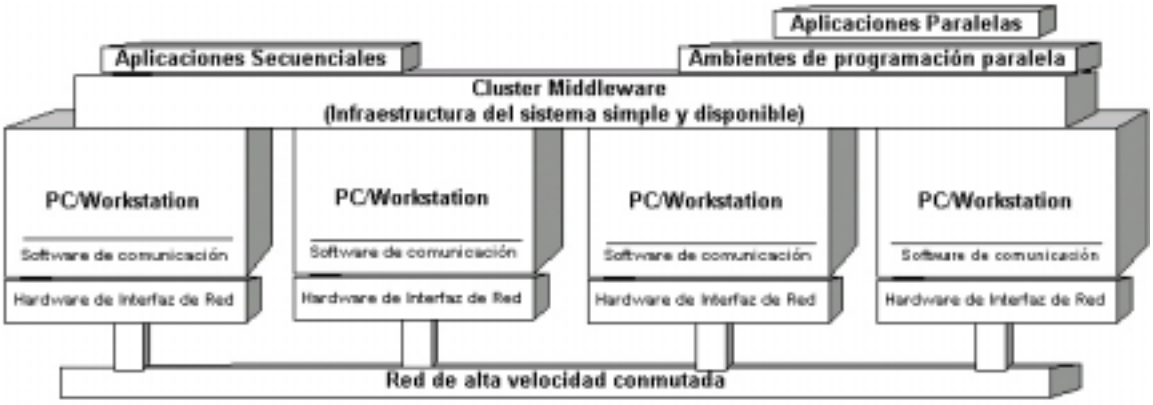
Cluster

Un cluster es un tipo de procesamiento paralelo o distribuido que:

- Consiste de un conjunto de computadoras conectadas a una red
- Es utilizado como un recurso computacional simple e integrado

Un nodo puede ser un sistema simple o multiprocesador (PCs, workstations, o SMPs) con memoria, cualidades de entrada/salida, y un sistema operativo. Un cluster generalmente se refiere a dos o más computadoras (nodos) conectados juntos. Un nodo puede existir en una carcaza simple o pueden estar físicamente separados y conectados mediante una red LAN. Un cluster de computadores

interconectados (basado en una red LAN) puede aparecer como un sistema simple para los usuarios y las aplicaciones.



Arquitectura de un computador Cluster

Los nodos en un cluster se comunican mediante redes de alta velocidad utilizando protocolos de comunicación tales como TCP/IP o protocolos de bajo nivel como Mensajes Activos (Actives Messages). En la mayoría de las implementaciones es común encontrar que las interconexiones se realizan mediante la tecnología Ethernet. En términos de performance (tales como latencia y ancho de banda), esta tecnología está llegando a su límite. Sin embargo, Ethernet es una tecnología conocida y fácil de implementar y aporta facilidades con respecto a compartir archivos e impresoras. Una conexión Ethernet simple no puede ser usada seriamente como base de la computación basada en cluster; la latencia de la red y el ancho de banda que aportan estas redes no están balanceadas comparadas con el poder computacional de las estaciones de trabajo actuales.

Existen en el mercado una variedad de tecnologías de interconexión entre las que se puede mencionar:

- Ethernet, Fast Ethernet y Gigabit Ethernet
- Asynchronous Transfer Mode (ATM)
- Scalable Coherent Interfaces (SCI)
- Myrinet

Los clusters ofrecen importantes ventajas sobre otros tipos de arquitectura paralela:

- Cada una de las máquinas de un cluster puede ser un sistema completo utilizable para otros propósitos. Por ejemplo, se puede montar un aula de informática con estaciones que den servicio a los alumnos durante el día y disponer del conjunto durante la noche para realizar cálculos complejos, siendo incluso posible usar parte de la CPU que los alumnos no precisen (“ciclos muertos”) para continuar los cálculos durante el día.

- Los elementos de proceso de un cluster son ordenadores "normales" y por lo tanto baratos. El hardware de red, es también cada vez más barato. La economía puede alcanzar puntos más altos, teniendo un único monitor, tarjeta de video y teclado para todo el cluster.
- Los cluster escalan bien hasta sistemas muy grandes. No es difícil construir clusters con cientos y hasta miles de ordenadores.
- Reemplazar un ordenador defectuoso de un cluster es trivial si se compara con el trabajo necesario para reparar una máquina SMP. Esto permite tener sistemas de muy alta disponibilidad, siendo posible incluso diseñar un cluster de tal forma que si un nodo falla el resto continúe trabajando.
- Existe mucho soporte software para la programación de clusters. Con el nivel de estandarización actual existe la garantía de que los programas escritos para un cluster funcionarán en cualquier otro con independencia del tipo de procesador de cada nodo.

Sin embargo, los clusters también presentan un conjunto de desventajas:

- Las redes ordinarias no están diseñadas para el procesamiento en paralelo. La latencia es alta y el ancho de bando relativamente bajo si se comparan con los de un sistema SMP. Si el cluster no está aislado del resto de la red, la situación es aún peor.
- Existe muy poco soporte software para tratar un cluster como un sistema único. Por ejemplo, el comando *ps* sólo lista los procesos de un sistema Linux, es decir sólo lista los procesos locales, no los de todo el cluster.

Existen dos tipos de clusters, dependiendo de si cada ordenador del cluster está o no exclusivamente dedicado a él. Si es así se habla de un cluster de clase Beowulf, y si no de una simple red de estaciones de trabajo (NOW, Network Of Workstations). Los Beowulfs tienen algunas ventajas sobre las redes de estaciones:

- Al estar todas las CPUs al servicio del cluster es más fácil mantener el balance de carga. Esto significa lograr que todos los procesadores tengan un grado de ocupación semejante. Además, el único uso de la red es debido a la aplicación paralela que se está ejecutando, lo que permite sincronizar el tráfico y disminuir así la latencia. Al ser un entorno más restrictivo, los programas diseñados para NOWs no tienen problemas para ejecutarse en Beowulfs, pero lo contrario puede no ser cierto.
- En un Beowulf es posible modificar el núcleo para mejorar el procesamiento en paralelo. En Linux es habitual utilizar un módulo que permite la asignación a cada proceso de un identificador global a todo el cluster.
- Un ordenador de una red ordinaria debe estar configurado para una respuesta interactiva adecuada al usuario pero en un Beowulf se pueden ajustar los parámetros de sistema de acuerdo a la granularidad de la

aplicación paralela que se vaya a ejecutar, lo que contribuye a mejorar su rendimiento.

Comparación de características entre SMP, MPP y Clusters

Característica	SMP	MPP	Cluster
Número de nodos	2-10 (procesadores)	2-100	100 o menos
Complejidad por nodo	Media o alta	Baja o media	Media
Comunicación entre nodos	Media o alta	Baja o media	Media
Planificación de las tareas	Una única cola de trabajo	Una única cola de trabajo corriendo en un nodo	Múltiples colas de trabajo pero coordinadas
Soporte para SSI (apariencia de sistema único)	Siempre	Parcialmente	Deseable
Tipo y número de SO por nodo	Un único SO monolítico	Numerosos micro-kernel monolíticos	Numerosas plataformas de SO

NOW (Network Of Workstations)

En la década de 1980 se creía que el mejor rendimiento de las computadoras se lograba creando procesadores más rápidos y eficientes. Esta idea fue alterada por el procesamiento paralelo, el cual en esencia significa unir dos o más computadoras con el fin de resolver en conjunto algún problema computacional. Desde principios de la década de 1990 se presenció un incremento en las tendencias de migrar los sistemas de procesamiento desde súper-sistemas de procesamiento paralelo costosos, especializados y propietarios hacia redes de estaciones de trabajo (workstations). Esta tecnología está logrando que redes de computadoras se transformen en sistemas de procesamiento paralelo.

A continuación se enumeran algunas de las ventajas que hacen de las redes de estaciones de trabajo preferibles sobre computadoras paralelas especializadas:

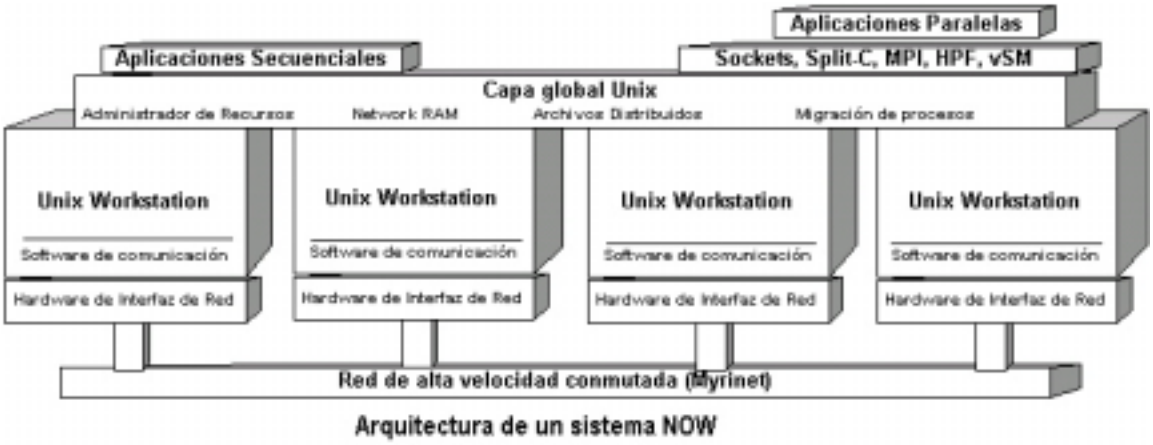
- Las estaciones de trabajo están logrando mayor poder de cómputo, incrementando dramáticamente la performance. En los últimos años, el poder de procesamiento se duplicó en períodos de 18 a 24 meses. Se pronostica que esta tendencia se mantendrá por varios años, con procesadores más rápidos y máquinas multiprocesadores más eficientes.

- El ancho de banda en las comunicaciones entre las estaciones de trabajo va en aumento y la latencia va en disminución como resultado de las nuevas tecnologías de redes y la implementación de nuevos protocolos.
- Nuevas estaciones de trabajo son fáciles de agregar a un grupo de estaciones para procesamiento paralelo existentes.
- Las estaciones de trabajo personales típicamente tienen baja utilización.
- Las herramientas de desarrollo para estaciones de trabajo están más maduras comparadas con las soluciones ofrecidas por las empresas que hacen desarrollo de soluciones computacionales paralelas propietarias, principalmente siendo estas últimas no estandarizadas.

Claramente, los ambientes de estaciones de trabajo están mejor diseñado para aquellas aplicaciones que no se centran intensivamente en las comunicaciones, ya que típicamente las redes LAN tienen altos retardos y bajo ancho de banda. Si una aplicación requiere altas performance en comunicaciones, las arquitecturas existentes de redes LAN, como Ethernet, no son capaces de proveer dicha performance.

Los recursos disponibles en la mayoría de los NOW, tales como procesadores, interfaces de red, memoria y discos rígidos, ofrecen un gran número de oportunidades de desarrollo, tales como:

- Procesamiento paralelo: utilizar los múltiples procesadores para construir sistemas para la computación paralela.
- Network RAM: utilizar la memoria asociada a cada estación de trabajo como una única memoria agregada; esto mejora las implementaciones de memoria virtual y la performance de los sistemas de archivos.
- Software RAID (Redundant Array Of Inexpensive Disks): utilizar los discos de las estaciones de trabajo para proveer almacenamiento de datos económico, altamente disponible y escalable, utilizando el almacenamiento redundante en las estaciones de trabajo y la red LAN como backplane de entrada/salida.
- Comunicación multipath: utilizar las múltiples redes para transferencia de datos en paralelo entre los nodos.



Arquitectura de un sistema NOW

Las aplicaciones paralelas requieren un buen rendimiento en operaciones de punto flotante, comunicaciones con baja latencia y gran ancho de banda escalable, y acceso rápido a los archivos. El software asociado a los cluster pueden lograr estas metas utilizando los recursos asociados a los clusters. Un sistema de archivos que soporte entrada/salida paralela puede ser construido utilizando los discos asociados a cada estación de trabajo en lugar de utilizar un costoso hardware que implemente RAID. El rendimiento de la memoria virtual puede ser mejorado utilizando Network RAM como medio de almacenamiento temporal en lugar de discos rígidos. Es muy común conectar los nodos de un cluster mediante redes Ethernet estándares o redes especializadas de alto rendimiento tales como Myrinet. Estas redes pueden ser utilizadas para la transferencia de datos en forma simultánea a través de los nodos de un cluster. El software de comunicación multipath logra la demultiplexación de los datos en el extremo transmisor a través de múltiples redes y multiplexa nuevamente los datos en el extremo receptor. Esto permite una mayor velocidad en la comunicación de los datos entre los nodos de un cluster.

BEOWULF

Beowulf es una arquitectura que permite utilizar múltiples computadoras para realizar computación paralela. Consiste usualmente de un nodo servidor y uno o más nodos cliente conectados entre sí. El nodo servidor controla el cluster completo y además sirve de file system para los otros nodos y de punto de comunicación con el mundo exterior al cluster. Los nodos cliente son del tipo minimalistas (tiene cada uno de ellos lo mínimo para funcionar), ligados por un medio de comunicaciones barato, en el que la topología de la red se ha diseñado para resolver un tipo de problema específico. Los nodos son ciertas partes de una máquinas que no son capaces de ejecutar por sí solas ni siquiera el sistema operativo. Habitualmente, constan solamente de una placa madre, una CPU, las memorias y algún dispositivo de comunicaciones. Pueden tener o no un disco duro pequeño para arrancar el sistema operativo remoto.

El proyecto Beowulf fue creado para investigar el potencial de un grupo de PCs para realizar tareas computacionales. Beowulf se refiere a una pila de PCs (Pile-of-PCs – PoPC) que describen un grupo de PCs, que tienen características similares a las redes de estaciones de trabajo (Network Of Workstations - NOW). Beowulf no es un paquete de software especifico, sino una tecnología para armar un cluster de varias máquinas Linux para conformar una supercomputadora virtual aprovechando el procesamiento paralelo. PoPC enfatiza en el uso de componentes de hardware de venta masiva o lo que se denomina "commodity hardware" (no debe contener ninguna pieza especializada de hardware), procesadores dedicados (en vez de aprovechar los ciclos inutilizados de las estaciones de trabajo) y la utilización de redes de comunicación privadas. El objetivo principal de Beowulf es lograr la mejor relación costo/rendimiento para un cluster.

Un cluster Beowulf se diferencia de un cluster de Workstations (NOW) en que el cluster Beowulf se comporta como una maquina única hacia el exterior.

El conjunto de herramientas de software de desarrollo dentro del proyecto Beowulf es conocido como *Grendel*. Estas herramientas son los recursos de administración y soporte para aplicaciones distribuidas. La distribución Beowulf incluye varios ambientes de programación y librerías de desarrollo como paquetes separados. Estos incluyen PVM (Parallel Virtual Machine), MPI (Message Passing Interface) y BPS (Beowulf Process Space).

Las topologías físicas de los Beowulf con frecuencia responden a estructuras de anillo (cada nodo tiene dos dispositivos de comunicaciones, que se unen en anillo en árbol, y existen nodos más potentes que tienen varios dispositivos de comunicaciones, y los nodos terminales un solo dispositivo de comunicaciones que se comunica con uno de los nodos con más de un dispositivo de comunicaciones), y los programas se diseñan de forma que coincida su estructura lógica de paralelización con la estructura física del Beowulf.

La comunicación entre procesadores de un Beowulf es a través del protocolo de comunicación TCP/IP sobre redes Ethernet. El rendimiento de las comunicaciones entre procesadores está limitada por las características de rendimiento de la red Ethernet y del software del sistema administrador del paso de mensajes. Beowulf ha estado explorando la viabilidad de utilizar múltiples redes Ethernet en paralelo para satisfacer los requerimientos de ancho de banda para la transferencia de datos interna. Cada estación Beowulf es transparente para el usuario en cuanto al acceso a múltiples redes Ethernet paralelas. Esta arquitectura es conocida como *"channel bonding"* y es una técnica de mejora implementada por los kernel del sistema operativo Linux. El proyecto Beowulf demostró que pueden ser utilizadas juntas hasta tres redes, con una significativa mejora en el rendimiento. Nuevas tecnologías como Fast Ethernet, mejorarán sin lugar a dudas el rendimiento de las comunicaciones entre procesadores.

Un programa desarrollado teniendo en cuenta la topología del Beowulf es más rápido que el equivalente en una red de computadores configurados como NOW, por la ausencia de colisiones, y el hardware resulta mucho más barato. Existen supercomputadores de clase Beowulf que emplean switches, routers o dispositivos

similares a os que emplean las redes de computadores, solo que son mucho más caros y no aseguran un mejor rendimiento.

La programación es fuertemente dependiente de la arquitectura; y pueden ser empleados como bibliotecas Sockets, PVM o MPI. Una característica común a los Beowulf es que no emplean mecanismos de memoria compartida entre nodos; la comunicación, pues, siempre se hace mediante paso de mensajes.

Para lograr la imagen de un sistema uniforme tanto para el usuario como para las aplicaciones, Beowulf extendió el kernel de Linux para permitir el ensamblado de los nodos que participan en un único espacio de nombres global. En un esquema distribuido es conveniente para los procesos tener un PID (Process Identification) que sea único para todo el cluster. Beowulf implementa 2 esquemas de identificadores de procesos globales (Global Process ID – GPID). El primero es independiente de librerías externas. El segundo, denominado GPID-PVM, está diseñado para ser compatible con el formato de los identificadores de tareas de PVM (PVM Task ID). A pesar de la existencia de estos paquetes de software que se pueden unir para armar un cluster Beowulf (modificaciones al kernel de Linux, software PVM y librerías MPI), en teoría se puede armar desde cero un cluster Beowulf con sólo 2 máquinas Linux conectadas vía Ethernet, con el /home file system compartido vía NFS y cada una con derechos de ejecutar shell remotos (rsh).

Beowulf vs. Red de Estaciones de Trabajo (NOW)

Enumeraremos algunas características que diferencian a Beowulf de las NOWs:

- **Exclusividad**: cada nodo de un Beowulf se dedica exclusivamente a procesos del supercomputador. La red de estaciones de trabajo, no. En esta última, cada nodo puede ejecutar simultáneamente los procesos correspondientes al NOW y cualquier otro programa mono-procesador. Cuando queremos usar una red de estaciones NOW lo hacemos desde cualquiera de ellas; cuando queremos usar un Beowulf lo tenemos que hacer a través de un terminal tonto conectado al Beowulf, no siendo posible acceder directamente a la consola de los nodos.
- **Independencia del nodo**: cada nodo de una red de estaciones de trabajo NOW es una estación de trabajo completamente funcional que, con el software apropiado, puede funcionar sin necesidad de participar del sistema distribuido. En un Beowulf, no; cada nodo tiene el hardware mínimo que le permite funcionar como unidad de cálculo. Los nodos carecen de monitores, teclados, disqueteras, ratones; muchas veces de tarjetas de video y hasta de disco duro. Y es frecuente que un Beowulf los nodos no tengan carcasas, y se compongan de una estructura metálica que va a sujetar las fuentes de alimentación y las placas madre.
- **Mecanismo de comunicaciones**: en una red de estaciones NOW suele haber un switch central, que es en gran parte el responsable del rendimiento por evitar o no las colisiones. En un Beowulf el mecanismo de comunicaciones es más rudimentario: conexiones placa a placa por cable UTP cruzado con conectores RJ-45.

- **Direccionamiento**: las redes NOW suelen ser todas igualmente accesible desde fuera, por lo que todas suelen tener direcciones IP válidas, o bien todas tienen direcciones IP privadas pero acceden a un NAT exterior a la red. En un Beowulf, en cambio, suele haber un nodo destacado que tiene un dirección IP válida, o privada mas que accede a un NAT que la decodifica, mientras que el reto de los nodos tienen direcciones de red privadas hasta para la propia subred, de forma que no es posible acceder directamente a un nodo del Beowulf que no se el nodo destacado.
- **Servicios de los nodos**: en una red de estaciones NOW habitualmente podemos hacer FTP, Telnet y todos los comandos habituales de red a todos los nodos. En un Beowulf, parte o todos estos comandos pueden estar inhabilitados entre los nodos.
- **Topología**: una red NOW puede tener prácticamente cualquier topología. Para programar, la red NOW abstrae la topología, por lo que ésta da igual. En un Beowulf las cosas van al revés: se diseña el modelo de paralelismo, se ve como van a ser las comunicaciones entre los nodos y luego se implementan físicamente, por lo que se evita el hardware innecesario a cambio de una fuerte dependencia entre software y topología de la red. En caso de que cambie la aplicación, se puede mantener el mismo modelo de paralelismo o rehacer la parte física de la red y las tablas de enrutamiento de las máquinas.
- **Conocimientos del usuario**: en una red NOW el usuario no tiene por qué saber nada acerca de la máquina, ni como configurarla, ni como montarla. Habitualmente, el usuario y el administrador son dos personas distintas, que no tienen ni por qué conocerse. En un Beowulf el equipo de desarrollo del programa (o el usuario en caso de ser solo uno) tienen que saber desde diseñar una topología de red y hacer aplicaciones paralelas hasta soldar cables, montar conectores y montar y desmontar ordenadores.
- **Número de usuarios**: una red de estaciones de trabajo suele tener un número determinado de usuarios que forman un grupo heterogéneo, con una cuenta por usuario, un administrador, que puede no usar la máquina, y un conjunto de problemas indeterminado de cada usuario por resolver. Un Beowulf se desarrolla para resolver un problema; habitualmente sólo resuelve un mismo tipo de problemas al mismo tiempo, tiene un usuario que al mismo tiempo opera con la máquina y la administra (habitualmente, un gurú en redes y Linux dentro de un grupo que investiga en algo no relacionado con la informática), y un grupo de expertos que da los problemas para que los resuelva dicho gurú.

Existen también una serie de pequeños detalles de importancia, como son que el espacio de PIDs es a nivel de red en el Beowulf, mientras que en una red de estaciones de trabajo cada máquina tiene su propio espacio de PIDs independiente; al mismo tiempo, existen una serie de llamadas al sistema operativo en un Beowulf que no existen en Unix estándar, y que permiten a un proceso controlar más parámetros del sistema operativo.

Concluyendo, una red de ordenadores NOW es una solución rápida de configurar, fácil de programar, de una potencia computacional buena y que no exige demasiados conocimientos a los usuarios que operan con las máquinas. Un

Beowulf exige gran cantidad de conocimientos y dedicación para puesta en marcha, desarrollo y ejecución; a cambio, su rendimiento es realmente impresionante, en comparación al bajo costo necesario; y el usuario-operador se encarga de tener todo optimizado para el problema del grupo científico.

Clusters tipo Beowulf

En esta sección, explicaremos mas en detalle las alternativas disponibles para el armado de un cluster del tipo Beowulf. Nos basaremos en las alternativas de software mas populares y de disponibilidad masiva. Esperamos que esta sección sea de utilidad para la aplicabilidad de los conceptos de clustering explicados anteriormente sobre todo en entornos Linux que es el mas popular donde se pueden emplear estos conceptos.

Clasificación

Para distinguir las diferentes implementaciones, los clusters Beowulf se clasifican en dos clases:

CLASE I:

- Hardware de distintos proveedores
- Los drivers de disco, red y actualizaciones al kernel deben ser los disponibles para Linux
- Basado en estándares: SCSI, Ethernet, etc.

Los cluster clase I serían los mas económicos, para mayor performance aparecen especializaciones en la clase II:

CLASE II:

- Usualmente armados por un único proveedor de hardware
- Puede tener algunas piezas de software también desarrolladas por el proveedor y no disponibles en la comunidad Linux.
- Los drivers pueden estar específicamente desarrollados para el propósito de clustering y combinación de hardware y software utilizados

Estos cluster pueden ser menos económicos que armar uno de clase I pero la performance se vería incrementada. Aún así, poseen una ventaja importante en relación precio/prestaciones contra grandes servidores que no se valen de clustering.

Software Beowulf

Existen varios componentes de software utilizados para construir un cluster Beowulf. Nos interesan en este trabajo aquellas que constituyen la base para el

procesamiento paralelo y son indispensables para el armado de un cluster Beowulf CLASE I en Linux. Los componentes que describiremos son:

- PVM: Parallel Virtual Machine
- MPI: Message Passing Interface
- BPROC: Implementación de BPS: Beowulf Distributed Process Space

PVM: Parallel Virtual Machine

PVM es un componente de software que permite básicamente dos cosas:

- La ejecución de procesos en máquinas remotas
- El paso de mensajes entre estos procesos

PVM provee un entorno unificado para el desarrollo de programas de procesamiento paralelo. En cuanto a la transparencia, podemos decir que PVM independiza la implementación del programa del hardware subyacente e incluso de la plataforma (existen implementaciones de PVM para Linux, Unix e incluso Windows). Además provee independencia de la estructura del cluster, esto es, permite que los programas desarrollados en un cluster con una determinada estructura, se ejecuten en otro cluster con distinta configuración de estaciones de trabajo asociadas.

Además, provee una API para la creación de procesos remotos, el paso de mensajes entre los procesos y la posibilidad de agregar y quitar nodos del cluster dinámicamente.

Podemos resumir las características del PVM en los siguientes puntos:

- Pool de máquinas configurable, es decir se permite agregar y quitar maquinas al cluster
- Acceso translúcido al hardware: las aplicaciones pueden acceder al hardware (local o remoto) como atributos de colecciones de elementos virtuales ó explotar las características de componentes específicos posicionando tareas específicas en maquinas que se sabe serán mas aptas para ello.
- Computación basada en procesos: la unidad de paralelismo en PVM es la “tarea” (que puede corresponder o no a un proceso Linux). La tarea se define como un hilo de control independiente que alterna entre cálculo y comunicación (I/O).
- Modelo de paso de mensajes explícito: las tareas pueden comunicarse entre si pasándose mensajes explícitamente, utilizando las API provistas por PVM. El tamaño de los mensajes sólo está limitado por el tamaño de la memoria.
- Heterogeneidad: soporta diferentes redes, tipos de procesador (simples o múltiples en una misma máquina) y diferentes tipos de datos para el paso de mensajes.

Arquitectura de PVM:

PVM esta compuesto de 2 piezas de software:

- La primera es el PVM Daemon llamado *pvmd3* o simplemente *pvmd.* Este componente debe correr en cada máquina del cluster. Cada set de daemons corriendo en distintas máquinas y comunicándose entre sí constituyen una máquina virtual.
- La segunda pieza es la librería de API de PVM, que debe ser utilizada para el desarrollo de aplicaciones paralelas o la paralelización de aplicaciones existentes.

La unidad de procesamiento distribuible en PVM es la tarea (Task). PVM asigna a cada tarea –independientemente de en qué máquina física se ejecute—un número o Task Identifier (TID). Ese número es único en toda la máquina virtual.

PVM API

Inicialmente PVM se diseñó para soportar C, C++ y Fortran. La API para C está puramente implementada como funciones pero existen librerías que se valen de estas funciones para proveer un entorno de programación orientado a objetos en C++. En Fortran se proveen subrutinas. Como parte de la distribución de PCM se encuentra el archivos de librería *libpvm3.a* para enlazar con los programas C o C++ y *libfpvm3.a* para Fortran.

Las funciones relacionadas con la creación de procesos y paso de mensajes que proveen estas librerías son:

- pvm_spawn: permite crear un proceso en una maquina remota del cluster, simplemente pasando como parámetro un string con el nombre del proceso y el TID del proceso actual que pasa a ser padre del proceso creado.
- pvm_mytid: obtiene el TID del proceso actual.
- pvm_send: envía mensajes a otro proceso.
- pvm_recv: espera y bloquea hasta recibir un mensaje de un proceso

Paradigmas de programación paralela aplicables

Si bien el modelo para la creación y administración de procesos es definible y dependiente de la aplicación que utilice PVM y no todos los modelos son puristas, existen 2 modelos básicos útiles para implementar procesamiento paralelo:

Modelo estrella: un proceso central coordinador deriva la realización de tareas independientes a otros procesos remotos. El proceso coordinador es el único que puede crear procesos remotos y el calculo solo lo realizan las tareas instanciadas.

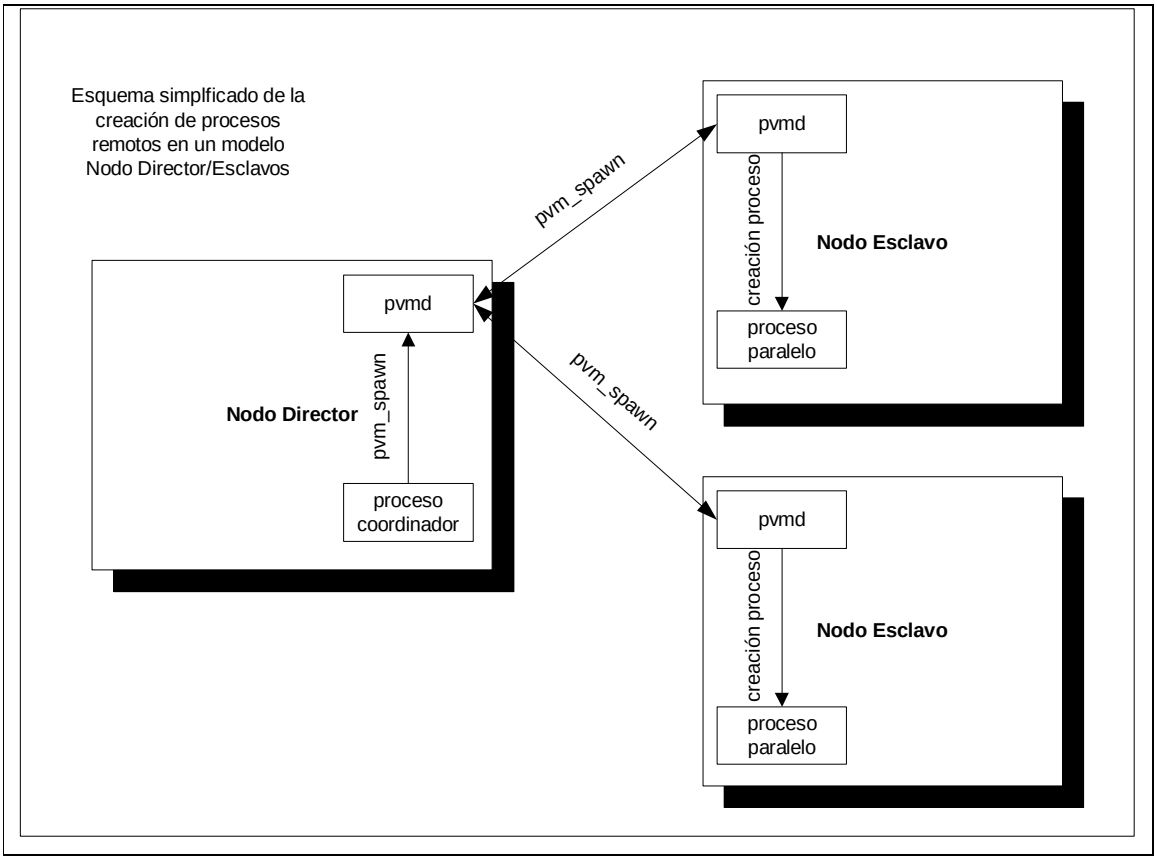
Modelo árbol: un proceso padre instancia tareas en paralelo. A su vez, cada una de estas tareas puede instanciar a otras remotamente para realizar partes paralelas de la misma.

Cabe destacar, que dependiendo de la aplicación pueden darse combinaciones de los modelos anteriores, por ejemplo una estrella que dispare procesos en nodos remotos por descomposición funcional. Luego uno de los nodos llamados, puede implementar un algoritmo de búsqueda por descomposición de datos que aproveche el modelo de árbol.

PVM como software para un cluster tipo Beowulf

Esta clasificación esta realizada desde el punto de vista de los procesos en sí, independientemente de la topología de la red involucrada. Para un cluster Beowulf, podemos utilizar PVM para construir un modelo estrella haciendo correspondencia 1 a 1 con la estructura de Beowulf:

- El primer paso es tener instalado *pvmd* en todas las maquinas del cluster y configurarlos de modo de armar una máquina virtual.
- En el nodo servidor o director de Beowulf, puede correr el proceso coordinador
- El proceso coordinador instancia tareas especificas en cada nodo cliente del cluster Beowulf utilizando la función *pvm_spawn*



El modelo estrella se adecua perfectamente y es el mas simple para la estructura de un cluster Beowulf, incluso, el nodo director podría ser el que determina qué maquinas del cluster se encuentran disponibles, agregar y quitar nodos de la máquina virtual PVM y derivar las tareas de cálculo a los nodos que crea convenientes.

Como vemos, PVM provee la funcionalidad necesaria en software para el armado de un cluster Beowulf simple.

BPROC: Beowulf Distributed Process Space

Si bien PVM provee una base para la distribución de procesos que trabaja a nivel de usuario y se puede utilizar desde las aplicaciones por medio de una API, en Linux hubo otro approach a la paralelización de procesos que proveía mayor trasparencia haciendo modificaciones al Kernel.

BPROC, o Espacio de Procesos Distribuido Beowulf, es una implementación de BPS (Beowulf Process Space) que incluye un conjunto de modificaciones al kernel de Linux que proveen la funcionalidad para paralelizar procesos en distintas máquinas.

Proceso fantasma (Ghost Process):

Básicamente BPROC crea un ID de proceso distribuido (PID) único en todo el cluster y posibilita que un proceso corra en otro nodo y parecer al usuario como si estuviera corriendo localmente. Para implementar esta apariencia de proceso local se introdujo el concepto de proceso fantasma o ghost process.

Un proceso fantasma corre en el nodo maestro de un cluster Beowulf. Este aparece como un thread del kernel. No tienen espacio de memoria, archivos abiertos o contexto de file system pero puede reaccionar a las señales o eventos a las que reacciona cualquier otro proceso y hacer algo en el kernel. Por cada proceso fantasma en el nodo servidor existe un proceso real corriendo en un nodo esclavo del cluster. El proceso fantasma entonces realiza lo siguiente:

- Las señales que recibe son reenviadas al proceso real que representa. Como trabajan a nivel kernel pueden capturar incluso las señales SIGKILL y SIGSTOP y parar el proceso real que representan antes de destruirse a si mismos.
- Cuando el proceso remoto finaliza, el proceso fantasma recibe el código de finalización y finaliza con ese mismo código.

Enmascaramiento de los PID

Para proveer la funcionalidad de procesos reales y fantasmas se modificaron las llamadas del kernel relacionadas a los process ID (getpid(), getppid(), fork() kill(), wait()) para manejar los procesos fantasma.

También se introduce un daemon para controlar las operaciones de asignación de PID normalmente realizadas por el kernel. Por ejemplo, para asignar un nuevo PID es necesario que se asigne uno único en el cluster no solo en el nodo maestro, entonces la creación de procesos remotos es manejada por este daemon. Básicamente lo que se hace es crear primero el proceso fantasma en el nodo maestro del cluster y utilizar el PID asignado por el sistema operativo para crear el proceso real en el nodo remoto. Como solo puede haber un master por cluster se garantiza la unicidad de los PID.

Inicio de procesos

Hay 2 formas básicas de iniciar un proceso en el esquema BPROC:

- La forma simple es utilizando *rexec* (remote execute) al cual se le debe indicar además del proceso a iniciar y otros parámetros, un número de nodo. Se requieren binarios y librerías instalados en la maquina master y el nodo remoto.
- La otra interface que BPROC provee para iniciar un proceso remoto es a través de *move* o *rfork*. Esto graba la región de memoria de un proceso que corre localmente y la recrea en el nodo remoto. Este tiene la ventaja de no requerir la instalación del binario y librerías en el nodo remoto aunque incrementa el overhead de comunicación al crear un proceso.

Comparación con PVM

Si bien BPROC es mas transparente en general que PVM ya que se puede tratar a un proceso remoto como si fuera local, su alcance esta limitado por que implica modificaciones complejas al kernel. La idea y el concepto es bueno pero la aplicabilidad de software del tipo PVM otorga mas flexibilidad y garantías de funcionamiento mejores en otras versiones de Linux.

MPI: Message Passing Interface

Si bien vimos que con PVM se puede obtener el software mínimo para la construcción de una aplicación paralela en un cluster Beowulf, muchas veces se utiliza PVM complementado con MPI. MPI provee funcionalidad adicional para el paso de mensajes remotos, que en PVM es bastante básica.

El modelo de paso de mensajes

La intención de MPI es ser un estándar para la implementación del Modelo de Paso de Mensajes de la computación paralela. Este modelo se basa en lo siguiente:

- Una aplicación paralela consiste en un numero de procesos cada uno trabajando sobre datos locales. Cada proceso tiene solamente variables locales y no hay mecanismo para que un proceso acceda a la memoria de otro.
- El intercambio de datos entre procesos tiene lugar mediante el paso de mensajes, es decir, explícitamente enviando y recibiendo datos entre procesos.

Estándares MPI

MPI es entonces una librería de funciones o subrutinas (en C ó Fortran) que se compila con las aplicaciones para proveer a estas la funcionalidad de paso de mensajes. Se han definido 2 estándares:

- MPI-1: contiene la definición de funciones que debe proveer cualquier implementación de MPI.
- MPI-2: incluye también I/O paralelo, manejo de procesos dinámicos y especificación de una librería para C++

Las implementaciones de los distintos proveedores pueden conformar al nivel MPI-1 o MPI-2, o parcialmente al MPI-2. Una de las cosas que no esta definido en MPI-1 es la activación de procesos remotos, que se considera dependiente de la plataforma, por eso es que algunas implementaciones de Beowulf que utilizan MPI para el paso de datos, también utilizan PVM o algún otro mecanismo para el manejo de procesos.

Objetivos de MPI

Los objetivos primarios de MPI, en su estándar 1, abarca los siguientes objetivos:

- Proveer portabilidad de código. Los programas MPI deberían compilar y ejecutar en cualquier plataforma.
- Permitir implementaciones eficientes a lo largo de diferentes arquitecturas homogéneas y heterogéneas
- Proveer funcionalidad avanzada de comunicación para el paso de mensajes entre procesos, desde el envió y recepción simple hasta rutinas especiales para broadcast y multicast.
- Manejo de tipos de datos definidos por el usuario.

Algunas cosas están explícitamente fuera del alcance de MPI-1. Estas cosas son:

- No se especifica el mecanismo preciso para poner en ejecución un programa MPI. Esto queda como característica dependiente de la plataforma.
- No se especifica manejo dinámico de procesos, esto es creación y destrucción de procesos mientras se ejecuta el código.

Estas últimas funcionalidades están mejor definidas en MPI-2.

MPI API

MPI establece que los elementos que participan en la comunicación son los procesadores (o nodos del cluster), cada uno de éstos corriendo instancias particulares de un programa.

Los programas orientados al paso de mensajes requieren distintos tipos de funcionalidad que podemos agrupar de la siguiente forma:

- **Llamadas para inicializar, administrar y finalmente terminar la comunicación entre procesadores**: estas llamadas permiten iniciar la comunicación entre procesadores, identificar los procesadores a ser usados, crear grupos de procesadores e identificar qué procesador está ejecutando una instancia particular del programa.
- **Llamadas para comunicar datos (mensajes) entre pares de procesadores**: permiten operaciones punto-a-punto, con bloqueo o asincrónicas para la transmisión de datos entre procesos ejecutandose en 2 procesadores distintos.
- **Llamadas para comunicar datos entre grupos de procesadores**: permiten operaciones de broadcast o multicast y sincronización para ese tipo de comunicaciones.
- **Llamadas para crear tipos de datos definidos por el usuario**: provee simplicidad para manejar tipos de datos complejos.

Las funciones y subrutinas de MPI comienzan con el prefijo **MPI_** y se pueden utilizar desde C/C++ o Fortran incluyendo el archivo de encabezado (mpi.h ó mpif.h)

La idea de estructura de un programa MPI es un solo programa que contiene toda la operatoria a realizar Luego este programa debe ser ejecutado en cada nodo o procesador donde va a correr y el mismo programa puede discriminar realizar una u otra tarea dependiendo del nodo o tipo de nodo en que se esta ejecutando. Un ejemplo de estructura de programa MPI es:

```
Si (ProcesadorActual = Maestro)
     ....
     Enviar Mensaje a Nodo Esclavo 1
```

```
      Enviar Mensaje a Nodo Esclavo 2
      ....
      Recibir Respuesta Nodo Esclavo 1
      Recibir Respuesta Nodo Esclavo 2
      ...
      Realizar Operación Final
      ...
Si (ProcesadorActual = Esclavo)
      Recibir Mensaje de Nodo Maestro
      Procesar
      Enviar Respuesta a Nodo Maestro
```

Este sería un ejemplo en pseudocódigo de un programa que aprovecha la descomposición de datos para realizar en paralelo una operación en distintos nodos esclavo.

Para la transmisión de datos en MPI, la unidad utilizada es el mensaje. Un mensaje consiste en un "sobre" (envelope) indicando el origen y destino y un cuerpo (body) conteniendo los datos a ser transmitidos. A su vez, el cuerpo del mensaje contiene un puntero al buffer en memoria donde esta almacenada la información a ser transmitida y especifica tanto el tipo de datos a ser enviado como la cantidad de items del tipo de datos que se transmitirán.

Describiremos dentro de estos grupos de funciones MPI para implementar un programa simple:

- MPI_Init: sirve para inicializar MPI. Se debe colocar una llamad a MPI_Init al principio de cada programa MPI. Se especifica el numero de procesador para luego identificarlo en el cluster.
- MPI_Finalize: libera los recursos utilizados por MPI.
- MPI_Send: permite enviar un mensaje a otro procesador. El procesador remoto debe estar esperando la recepción del mensaje. El procesador local bloquea hasta que el remoto acuse recibo.
- MPI_Recv: espera y recibe un mensaje de un procesador remoto.
- Modo de transmisión asincrónica: si bien la transmisión con bloqueo es ampliamente usada, MPI posee la opción de utilizar transmisión asincrónica. Para eso almacena los mensajes en un buffer MPI y la ejecución del programa MPI que realizó un Send continúa.

Uso de MPI para un cluster Beowulf

Como hicimos en PVM, se puede implementar una aplicación MPI que aproveche la arquitectura de nodos maestro/esclavos de un cluster Beowulf. La diferencia sería que los procesos deberán ser iniciados en cada procesador desde un inicio (no es el mismo programa MPI el que ejecuta procesos remotos) y la aplicación podría concentrarse en un solo programa MPI tal como el pseudocódigo del punto anterior.

PVM vs. MPI. Como aprovechar las ventajas de cada uno

PVM nació en 1.989 como un proyecto universitario y ha evolucionado desde entonces beneficiándose de las experiencias de sus usuarios. El concepto clave en su diseño es la máquina virtual: un conjunto de procesadores conectados por una red es percibido por el usuario como un único ordenador paralelo. El objetivo de este diseño es la portabilidad y flexibilidad, pero con un posible impacto en el rendimiento. Por el contrario, MPI es un estándar formal desarrollado en 1.993 por un comité de 60 expertos. El objetivo de MPI era definir un estándar que cada fabricante pudiera implantar para lograr el máximo rendimiento posible en su hardware. Los distintos orígenes de PVM y MPI así como sus distintas filosofías de diseño se traducen en un conjunto de diferencias:

- **Estándar**: PVM es el estándar de hecho para el procesamiento en paralelo, MPI es el estándar formal. Trabajar con un estándar formal puede ser una ventaja en algunos contextos.
- **Tolerancia a fallos**: PVM es tolerante a fallos en el sentido de que si un nodo falla los demás pueden seguir adelante con la ejecución del programa. Este mecanismo de recuperación no existe en MPI-1
- **Heterogeneidad**: la máquina virtual hace que PVM funcione muy bien en clusters heterogéneos (aquel construido con ordenadores distintos entre sí). MPI espera que el hardware sea homogéneo, una máquina MPP o, al menos, un Beowulf de ordenadores idénticos.
- **Rendimiento**: cabe esperar que MPI sea de mejor rendimiento que PVM, especialmente en máquinas MPP.
- **Base instalada**: al ser bastante posterior, el número de programas que hacen uso de MPI es significativamente inferior a los que emplean PVM.
- **Control de ejecución y gestión de recursos**: PVM especifica cómo ejecutar los programas independientemente del hardware y del sistema operativo. Asimismo, incluye un mecanismo para la gestión de recursos. MPI ni lo uno ni lo otro. Se espera que las implementaciones de MPI-2 incorporen esta funcionalidad.
- **Topología**: con MPI es posible especificar la topología de conexión entre elementos de proceso, lo que permite optimizar las comunicaciones. PVM no dispone de esta capacidad.

- **API**: al ser posterior a PVM, MPI aprendió de sus errores y ofrece un API más sencilla y eficiente.
- **Interoperabilidad**: distintas aplicaciones de MPI pueden no cooperar correctamente entre sí. Más aún, MPI no fuerza la interoperabilidad entre lenguajes, de tal forma que un programa en C puede no poder comunicarse con otro en Fortran. PVM carece de estos problemas.
- **Complejidad**: PVM muestra un propósito definido en todo su diseño. En la especificación de MPI 2 se han incluido toda una serie de capacidades al margen del paso de mensajes, como acceso remoto a memoria o un sistema paralelo de entrada/salida. Estas cosas son útiles, pero hacen de MPI una librería compleja.
- **Flexibilidad en las comunicaciones**: el conjunto de funciones de comunicación de MPI es más rico que el de PVM.

En conclusión, teniendo en cuenta la lista anterior, se ve que PVM se encuentra en mayor cantidad de implementaciones y es muy común encontrarlo en la mayoría de los casos de implementaciones de clusters Beowulf. En particular, su mejor comportamiento en entornos heterogéneos y su tolerancia a fallos puede que hagan de PVM la mejor alternativa para la programación de clusters. Por otro lado, MPI aparece como un estándar –lo cual garantiza cierta continuidad en el tiempo y que todas las implementaciones sean intercambiables—. Además, MPI-2 estandariza la generación de procesos dinámicamente lo cual agregaría a MPI la funcionalidad básica de una implementación de PVM. En la práctica, muchas implementaciones utilizan PVM y MPI combinadas, PVM para la administración de procesos y nodos del cluster y MPI para la comunicación entre procesos o paso de mensajes.

Anexo: Implementación del Cluster Gerónimo en el INTEC

En el INTEC Santa Fe se ha implementado con propósitos investigativos un cluster Beowulf consistente en 1 nodo maestro y 11 nodos esclavos.

El hardware utilizado consisten en:

- 7 nodos Pentium III de 833Mhz con 128Mb de memoria
- 4 nodos Pentium III de 500Mhz y 128Mb de memoria
- 1 nodo maestro Pentium III de 500Mhz y 256Mb de RAM

El cluster en si corre sobre Linux Red Hat recientemente actualizado a la versión 7 y el software adicional instalado es PVM y una implementación de MPI: MPICH 1.2.2. Esta implementación soporta la creación dinámica de procesos.

Las comunicaciones consisten en una Red Fast Ethernet con un switch dedicado al cluster. Una característica particular de este cluster y muy común hoy en día es el uso de nodos esclavos sin disco. El booteo se realiza vía RARP desde el NFS del nodo maestro. Para esto se han adaptado una serie de scripts de booteo de Red Hat Linux.

El cluster se utiliza para resolver problemas de cálculo numérico de elementos finitos. Estos problema consisten en sistemas de ecuaciones representados por matrices. Las dimensiones de esas matrices llegan al orden de las 300.000 incógnitas. Básicamente la paralelización se logra subdividiendo el dominio del problema en partes mas pequeñas representados por matrices simples. Cada una de estas matrices simples posee una serie de incógnitas que se pueden resolver independientemente y a la vez partes compartidas con otras subdivisiones, por lo cual la tarea de sincronizar los procesos paralelos es bastante compleja.

Con respecto a la resolución de problema de este tipo con manejo de gran volumen de acceso a memoria en las operaciones, se prevé la actualización a Pentium IV con Bus de acceso a RAM de 600Mhz. Según los benchmarks realizados esto aumentaría la performance en 4.5 veces.

Para la resolución de estos problema se utiliza la librería PETS que se encarga de la paralelización de la resolución de este tipo de problemas de calculo numérico. Esta librería permite definir objetos de tipo matriz y establecer bloques correspondientes a cada procesador del cluster. La sincronización de procesos la realiza PETS internamente utilizando MPI como estándar para el paso de mensajes.

Se investiga también el uso de compiladores de alta performance para C y para Fortran que pueden ser aplicados para la resolución de problemas que soporten paralelización en forma trasparente como pueden ser HPF (High Performance Fortran), OpenMP (Compilador de C paralelo) y el software PGI de Portland Group.

Mas información, software y scripts utilizados se pueden encontrar en http://minerva.ceride.gov.ar/geronimo

Referencias:

- Juan Carlos Ruiz; Proceso paralelo sobre clusters de máquinas Linux; Revista Linux Actual No. 4; 2000
- David Santo Orcero; Supercomputadores clase Beowulf; Revista Linux Actual No. 9; 2000
- Mark Baker, Rajkumar Buyya; Cluster Computer at a Glance; Division of Computer Science, University of Portsmouth; Hants (UK); School of Computer Science and Software Engineering, Monash University; Melbourne; 1999
- Donald J. Becker, Thomas Sterling: BEOWULF: A PARALLEL WORKSTATION FOR SCIENTIFIC COMPUTATION; Center of Excellence in Space Data and Information Sciences; http://cesdis.gsfc.nasa.gov/linux/beowulf/icpp95.html; 1995
- Beowulf Underground; Current Software; http://www.beowulf-underground.org/software.html; Noviembre 2001
- Jacek Radajewski and Douglas Eadline; Beowulf HOW-TO ; Version 1.1.1 22 November 1998
- Richard A. Sevenich; Parallel Processing using PVM; Linux Journal Magazine; Enero 1998
- Wayne J. Salamon and Alan Mink; Linux Clusters at NIST; Linux Journal Magazine; Junio 1999
- PACS Training Group; Introduction to MPI; Board of Trustees of the University of Illinois.; 2001
- Prabhaker Mateti; Network of Workstations; Wright State University; 1999
- Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam: PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing; MIT Press; 1994
- Al Geist; Advanced Tutorial on PVM 3.4; EuroPVM-MPI '97; 3 de noviembre de 1997; Polonia.
- Mario Storti y otros: Cluster Gerónimo en INTEC; http://minerva.ceride.gov.ar/geronimo; 2001

Entrevistas:

- Mario Storti; Implementación del Cluster Gerónimo en INTEC; 18 de diciembre de 2001; INTEC; Santa Fe; Argentina