UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IMPLEMENTACIÓN DE UN CLUSTER BEOWULF EN LINUX

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

JAIME GÓMEZ MARÍN

PROMOCIÓN
1996 - I

LIMA – PERÚ
2006

IMPLEMENTACIÓN DE UN CLUSTER BEOWULF

EN LINUX

Dedico este trabajo a:

Mis padres, inspiración plena de lucha y sacrifico,

Mis Hermanos, por el apoyo incondicional en mi carrera,

Y mis sobrinos esperanza de superación.

SUMARIO

El presente trabajo pretende describir la implementación de un Cluster Beowulf en Linux. Inicialmente en el capitulo I se da una introducción de los conceptos básicos de la computación distribuida y la terminología relacionada, en el capitulo II se indican los pasos para construir un Cluster Beowulf en una plataforma de Linux. Se trabaja básicamente con 2 distribuciones de Linux: SUSE 9.0 y RedHat 9.0. El Clúster está compuesto por 2 computadoras. El capitulo III presenta los resultados obtenidos de pruebas de benchmarking del clúster usando un algoritmo para medir el rendimiento de todo un sistema, el software usado es el HPL (High Performance Linpack), los resultados obtenidos nos muestran que nuestro clúster de 2 computadoras puede ser optimizados hasta 2,665. Gflops.

ÍNDICE

PRÓLOGO

En las diferentes áreas de investigación, el empleo de la información y su procesamiento siempre ha sido un limitante de la capacidad de procesamiento disponible en las computadoras usadas. Con los nuevos descubrimiento científicos, la necesidad de una mayor capacidad de procesamiento a conducido a soluciones relacionados con los cluster de computadoras.

Un cluster es un conjunto de computadoras, trabajando juntas para una finalidad común; el cluster puede estar compuesta de 2 o mas computadoras. El término cluster es aplicable a diferentes niveles: Sistema Operativo, Servicios ó Aplicaciones. Dependiendo de la arquitectura utilizada existen diferentes denominaciones para los cluster.

Específicamente en el año 1994, la NASA comenzó un proyecto denominado Beowulf en el Centro para la Excelencia en Datos Espaciales y Ciencias de la Información (CESDIS).El cual consistía en un cluster de 16 computadoras usando el sistema operativo Linux, el cual era destinado al proyecto de ciencias espaciales y terrestres (EES) ubicado en el centro de vuelo espacial de Goddard (GSFC) . La finalidad era construir un sistema computacional especifico usando elementos de bajos recursos, posteriormente la idea tuvo una fuerte aceptación en las comunidades académicas y científicas de todo el mundo. Un Cluster del tipo Beowulf es una arquitectura escalable de múltiple computadoras, que puede ser usada para cómputo paralelo y distribuido. La característica principal es la construcción de un sistema con componentes de hardware y software de bajo precio y de uso general, es decir, no contiene ningún tipo de hardware especializado.

CAPITULO I

MARCO TEÓRICO CONCEPTUAL

1.1. Cluster de computadoras

Los sistemas de cluster ofrecen una alternativa al mercado comercial computacional para sistemas computacionales escalables, con capacidades de computación mediana y alta. Varias aplicaciones han reemplazado a las supercomputadoras y los MPPs (procesadores paralelos masivos) han incorporando solo componentes desarrollados para un amplio mercado, explotando la economía escalar no disponible en el mercado de computación de alto nivel y evitando un significativo costo de desarrollo, resultando una ventaja entre el precio y la eficiencia que puede exceder en orden de magnitud para los trabajos de los usuarios. Adicionalmente, los usuarios tienen una gran flexibilidad de configuración, actualización y suministros, asegurando longevidad de esta clase de sistemas distribuidos y la confianza del usuario en su inversión de software

La clase de sistemas Beowulf explota el extenso mercado de componentes, tal como las PCs para entregar una excepcional ventaja de costo, con varias posibilidades para construir diferentes sistemas. Beowulf integra un software de sistema ampliamente disponible y fácilmente accesible a un bajo costo o a un costo cero suministrando varias de las capacidades requeridas por un entorno de sistema. Como resultado de todas estas características los Cluster Beowulf ha penetrado ampliamente en cada uno de los aspectos de la computación y rápidamente ha llegado a dominar los sistemas altos y medianos del procesamiento computacional.

La computación con el cluster Beowulf abarca 4 distintas, pero relacionadas áreas que son[1]

- Estructura del hardware del sistema
- El entorno de administración y la administración de recursos
- Librerías y herramientas para programación distribuida.
- Algoritmos paralelos

La estructura de hardware abarca todos los aspectos de los componentes del nodo de hardware y sus capacidades, los controladores de red dedicados y los switches, y la topología de interconexión que determina el sistema global de organización. El entorno de la administración de recursos es el organizador del software del sistema y las herramientas que gobiernan todas las fases de la operación del sistema desde la instalación, configuración e inicialización, a través de la administración y manejo de tareas, para el monitoreo del estado del sistema, diagnostico de fallas y mantenimiento.

Las herramientas y librerías de programación distribuida determinan el paradigma por el cual el usuario final coordina la distribución de recursos computacionales para ejecutarlo simultáneamente y cooperativamente la concurrencia que constituye los programas de aplicaciones paralelas. Finalmente, el dominio de los algoritmos paralelos suministra los modelos y acercamiento para organizar las aplicaciones de los usuarios para explotar el problema intrínsico del paralelismo mientras opera dentro de una eficiente organización.

1.2. Clasificación de la computación paralela

Una primera clasificación de la arquitectura de la computación paralela puede ser considerada en términos de acoplamientos: La típica latencia generada al ejecutar operaciones paralelas. Podrían variar desde los sistemas altamente acoplados de clase systolic con retardos de nanosegundos hasta los sistemas distribuidos que involucran recursos de computación que cruzan el mundo, cuyos retardos pueden llegar hasta los cientos de milisegundos. A continuación detallamos los sistemas existentes desde mayores acoplamientos hasta menor acoplamiento:

Systolic : Las computadoras son usualmente implementaciones de hardwired de propósitos específicos de una gran integración de algoritmos paralelos explotando 1, 2 ó 3 dimensiones de pipeling. Frecuentemente usado para procesadores de tiempo real de postsensor, DSP (procesadores digitales de señales), procesadores de imágenes, y generación de gráficos. La computación Systolic experimenta un renacimiento a través de la computación adaptiva, explotando la versatilidad de la tecnología FPGA (matriz de puertas programables) que permite diferente algoritmos systolic para ser programados dentro del mismo medio de FPGA en diferentes momentos.

Vector: Las computadoras explotan su operación de vectores altamente acoplados a través de un fuerte pipeling de accesos de bancos de memoria y estructuras de ALU, El hardware soporta operaciones gather-scatter, y acomoda las instrucciones de ciclo fetch/execute sobrecargadas en varias operaciones básicas con las operaciones de vector.

SIMD: (Single instruction, múltiple data) Esta arquitectura explota el paralelismo de los datos altamente acoplados por tener varios (en el orden de los miles) o simples procesadores funcionando la misma operación en pasos de bloqueo pero en diferentes datos. Un simple control procesa los comandos globales a todas los procesadores esclavos simultáneamente a través de un mecanismo de broadcasting. Tales sistemas incorporan redes de comunicación extensas para facilitar los movimientos de datos masivos que cruzan el sistema en poco ciclos. No usado en el área comercial, las estructuras SIMD continúan usándose en aplicaciones de propósitos específicos para procesamientos postsensor.

Dataflow: El modelo emplea un alto acoplamiento asíncrono al flujo de control que depende exclusivamente de la data precedente, esto genera una gran degradación del paralelismo y suministra un mecanismo de adaptación dinámica de secuencia en respuesta a los recursos cargados. Debido a esto sufre una terrible degradación, a pesar de no encontrar una entrada en el mercado, muchos de los conceptos relacionados con el paradigma dataflow ha tenido una fuerte influencia

en los análisis y optimización de los compiladores modernos, diseño de ALU y arquitectura multithreaded

PIM : (processor in memory) Esta arquitectura surge tan solo como una posible ayuda a la estructura de los sistemas de alto nivel, combinando memoria (DRAM ó SRAM) con procesamiento lógico en el mismo circuito integrado exponiendo una memoria de alto ancho de banda y a baja carencia para muchas operaciones orientado a los datos. Diversas estructuras fueron propuestas, incluyendo System on Chip (colocando bancos DRAM y la base de un procesador convencional en el mismo chip); SMP en un chip (colocaba la base de múltiples procesadores convencionales y 3 niveles de cache coherente con una arquitectura jerárquica en un chip) y la Smart Memory (colocaba lógica en la memoria DRAM para manipulación de datos) PIMs pueden ser usados en sistemas standalone, en arreglos de dispositivos, o como una pequeña capa de un multiprocesador convencional.

MPPs: (massively parallel processors) Constituye una amplia clase de arquitectura de multiprocesadores que explota los procesadores off-the-shell y chip de memorias en diseños personales de nodos de tarjetas, jerarquía de memoria, y sistemas globales de área de redes. MPPs abarca maquinas de memoria distribuida como el Intel Paragon, a través de memoria compartida sin caches coherentes tales como la BBN Butterfly

Clusters: Es un conjunto de computadoras off-the-shell integradas bajo una conexión de red dentro de un simple dominio de administración y usualmente en un simple cuarto de maquinas. Los cluster emplean redes comercialmente disponibles poniéndose a redes personalizadas, Cluster Beowulf incorpora la tecnología de PC masiva del mercado para sus nodos para lograr un desempeño en funcionamiento y bajo precio

Distribuided: La computación distribuida ó "**metacomputing**", combina la capacidad de procesamiento de números, las computadoras están distribuidas en el Internet, donde las computadoras interactúan sobre ciertas reglas especiales. Esta

clase de computación paralela explota los ciclos disponibles en las computadoras, obteniendo algo por casi nada.

Worktation cluster: Son estaciones de trabajo integradas por un sistema de redes de área. El hardware y el software están amarrados a un vendedor específico. Mientras superan en precio y rendimiento a las MPPs, pueden ser 2.5 a 4 veces más costosos que los clusters basados en PC.

Beowulf-class systems : Comprende a las PCs (por ejemplo Intel Pentium 4) integradas con redes de área local comerciales (Fast Ethernet, etc) o redes de área de sistema y tienen un software de bajo costo o costo cero para manejar los recursos de los sistemas y coordinar la ejecución paralela. Tales sistemas destacan por su excepcional precio/ejecución para varias aplicaciones

Cluster farm: Aprovecha la existencia de redes de PCs o estaciones de trabajo que sirven como estaciones de usuario dedicados o servidores, que cuando tiene tiempo muertos (idle) pueden ser empleados para ejecutar trabajo de terceros usuarios. Explotando el flujo de trabajo en paralelo, el sistema de software ha sido diseñado para distribuir las colas de trabajo mientras impide la intromisión en un recurso de usuario cuando es requerido. Estos sistemas tienen una baja eficiencia y efectividad debido a que comparte los recursos integrados de la red, oponiéndose a las redes dedicadas incorporados por los clusters de estaciones de trabajo y los Beowulfs

Superclusters: Son clusters de clusters, aun en una área local, tal como compartir un salón de maquinas ó en edificios separados en una empresa o campus universitario, usualmente integrado por el backbone WAN de la infraestructura de la institución. Aunque usualmente en el mismo dominio de Internet, los clusters podrían estar separados por responsabilidades administrativas o personales.

1.3. Estructura del hardware del sistema

El más notorio y discutible aspecto de los sistemas de cluster de computadoras son sus componentes físicos y su organización. Esto reparte las capacidades en

bruto del sistema, estableciendo considerables salones en los pisos de las maquinas y produciendo sus excelentes capacidades de precio / eficiencia. Los 2 principales subsistemas de un cluster Beowulf son sus nodos de computadoras y su red de interconexión que integra los nodos en un solo sistema.

1.3.1. Nodo de Computo Beowulf

El cómputo o nodo de procesamiento incorpora todo los dispositivos de hardware y mecanismos responsables para la ejecución de un programa, incluyendo la eficiencia de operaciones básicas, almacenamiento de la data procesada, suministro de almacenaje persistente y habilitando comunicaciones externas de resultados intermedios e interfase de comando de usuarios. 5 componentes claves constituyen el nodo de computo de un cluster Beowulf :

- El microprocesador
- La memoria principal
- La tarjeta madre
- Almacenamiento secundario
- Empaquetamiento

El microprocesador suministra la potencia de computo del nodo con sus picos de eficiencia medidos en Mips (millones de instrucciones por segundo) y Mflops (millones de operaciones de punto flotantes por segundo). Aunque el Beowulf puede ser implementado con cualquier tipo de familia de microprocesadores, las 2 familias de microprocesadores que mas resaltan son los Pentium III y IV de 32 bits de Intel y los Compaq Alpha 21264 de 64 bits. Los modernos microprocesadores comprenden de 20 a 50 millones de transistores, incluyendo una cantidad considerable de memorias on-chip de alta velocidad llamadas cache para el rápido acceso a los datos. El Cache es organizado en jerarquía usualmente en 2 ó 3 capas, la mas cercana al procesador es la mas rápida, pero la menor y la mas distante es relativamente la mas lenta pero de mayor capacidad. Los caches almacenan datos e instrucciones de la memoria principal, y donde la data es rehusada o localizada con

un acceso rápido, como resultado se mejora el rendimiento del sistema. El microprocesador se comunica externamente por 2 buses: Una interfase optimizada de un gran ancho de banda hacia la memoria principal y otra en soporte a los datos I/O

La memoria principal almacena el conjunto de datos y programas de trabajo, usado por el microprocesador durante la ejecución de su trabajo. Basado en la tecnología DRAM en el cual un simple bit es almacenado como una carga en un pequeño capacitor accesado a través de transistores de conmutación, la lectura y escritura de datos es mucho mas lento para la memoria principal que la memoria cache. Sin embargo la velocidad de acceso a la memoria se ha mejorado con el diseño del bus de la memoria, esta tecnología es conocida como RAMbus.

La tarjeta madre es el medio de integración que combina todos los componentes de un nodo en un solo sistema operacional. La tarjeta madre incorpora sofisticado chips que son tan complejos como el mismo microprocesador. Estos chips manejan todas la interfaces entre los componentes y el control a los protocolos de bus. Un importante BUS es PCI, la primera interfase entre el microprocesador y los dispositivos externos de mayor velocidad. La primera versión era de un bus de 32 bit y operaba a 33Mhz, las mas actual es de 64 bits y opera a 66Mhz, cuadriplicando su rendimiento. En la mayoría de los sistemas de red, los controladores de interfases son conectados a los nodos mediante un bus PCI. La tarjeta madre también contiene una memoria de solo lectura (el cual puede ser actualizada) , el cual almacena la BIOS del sistema (Basic Input/Output System), un juego de servicios de bajo nivel, primeramente relacionados con las funciones de I/O y las tareas básicas de bootstrap, que definen la interfase lógica entre el Sistema Operativo de alto nivel y el hardware. La tarjeta madre también soporta diferentes puertos de entrada y salida (I/O) para los periféricos de entrada y salida (teclado, ratón, monitor, USB, impresora).

El almacenamiento secundario suministra alta capacidad de dispositivos de almacenamiento persistentes. Mientras la memoria principal pierde todo su contenido cuando el sistemas es desconectado de una fuente de alimentación, el

almacenamiento secundario retiene su data en el estado de apagado, mientras varias PC standalone incluyen diversas clases de almacenamientos secundarios, algunos sistemas Beowulf pueden tener muchos nodos que mantienen solo algunas cosas necesarias para almacenar la imagen de boot para inicializar el arranque del sistema, los demás datos pueden ser bajados desde nodos externos o desde el nodo master. El principal sistema de almacenamiento secundario existente es el disco duro, el cual es basado en un medio magnético de pocas dimensiones. Esta tecnología es tan antigua como la misma computación digital, y continúa expandiendo su capacidad de almacenamiento en orden exponencial, aunque su velocidad de acceso y ancho de banda solo han sido mejorados en forma gradual. 2 principales contenedores: SCSI (small computer system interface) y EIDE (enhanced integrated dual electronics) son diferenciados por su velocidad y su capacidad en el primer caso, y luego por el costo en el segundo caso. La mayoría de las workstations usan SCSI, y la mayoría de PC usan EIDE. Otras 2 formas de almacenamiento secundario son las disqueteras y los discos ópticos. Los disquetes de 3.5 que pueden almacenar hasta 1.4 Mbytes son un ideal medio de boot para los nodos de un cluster Beowulf. Los discos ópticos: CD (compact disk), CD-ROW (compact disk read/write) y DVD (digital versátil disk), de los cuales los 2 primeros pueden almacenar hasta 600 Mbytes, con tiempos de acceso del orden de los pocos milisegundos.

Los empaquetamientos para PC fueron originalmente en la forma de "paquetes de envolturas de pizza". Los primeros cluster Beowulf fueron configurados con tales empaquetamientos, usualmente hasta 8 de ellos encima uno del otro, pero para el tiempo que fueron implementados los primeros Cluster en 1994, los tower case era lo que predominaban y se usaban pilas de computadoras con tower case. Pero la integración ha avanzado hacia una integración y el mercado potencial de cluster Beowulf ha hecho posible la generación de un nuevo conjunto de rack especializados para cluster Beowulf.

1.3.2. Redes de Interconexión

Sin la reducción del costo de la tecnología de redes, el cluster Beowulf nunca hubiera existido. Las 2 ramas de origen del clusters Beowulf tiene sus origenes en diferentes tecnologías. Ethernet fue desarrollada para redes de área local para interconectar sistemas aislados de usuarios y compartir recursos de comunidades de cómputo. En cambio Myrinet fue desarrollada desde la base de la experiencia con acoplamiento de procesadores en MPPs, tales como el Intel Paragon. Juntos Fast y Gigabite Ethernet, y Myrinet suministra la mayoría de clases de cluster Beowoulf.

Una red es una combinación del transporte físico y mecanismo de control asociado con una jerarquía de capas de mensajes encapsulados. El núcleo es el mensaje. Un mensaje es una colección de información organizada en un formato (orden y tipo) que tanto el proceso del enviador y del receptor entienden y pueden ser interpretado correctamente. El mensaje puede ser de poco bytes hasta miles de bytes. Realmente lo que sucede es que el proceso de envió llama a rutinas que manejan las interfases entre la aplicación y la red. La ejecución de una operación de alto nivel de envió causa que el mensaje del usuario sea empaquetado con cabeceras de informaciones adicionales. Las funcionalidades adicionales de ruteos se agregan al cluster. Luego el hardware de bajo nivel dirige las líneas de los canales de comunicación con la señal y el conmutador de la red rutea el mensaje de acuerdo a la información de ruteo, hasta que el mensaje es recibido por el proceso de recepción, el cual hace el proceso inverso al enviador.

La red se caracteriza principalmente en términos de ancho de banda y latencia. El ancho de banda es la tasa al cual un mensaje de bits son transferidos, usualmente citados en términos de picos de rendimiento (throughtput) como bit por segundos. La latencia es el tiempo que se requiere para enviar el mensaje. Tanto el ancho de banda y la latencia son sensibles al tamaño del mensaje y al trafico de la red. Mensajes largos pueden mejorar el uso de los recursos de la red y el throughput. Mensajes cortos reducen el tiempo de transmisión y recepción, pero suministra una sobrecarga baja de latencia de transferencia, pero causa un uso ineficiente del ancho de banda. Un alto trafico en la red (número de mensajes por una unidad de tiempo)

incrementa la sobrecarga del throughput de la red, pero produce una efectiva tasa de latencia en la transferencia de mensajes

1.4. Software del Nodo del Cluster

Un nodo del cluster es frecuentemente una entidad de cómputo autónoma, con su propio sistema operativo. Los nodos del cluster Beowulf explotan los modernos sistemas operativos para manejar efectivamente el recurso del nodo y la comunicación con otros nodos por medio de la interconexión de las redes.

Linux ha emergido como el sistema operativo dominante, parecido al UNIX. Recientemente Linux ha recibido un alto apoyo de los mayores vendedores de computadoras del mundo incluidos IBM, Compaq, HP, etc. Linux tiene características completas de un sistema operativo multiusuario, multitareas , demanda de paginación de memoria virtual con avanzadas características del software del kernel soportado para un alto rendimiento en operación de red.

1.4.1. Linux

a. Introducción

Linux es un Sistema Operativo, que se ha ido posicionando en diferentes campos como: Servidores de Aplicaciones, Base de Datos, Estaciones de Trabajo, Terminales X, Desarrolladores de UNIX, Servidores de Internet, Clusteres, Sistemas Embedded, Sistemas de Universidades. Soluciones en áreas de: Hotelería, Medica, Sistemas de Reservaciones, Oficinas Legales, Compañías Petroleras, Gobiernos, Telecomunicaciones, ISP, Manufacturas y otros.

b. Características

- Multiplataforma: Las plataformas en las que en un principio se puede utilizar Linux son 386-, 486-. Pentium, Pentium Pro, Pentium II/III,

Amiga y Atari, también existen versiones para su utilización en otras plataformas, como Alpha, ARM,MIPS, PowerPC y SPARC.

- Multiprocesador: Soporte para sistemas con mas de un procesador está disponible para Intel y SPARC.
- Funciona en modo protegido 386.
- Protección de la memoria entre procesos, de manera que uno de ellos no pueda colgar el sistema.
- Carga de ejecutables por demanda: Linux sólo lee del disco aquellas partes de un programa que están siendo usadas actualmente.
- Política de copia en escritura para la comparación de páginas entre ejecutables: esto significa que varios procesos pueden usar la misma zona de memoria para ejecutarse. Cuando alguno intenta escribir en esa memoria, la página (4Kb de memoria) se copia a otro lugar. Esta política de copia en escritura tiene dos beneficios: aumenta la velocidad y reduce el uso de memoria.
- Memoria virtual usando paginación (sin intercambio de procesos completos) a disco: A una partición o un archivo en el sistema de archivos, o ambos, con la posibilidad de añadir más áreas de intercambio sobre la marcha Un total de 16 zonas de intercambio de 128Mb de tamaño máximo pueden ser usadas en un momento dado con un límite teórico de 2Gb para intercambio. Este limite se puede aumentar fácilmente con el cambio de unas cuantas líneas en el código fuente.
- La memoria se gestiona como un recurso unificado para los programas de usuario y para él caché de disco, de tal forma que toda la memoria libre puede ser usada para caché y ésta puede a su vez ser reducida cuando se ejecuten grandes programas.

  Librerías compartidas de carga dinámica (DLL's) y librerías estáticas.

  Se realizan volcados de estado (core dumps) para posibilitar los análisis postmortem, permitiendo el uso de depuradores sobre los programas no sólo en ejecución sino también tras abortar éstos por cualquier motivo.
- Compatible con POSIX, System V y BSD a nivel fuente.

- Emulación de iBCS2, casi completamente compatible con SCO, SVR3 y SVR4 a nivel binario.
- Todo el código fuente está disponible, incluyendo el núcleo completo y todos los drivers, las herramientas de desarrollo y todos los programas de usuario; además todo ello se puede distribuir libremente. Hay algunos programas comerciales que están siendo ofrecidos para Linux actualmente sin código fuente, pero todo lo que ha sido gratuito sigue siendo gratuito.
- Control de tareas POSIX.
- Pseudo-terminales (pty's).
- Emulación de 387 en el núcleo, de tal forma que los programas no tengan que hacer su propia emulación matemática. Cualquier máquina que ejecute Linux parecerá dotada de coprocesador matemático. Por supuesto, si el ordenador ya tiene una FPU (unidad de coma flotante), esta será usada en lugar de la emulación, pudiendo incluso compilar tu propio kernel sin la emulación matemática y conseguir un pequeño ahorro de memoria.
- Soporte para muchos teclados nacionales o adaptados y es bastante fácil añadir nuevos dinámicamente.
- Consolas virtuales múltiples: varias sesiones de login a través de la consola entre las que se puede cambiar con las combinaciones adecuadas de teclas (totalmente independiente del hardware de vídeo). Se crean dinámicamente y puedes tener hasta 64.
- Soporte para varios sistemas de archivo comunes, incluyendo minix-1, Xenix y todos los sistemas de archivo típicos de System V, y tiene un avanzado sistema de archivos propio con una capacidad de hasta 4 Tb y nombres de archivos de hasta 255 caracteres de longitud.
- Acceso transparente a particiones MS-DOS (o a particiones OS/2 FAT) mediante un sistema de archivos especial: no es necesario ningún comando especial para usar la partición MS-DOS, esta parece un sistema de archivos normal de UNIX (excepto por algunas restricciones en los nombres de archivo, permisos, etc). Las particiones comprimidas de MS-DOS 6 no son accesibles en este momento, y no se espera que lo

sean en el futuro. El soporte para VFAT, FAT32 (WNT, Windows 95/98) se encuentra soportado desde la versión 2.0 del núcleo y el NTFS de WNT desde la versión 2.2 (Este ultimo solo en modo lectura).

- TCP/IP, incluyendo FTP, telnet, NFS y otros.
- Appletalk.
- Software cliente y servidor Netware.
- Lan Manager / Windows Native (SMB), software cliente y servidor.
- Diversos protocolos de red incluidos en el Kernel: TCP, IPv4, IPv6, AX.25, X.25,IPX, DDP, Netrom.
- Diversos Protocolos de Wan: Frame-Relay, X25, ISDN, ATM.

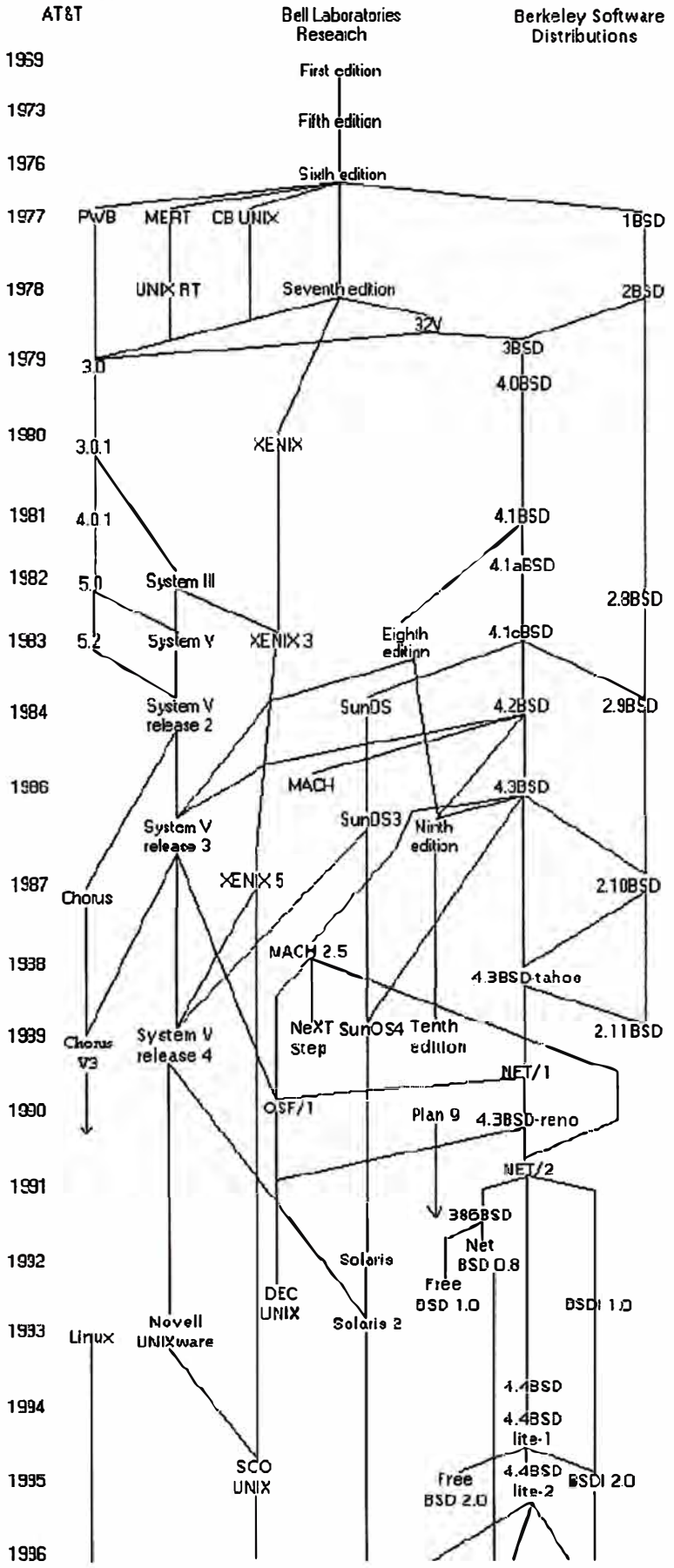
c. Historia de UNIX

El UNIX sistema operativo inventando en 1969 en los Laboratorios Bell de AT&T por Dennis M.Ritchie y Kenneth Thompson. Inicialmente desarrollado en lenguaje Asembler luego en 1973 portado al lenguaje C. Luego apareció distribuciones UNIX de otras empresas propietarios y del tipo libre, a continuación algunas distribuciones: AIX / AIXL, BSD, DUNIX, DYNIX/ptx, HP-UX, IRIX, MINIX, OSF/1, QNX, SCO, Sun Solaris, System V, Tru64 UNIX, Ultrix, Unisys, UnixWare, Xenix. En la Figura 1 se muestra la cronología de UNIX.

d. Distribuciones de Linux

Una distribución es la unión del Kernel de Linus Torvalds (www.kernel.org) y los programas con licencia GNU como son los editores, compiladores, software de administración, software de servicios y otros. Una herramientas GNU puede copiarse, modificar e instalarse libremente desapareciendo el concepto de licencia. El costo significativo seria la media. (el Cdrom). Existen direfentes tipos de distribuciones, pero todas las distribuciones poseen igual kernel (www.kernel.org) y herramientas GNU. Las distribuciones son personalizadas para determinados sectores o determinadas regiones. Ejemplo:

Figura 1.1 Cronología de UNIX



- Sector Hogar: Ubuntu : Distribuciones que lidera la demanda de Linux por su facilidad en la instalación y por una gran cantidad de utilitarios, teniendo una gran aceptación en el mercado del Hogar.
- Sector Servidores: Distribuciones RedHat, Suse, Conectiva, TurboLinux,
- Sector Regionales: RedHat (EEUU), Conectiva (Brasil), SUSE (Europa) , Asianux (China).

Las agrupaciones de las distribuciones realizadas no descarta que una distribución del Sector Hogar pueda instalarse en el Sector de Servidores y viceversa. Será la decisión del usuario la elección.

La administración de las distribuciones no es diferente, porque al ser Sistemas Operativos Linux poseen igual estructura de administración. (usuarios, passwords, networking, procesos y otros). Esta característica permite que conociendo la administración de una distribución pueda manejar otra distribución. En el caso de los servicios y sus respectivas configuraciones las distribuciones son muy similares porque empaquetan el mismo software GNU disponible en Internet, que en su mayoría son iguales. A continuación se muestra algunos servicios basados en software GNU

Tabla 1.1 Servicios de Internet

Servicio	Software	Pagina Web
Web	Apache	www.apache.org
Correo	Sendmail	www.sendmail.org
Proxy	Squid	www.squid.org

Esta característica permite que el software de una Distribución pueda instalarse en otra distribución. Para obtener una distribución debes acceder al pagina web de la distribución y entrar a la sección de descargas del contenido del CD's.

e. Principales Distribuciones

A continuación se describe las distribuciones mas importantes

- **RedHat**: Distribución que tiene muy buena calidad, contenidos y soporte a los usuarios por parte de la empresa que la distribuye. Fácil de instalar.

  FTP : ftp.redhat.com/pub/

  WEB : www.redhat.com

- **DEBIAN**: Distribución con muy buena calidad. El proceso de instalación es un poco mas complicado que el anterior, pero sin mayores problemas.

  FTP : ftp.debian.org/debian/

  WEB : www.debian.org

- **SUSE**: Distribución de calidad germana. Con una buena aceptación en el mercado de servidores y con soporte para Oracle.

  FTP : ftp.suse.com

  WEB : www.suse.de/es/index.html

- **Caldera OPENLINUX**: Caldera ofrece una versión libre y reducida de esta distribución para su respectiva evaluación. Para obtener el software completo deberá de comprar la distribución. En el 2000 Caldera adquirió la empresa SCO UNIX, distribución de prestigio.

  FTP : ftp.calderasystems.com/pub/OpenLinux

  WEB : www.calderasystem.com

- **SLACKWARE**: Esta distribución fue una de las primeras que circulo. Últimamente esta incorporando actualizaciones.

  WEB : www.slackware.com

- **COREL LINUX**: Nueva distribución de empresa Corel, basada en Debian. Programa de instalación gráfico y simplificado.

  FTP : ftp.corel.com/pub/corel

  WEB : www.corel.com

- **Mandrake**: Distribución basada en RedHat. Esta distribución trae el KDE totalmente integrado en el sistema. Fácil de instalar y configurar.
  WEB : www.linux-mandrake.com/es/

- **MKLINUX**: Distribución de Linux para los Power Macintosh y PowerMac.
  FTP : ftp.mklinux.apple.com/pub
  WEB : www.mklinux.apple.com
- **CONECTIVA**: Distribución en español y portugués desarrollado en Brasil.
  WEB : www.conectiva.com/es/
  PERU : www.conectiva.com.pe

1.5. Software de Red

Existen software de red disponibles para la programación[2], administración y uso de sistemas Beowulf . Los software de red son normalmente descriptos como una pila, hecho de diferentes capas de protocolos que ínter operan unos con otros.

1.5.1. TCP/IP

Las computadoras paralelas tienen la tradición especial de usar una red de comunicación de alto rendimiento que usan protocolos personalizados para trasmitir datos entre los elementos de procesamiento. Debido al gran crecimiento de Internet, el TCP/IP se volvió en el estándar de facto para los protocolos de comunicación de red.

El protocolo IP esta conceptualmente dividido en capas lógicas diferentes que se combinan para formar un pila de protocolos. La capa IP es un capa de datagramas ruteables. Los datos hacer transmitidos son fragmentados en datagramas (paquetes de datos). La longitud del paquete es limitada por la capa física de transporte y la capa IP contiene la lógica para fragmentar los requerimientos que son muy

extensos y fragmentarlo en múltiples paquetes IP que son reensamblados en el destinatario. Cada paquete es routeado en forma individual y contiene 4 byte para la dirección IP , el cual especifica el host destino. Esta versión de IP es conocida como IPv4. Una nueva versión, denominada IPv6, incrementaría el espacio de direccionamiento disponible para las aplicaciones IP. Los 4 bytes para la dirección IPv4 no dan abasto para todas las computadoras conectadas en todo el mundo. La IPv6 contiene 16 byte para representar la dirección de un host.

La pila IP comúnmente soporta 2 servicios : TCP (Protocolo de control de comunicación) y UDP (Protocolo de data grama de usuario). TCP, el mas común servicio IP, suministra un confiable servicio de flujo secuenciado por un secuencia de bytes. Mientras la capa física usualmente esta llena de errores, el TCP suministra la integridad de la data. TCP incorpora un mecanismo de retransmisión que recupera paquetes perdidos. También tolera la latencia mientras mantiene un buen rendimiento en el caso normal de no perdida de paquetes. TCP suministra su propia flujo de datos de empaquetado, evitando la fragmentación en la capa IP. La desventaja es que el TCP no tiene un buen manejo en redes WAN. El otro servicio de IP, UDP, suministra un protocolo de datagramas no confiables. La ventaja de UDP es que tiene una baja latencia debido a que no ocurre los problemas de retardo al inicio de una transmisión.

La dirección IP es definida por un numero de 32 bits que únicamente identifica un host destino. La dirección IP son usualmente escrito en notación decimal, con los bytes de la dirección escritos como un numero decimal separado por puntos decimales. El rango de direcciones IP es dividido en redes a lo largo de su bit de limites. La porción de la red que permanece fijo es conocido como dirección de red y el restante es conocido como dirección de host. La división entre estos dos es especificado por la mascara. Una red típica es 255.255.255.0, el cual especifica los 24 bits de la dirección de red y 8 bits de direccionamiento de host.

3 direcciones de red han sido reservados para redes privadas:

- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 -172.31.255.255
- 192.168.0.0 - 192.168.255.255

Estas rango de direcciones son permanentemente no asignados y no son redireccionados en los router de las redes de internet. En el pasado pocas mascaras eran permitidos. Las redes se agrupaban por grupos denominados: clase A (255.0.0.0 con 16 millones de direcciones de host), clase B (255.255.0.0 con 64000 direcciones de host) y la clase C (255.255.255.0 con 254 direcciones de host)

1.5.2. Socket

Los sockets son las interfaces de bajo nivel entre los programas a nivel de usuario y el sistema operativo y la capa de hardware de una red de comunicaciones. Suministra un mecanismo portable para que 2 o mas aplicaciones se comuniquen, y soporten una variedad de formatos de direccionamiento, semánticas y protocolos. Los sockets fueron introducidos en el BSD 4.2 release de UNIX. Desde su introducción, los sockets API han sido ampliamente adoptados y estan disponibles en todos los variantes de UNIX de la actualidad, incluidos el Linux. En linux el sockets API es soportado directamente por el sistema operativo. Los sockets API son una potente pero no necesariamente elegante. Muchos programadores evitan trabajar directamente con los sockets, ocultándolo con capas de abstracción (por ejemplo la llamada a procedimiento remotos ó las librerias MPI).

La idea básica bajo la abstracción de los sockets es que la comunicación de red es tratada de la misma forma como se maneja los archivos de entrada y salida Una vez que la conexión de red es establecida entre dos procesos, la transmisión y recepción de datos son ejecutados por los funciones read() y write(), de la misma forma como se envía datos a un archivo, a una cinta, o algún otro dispositivo. El socket API esta principalmente relacionado con el nombramiento y ubicación de

los puntos de comunicación y la asignación de descriptores de archivo adecuados par el uso de read() y write(). Un socket es creado haciendo una llamada al sistema a socket() con sus argumentos definiendo un dirección de familia, un tipo de socket, y un protocolo. Teóricamente un numero enorme de posibilidades es posible, pero en el practica solo dos maneras son usadas.

1.5.3. Sistema de archivos distribuidos

Cada nodo de un cluster Beowulf esta equipado con un disco duro que tiene un sistema de archivos local que ejecuta procesos en otros nodos que podrían ser accesados. La mayoría de sistemas de archivos distribuidos tienen las siguientes características :

- Red transparente: Los archivos remotos deben ser accedidos usando las mismas operaciones o llamadas al sistemas que se usarían si fuera archivos locales.
- Localización transparente: El nombre de un archivo no esta amarrado a su localización en la red. La localización del servidor de archivos no es una parte intrínseca de la ruta del archivo.
- Localización independiente: Cuando una localización física de un archivo cambia, su nombre no debe ser forzado a cambiar. El nombre de un servidor de archivos no es una parte intrínseca de la ruta del archivo

El protocolo de sistema de archivos de red (NFS) es el protocolo que suministra un servicio de sistema de archivos distribuidos para los clusters Beowulf. NFS es estructurada como una arquitectura cliente servidor, usando mecanismos de llamada a procedimientos remotos para la comunicación entre el cliente y el servidor. El servidor exporta archivos hacia los clientes, los cuales acceden a los archivos como si fueran archivos locales. A diferencia de otros protocolos NFS es sin estado, y no graba ninguna información acerca de un cliente entre sus requerimientos. En otras palabras todas las llamadas del cliente son consideradas independientes y deben contener toda la información para su ejecución . Todos los requerimientos de lectura y escritura de NFS

deben de incluir un archivo de desplazamiento, diferente a como la lectura y escritura de archivos que proceden desde donde el ultimo fue dejado. La naturaleza sin estado del servidor NFS produce que los mensajes sean muy largos, consumiendo gran parte del ancho de banda. La ventaja de no tener estado, es que el servidor no se ve afectado cuando un cliente colapsa.

1.5.4. Ejecución de Comandos Remotos

Para la administración de un cluster Beowulf es necesario la ejecución de comando en nodos sin previamente haberse logeando en el. A continuación se describira los programas disponibles.

Los comandos R es un set de programas que permiten ejecutar comandos y copiar archivos en forma remota. Los comandos mas principales son :

- rsh: Es un shell remoto que te permite ejecutar comandos del shell en nodos remotos y también iniciar sesiones interactivas de logeos. Las sesiones interactivas son ejecutadas por rlogin.
- rlogin: El comando rlogin remoto te permite iniciar un sesión de un terminal por medio de un logeo a un nodo remoto.
- rcp: El comando remoto copy te permite copiar archivos de un nodo a otro.

Los comandos rsh es la forma estándar de cómo ejecutar comandos y empezar aplicaciones paralelas en otros nodos. Una considerable cantidad de software de sistemas , incluidos alguna implementaciones de MPI, confían en los comandos remotos para su ejecución. Los comandos rsh requieren que un servidor rsh se ejecute en el nodo donde ser ejecutara los comandos. Los programas rsh se conectan al servidor, el cual verifica que el puerto original del cliente es un puerto privilegiado antes de tomar una acción. Los comandos rsh son útiles para ejecutar tareas de administración de sistemas y lanzar aplicaciones paralelas. Sin embargo solo permite la ejecución en un solo nodo,

y muchas veces se requiere ejecutar comando simultáneamente en varios nodos, esta carencia se puede superar escribiendo script que multipliquen la acción de los comandos rsh.

El shell seguro SSH es un shell seguro para el reemplazo de los comando rsh, rlogin y rcp. El SSH tiene su contrapartida de comandos que son ssh, slogin, scp. El mayor problema con los comando R es que ellos transmiten por la red los clave en formato plano. SSH es un producto comercial desarrollado por SSH Communications Security, Ltd., el cual es ofrecido tanto para las versiones de Win32 y UNIX. La versión de UNIX esta disponible como código fuente y pueden ser bajada desde www.shh.fi. SSH encripta toda la comunicación de la red, usando para tal fin sistemas de autenticación basados en llaves públicas para verificar la autenticidad de un host y de un usuario.

1.6. Programación Distribuida

Para aprovechar al máximo un cluster Beowulf, ha sido necesario el desarrollo de un amplio rango de aplicaciones paralelas que aprovechen los recursos de un sistema paralelo para permitir resolver complejos problemas en cortos tiempos. La programación difiere de la programación de un único procesador, la diferencia parte debido al hecho que el compartir información entre nodos del cluster Beowulf puede ser mayor que entre los nodos de un sistema altamente acoplado debido a la fragmentación del espacio de memoria entre los nodos, originando una sobrecarga adicional en comparación a un sistema de compartición de memoria. Como consecuencia un programador del código de una aplicación paralela debe tener en consideración esto y otras fuentes de degradación del rendimiento del cluster.

Un numero diferentes de modelos han sido empleados para la programación y ejecución de aplicaciones paralelas, pero el que ha predominado es el proceso de comunicación secuencial, mejor conocido como el modelo de paso de mensajes. A través de esta metodología el programador particiona el problema global de datos entre un grupo de nodos y específica el proceso que debe ser ejecutado en cada uno, cada nodo trabaja en su respectivo espacio de trabajo con los datos asignados.

Cuando se necesita información de otros nodos, el usuario establece rutas lógicas de comunicación entre los procesos de los nodos separados. La aplicación del programa por cada proceso explícitamente envía y recibe mensajes pasados entre el mismo y entre unos o mas nodos remotos. Un mensaje es un paquete de información que contiene uno o mas valores en un orden y formato que todos los procesos involucrados en el intercambio entienden, Los mensajes son también usados para sincronizar procesos concurrentes en orden para coordinar la ejecución de tareas paralelas en diferentes nodos.

Dos sistemas de programación de paso de mensaje han sido desarrollados para facilitar la programación paralela. Son librerías que pueden ser enlazadas en conjunto con lenguajes de programación convencionales como C y Fortran. PVM fue el primer sistema de programación empleado en los primeros cluster Beowulf, MPI fue el secundo y mas reciente sistema de programación de sistemas distribuidos, fue desarrollado como un producto por una comunidad. MPI es el modelo escogido en la mayoría de las comunidades de desarrollo de aplicaciones de cluster Beowulf . Existe un número de software de código abierto y comercial que implementan MPI, se usara uno de código abierto para el presente informe. Las nuevas características en la programación distribuida orientada hacia I/O a generado el origen del MPI-2.

1.6.1. Programación Paralela con MPI

El computo paralelo en un cluster Beowulf es logrado dividiendo un computo en partes y haciendo uso de múltiples procesadores, cada ejecución de estas partes es efectuado por cada procesador. Algunas veces un programa ordinario puede ser usado por todos los procesadores, pero los parámetros o archivo de entradas son totalmente diferentes para cada uno, en tales circunstancias no existe una comunicación entre las tareas separadas. Cuando el poder de un computador paralelo es necesario para atacar un gran problema con una estructura mas complejas, tal comunicación es necesaria. Uno de los mas sencillos logros para la comunicación es tener un procesador coordinando las actividades por el envió y recepción de mensajes, muy similar a como un grupo de personas podría coordinar

le ejecución de una tarea compleja. Este logro para lograr el paralelismo es llamado paso de mensajes[3].

MPI es una especificación de la librería de paso de mensajes, que nos permitirá escribir programas paralelos, entre las características más importantes de MPI podemos citar:

- MPI direcciona el modelo de paso de mensajes de un computador paralelo, en el cual los procesos con espacio de direcciones separadas se sincronizan con otros y mueven datos desde las direcciones de espacio de un proceso a otro por el envió y recepción de mensajes.

- MPI especifica una interfase de librería, el cual es una colección de subrutinas y sus argumentos. No es un lenguaje, en vez de eso, las rutinas MPI son llamadas por programas escritos en lenguajes convencionales tales como Fortran, C y C++.

- MPI es una especificación, no una implementación particular. La especificación fue creada por el Forum MPI, un grupo de vendedores de computadoras paralelas, computadora de ciencias y usuarios quienes vienen trabajando en cooperación con los estándares de la comunidad. La primera fase de esta reunión resulto en el release del estándar de 1994, que es referido como el MPI-1.Una vez que el estándar fue implementado y ampliamente usado, una segunda series de reuniones resulto en un set de extensiones, referidos como el MPI-2. MPI se refiere tanto a MPI-1 y MPI-2.

Como una especificación MPI es definido por un documento estándar, la forma como C, Fortran ó POSIX son implementados. Los documentos del estándar de MPI esta disponibles en www.mpi-forum.org y pueden ser libremente bajados. Las implementaciones de MPI están disponibles para la mayoría de computadoras paralelas.

El logro del Forum MPI fue crear una potente y flexible librería que podría ser implementado eficientemente en las más grandes computadoras y suministrar una herramienta para atacar el más difícil problema en el cómputo paralelo.

a. Programa MPI: impresión de un mensaje

Se mostrara un programa básico de MPI, el cual mostrara el típico programa de hola mundo impreso en la consola del proceso

```
#include <stdio.h>
#include <mpi.h>

/**
 *   Programa principal : imprime hola mundo
 */
int main(int argc, char **argv){

    // inicializa valores del cluster
    MPI_Init(&argc,&argv);
    // inicializa valores del cluster
    printf("hola mundo \n");
    // Finaliza
    MPI_Finalize();
    //
    return 0;
}
```

Todos los programas MPI contienen una llamada a `MPI_Init` y uno a `MPI_Finalize`. Las otras rutinas MPI deben ser llamadas después de `MPI_Init` y antes de `MPI_Finalize`. Todos los programas en C y C++ deben de incluir el archivo `mpi.h`.

Todos los procesos imprimen los mismos textos. A continuación se muestra la segunda versión del programa anterior:

```
#include <stdio.h>
#include <mpi.h>

/**
 *   Programa principal : imprime hola mundo identificando
 *                        el proceso
 */
int main(int argc, char **argv){
```

```
    //
    int rank, size;
    // inicializa valores del cluster
    MPI_Init(&argc,&argv);
    //
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    //
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    //
    printf("hola mundo | de proceso %d de %d\n", rank,
size);
    // Finaliza
    MPI_Finalize();
    //
    return 0;
}
```

En esta versión se usa dos variables: `rank` y `size`. Debido a que el programa MPI es una comunicación de procesos, cada proceso tiene su propio set de variables. En este caso cada proceso tiene su propio espacio de direccion conteniendo sus propias variables `rank` y `size` (y `argc`, `argv`, etc). La rutina `MPI_Comm_size` retorna el número de procesos de trabajo del MPI en el segundo argumento. Cada uno de los procesos MPI es identificado por un número, denominado el `rank`, variando desde 0 hasta `size-1`. La rutina `MPI_Comm_rank` retorna en el segundo argumento el rango del proceso. La salida de este programa podría mostrar los siguientes mensajes:

hola mundo | de proceso 0 de 4
hola mundo | de proceso 2 de 4
hola mundo | de proceso 3 de 4
hola mundo | de proceso 1 de 4

Se debe considerar que la salida no esta ordenado desde el proceso 0 a 3. MPI no especifica el comportamiento de las rutinas o sentencias del lenguaje, como la rutina `printf`, en particular no especifica el orden de la salida de las sentencias `printf`.

b. Programa MPI: comunicación entre procesos

El siguiente programa nos permitirá controlar la comunicación entre procesos, el control de comunicación entre procesos se vera reflejado en el orden del envió de mensajes de consola del programa. A continuación se muestra el programa:

```
#include <stdio.h>
#include <mpi.h>

/**
 *  Programa principal : imprime hola mundo
 */
int main(int argc, char **argv){

    //
    int numprocs, myrank, namelenm, i;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    char greeting[MPI_MAX_PROCESSOR_NAME+80];
    MPI_Status status;


    // inicializa valores del cluster
    MPI_Init(&argc,&argv);
    //
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    //
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    //
    MPI_Get_processor_name(processor_name, &namelen);
    //
    sprintf(greeting,"hola mundo, desde proceso %d de
    %d on %s\n", myrank, numprocs, processor_name);

    if(myrank == 0 ) {

        printf("%s\n",greeting);

        for(i=1 ; i < numprocs ; i++){

            MPI_Recv(greeting, sizeof(greeting), MPI_CHAR,
                     i, 1, MPI_COMM_WORLD, &status);


            printf("%s\n",greeting);
        }

    } else {

        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR,
                 0, 1, MPI_COMM_WORLD);

    }
```

```
    // Finaliza
    MPI_Finalize();
    //
    return 0;
}
```

El programa obtendrá el nombre del procesador que esta ejecutado el proceso, luego si el proceso tiene un rango mayor a 0 entonces enviara el nombre del procesador al proceso con rango 0, en caso contrario imprimirá el nombre de este procesador y por cada rango recibirá el nombre del procesador y lo imprimirá.

La rutina `MPI_Get_processor_name` obtiene el nombre del procesador en el cual el proceso esta ejecutándose. Las rutinas `MPI_Send` y `MPI_Recv` son los 2 requerimientos que deben satisfacer la comunicación de datos entre 2 procesos:

- Describe el dato ha ser enviado o la localización en el cual se recibirá los datos.

- Describe el destino (para ser enviado) o el origen (para ser recibido) de los datos

Adicionalmente, MPI provee una forma para etiquetar los mensajes y para encontrar información acerca del tamaño y la fuente del mensaje. Un buffer de datos es descrito por su tamaño y su longitud , por ejemplo "a, 100", donde "a" esta apuntando a un buffer de 100 bytes de datos. En Unix la llamada a "write" describe los datos hacer escritos con un dirección y longitud. MPI generaliza esto suministrando 2 capacidades adicionales: describiendo no continuas regiones de datos y describiendo datos de modo tal que puedan ser comunicadas entre los procesadores con diferentes representaciones de datos. Para hacer esto MPI usa 3 valores para describir un buffer de datos: la dirección, el tipo de dato(MPI), y el numero o cantidad de items de el tipo de dato. Por ejemplo un buffer que contiene 4 ints C es descritos por el par "a, 4, MPI_INT". Hay un predeterminado tipo de datos de MPI para todos lo tipos de datos básicos

definidos en C, C++ y Fortran. Los tipos de datos mas comunes son presentados a continuación:

Tabla 1.2 Tipos de datos MPI para diferentes lenguajes de programación

C	
int	MPI_INT
double	MPI_DOUBLE
float	MPI_FLOAT
long	MPI_LONG
char	MPI_CHAR

C++	
int	MPI::INT
double	MPI::DOUBLE
float	MPI::FLOAT
long	MPI::LONG
char	MPI::CHAR

FORTRAN	
INTEGER	MPI_INTEGER
DOUBLE PRECISION	MPI_DOUBLE_PRECISION
REAL	MPI_REAL
CHARACTER	MPI_CHARACTER
LOGICAL	MPI_LOGICAL

c. Descripción del origen o destino

El origen o destino de un mensaje es especificado pasando el rango de el proceso. MPI generaliza la noción de rango de origen y destino por rango haciendo el rango relativo a un grupo de procesos. Este grupo podría ser un subset del original grupo de procesos. Permitiendo el uso de subset de procesos y usando rangos relativos, permite usar de manera sencilla el uso de MPI para escribir software orientado a componentes. El objeto MPI que describe un grupo de procesos es llamado comunicador. Tanto origen como destino están dados por 2 parámetros: un rango y un comunicador. EL comunicador `MPI_COMM_WORLD` esta predefinido y contiene todos los procesos iniciados por los comandos `mpirun` ó `mpiexec`. Como un origen, el valor especial `MPI_ANY_SOURCE` puede ser usado para indicar que el mensaje podría ser recibido desde cualquier rango de los procesos MPI en este programa MPI.

d. Seleccionando la cantidad de mensajes

El "extra" argumento para `MPI_Send` es un entero no negativo etiquetando un valor. Esta etiqueta permite a un programa enviar un número extra con la data. `MPI_Recv` puede usar este valor tanto para seleccionar cual mensaje va ha recibir (especificando un valor en la etiqueta) ó usando la etiqueta para transportar una extra data (especificando con el valor `MPI_ANY_TAG`). En el ultimo caso, el valor de la etiqueta del mensaje recibido es almacenado en el argumento del estado (es el ultimo parámetro de la rutina `MPI Recv` en el código C mostrado anteriormente). Esta es un estructura en C, un arreglo de enteros en Fortran, y una clase en C++. La etiqueta y el rango del proceso que envía, puede ser accedido por una referencia hacia un elemento apropiado del estado, el cual es mostrado a continuación:

Tabla 1.3 Funciones para obtener la etiqueta y rango de procesos

C	C++	Fortran
status.MPI_SOURCE	status.Get_source()	status(MPI_SOURCE)
status.MPI_TAG	status.Get_tag()	status(MPI_TAG)

e. Determinando la cantidad de data recibida

La cantidad de data recibida puede ser encontrada usando la rutina `MPI_Get_count` . Por ejemplo:

```
MPI_Get_count(&status, MPI_CHAR, &num chars);.
```

Retorna en `num_chars` el número de caracteres enviados. El segundo argumento debería ser del mismo tipo de dato MPI que se uso para enviarlo (desde que varias aplicaciones no necesitan esta información , el uso de una rutina permite la implementación para evitar el computo de `num chars` a menos que el usuario necesite este valor).

A continuación se muestra un ejemplo que suministra el tamaño máximo del buffer de datos en el receptor. Es también posible encontrar la cantidad de memoria necesaria para recibir un mensaje usando `MPI Probe`.

```
#include <stdio.h>
#include <mpi.h>

/**
 *  Programa principal
 */
int main(int argc, char **argv){

    //
    int num_chars, src;
    MPI_Status status;

    .......
    .......

    //
    MPI_Probe(MPI_ANY_SOURCE, 1, MPI_COMM_WORLD,&status);
    //
    MPI_Get_count(&status, MPI_CHAR, &num_chars);
    //
    greeting = (char *) malloc (num_chars);
    //
    src = status.MPI_SOURCE;
    //
    MPI_Recv(greeting, num_chars, MPI_CHAR, src, 1,
             MPI_COMM_WORLD, &status);
    //
    MPI_Get_processor_name(processor_name, &namelen);
    //

    .......
    .......

    // Finaliza
    MPI_Finalize();
    //
    return 0;
}
```

MPI garantiza que los mensajes estén ordenados y que el `MPI_Recv` después de `MPI_Probe` podria recibir el mensaje que el provador retorna la información tanto largo como el mismo criterio de selección del mensaje (rango de origen, comunicador, etiqueta del mensaje) son usados . En este ejemplo el origen para la rutina `MPI Recv` es especificado como un `status.MPI_SOURCE` , no como

`MPI_ANY_SOURCE` para estar seguro que el mensaje recibido es el mismo como uno al cual `MPI Probe` retorna la información.

1.7. Resumen

En el presente capitulo se hace mención a los cluster de computadoras como una alternativa al mercado comercial computacional de sistemas escalables. Específicamente se describe 2 características importantes de los cluster de computadora del tipo Beowulf (ó simplemente cluster Beowulf), los cuales son la reducción de costo y el rendimiento de procesamiento en comparación con otros tipos de cluster existentes. Se muestra la clasificación de la computación paralela, mostrando la ubicación de los cluster Beowulf dentro de esta clasificación.

Finalmente se muestra 3 puntos a tener en cuenta en la implementación de un cluster Beowulf:

- El hardware: Las consideraciones sobre el microprocesador, memoria principal, tarjeta madre, almacenamientos secundarios, empaquetamiento del computador y la red de interconexión entre las computadoras.
- El software: Las consideraciones sobre el sistema operativos a usar, se menciona el UNIX y Linux como sistemas operativos (el sistema operativo Linux ha tenido gran aceptación por parte de las más importantes empresas tecnológicas: IBM, Compaq, HP, etc).
- Programación Distribuida: Un nuevo paradigma aparece en la ejecución de programas en los cluster de computadoras. La programación de paso de mensaje es la solución para la programación en los cluster (las 2 especificaciones existentes son PVM y MPI).

CAPITULO II

IMPLEMENTACIÓN DE UN CLUSTER BEOWULF

2.1. Introducción

En el presente capitulo se detallara el proceso para la construcción de un Cluster Beowulf con 2 computadoras, empleando como sistema operativo la distribuciones de Linux de Suse 9.0 y RedHat 9.0 , sobre una red Ethernet [4].

2.2. Requerimientos

Para el presente informe de la implementación de un cluster Beowulf se empleo:

- 2 computadoras con NIC (tarjetas de Red) de 100 Mbps
- Un switch o un hub conectado a las computadoras
- Como sistema operativo Linux (SUSE 9.0 - master y RedHat9.0 – slave)
- Como implementación de MPI (message passing interface), se uso MPICH

2.3. Arquitectura del Cluster Beowulf

En la implementación del Cluster Beowulf, las computadoras se distribuyen de la siguiente manera:

- Un nodo master:
    - Sistema Operativo RedHat 9.0,
    - CPU Pentium4-2.0GHz

- 2GB RAM

- Un nodo slave:
  - Sistema Operativo SUSE 9.0,
  - CPU Penitum4 - 3.0 GHz
  - 512 MB RAM

La topología empleada es en BUS

2.4. Configuración del Nodo Master

Se debe usar la computadora con mas recursos disponible e instalar el Sistema Operativo Linux (RedHat 9.0). Se recomienda instalar el software necesario y los servicios de red, especialmente NFS y ssh. Si se necesita algún tipo de desarrollo dentro del cluster Beowulf podría adicionarse herramientas de desarrollo de lenguaje C. El servicio de NFS no es indispensable, pero es un buen mecanismo para copiar archivos entre los nodos y para automatizar los procesos de instalación. En el nodo master se podría agregar una tarjeta de red adicional para poder acceder en forma remota desde otro punto fuera del Cluster Beowulf, aunque esto no es requerido para la operación del Cluster. Se recomienda no usar claves muy cortas, porque algunos algoritmos de encriptación de ssh no podrían funcionar con la creación de llaves u otra funcionalidad.

Como se va a usar MPICH (una implementación de MPI) es necesario bajar los servicios de Firewall porque MPICH usa en forma aleatoria puertos de comunicación entre los nodos. Se debe seguir con detenimiento el proceso de instalación de MPICH que viene en la documentación de los instaladores. Debes de ejecutar el MPICH sin necesidad de ser root, es decir otro usuario y repetir el mismo proceso en las otras computadoras. Se construirá cada nodo del Cluster con el mismo usuario y clave, y no deberán ser root.

2.4.1. Hosts

Lo primero que se debe hacer es la modificación del archivo /etc/hosts. Modificar directamente la línea que dice:

127.0.0.1 wolf00 localhost.localdomain localhost

Deberá decir:

127.0.0.1 localhost.localdomain localhost

Luego deberás adicionar los otros nodos que están considerados en tu Cluster. Aunque esto no es requerido para la operación del Cluster, es recomendable porque facilita la referencia a cualquier computadora del Cluster por medio de alias o nombres.

192.168.0.100 wolf00 # master 1
192.168.0.101 wolf01 # slave 1
192.168.0.102 wolf02 # slave 2
192.168.0.103 wolf03 # slave 3

En este caso se considera que el Cluster esta formado por 4 computadoras, una de las cuales es el Master y los demás son los Slaves

2.4.2. Grupos

Para poder configurar el Cluster se asignara la responsabilidad a un usuario, por lo tanto se recomienda configurar un nivel de seguridad aceptable a tal usuario. Después de crear el usuario "wolf", crea un grupo "beowulf" y adiciona el usuario al grupo, de modo que solo podrás modificar algún archivo o directorio que puedan ser accesibles solo por el grupo o por los usuarios que pertenezcan al grupo.

groupadd beowulf
usermod –g beowulf wolf

Y adicionar la siguiente línea en el archivo profile de tu usuario (/home/wolf/.bash_profile)
umask 007

Con esta ultima acción, cualquier archivo o directorio creado por el usuario "wolf" podrá ser modificado solamente por el grupo "beowulf"

2.4.3. NFS

Con el servicio NFS se desea mantener un repositorio compartido con todos los nodos para compartir algún tipo de información tanto de entrada como de salida. A continuación se detalla como llegar a tal fin:
Crear un directorio para que cada nodo pueda compartir algún tipo de información:

mkdir /mnt/wolf
chmod 770 /mnt/wolf
chown wolf:beowulf /mnt/wolf -R

En el directorio /etc, adicionamos nuestra carpeta compartida en el archivo de exportacion /etc/exports:

cd /etc
cat >> exports
/mnt/wolf 192.168.0.100/192.168.0.255 (rw)
<ctrl>+<d>

2.4.4. Direcciones IPs

Se usara una red con formato 192.168.0.nn debido a que es una red privada de propósitos generales. El nodo master, se llamara wolf00 y su IP será 192.168.0.100, y cada otro nodo se denominara wolfnn y tendra un IP que seguirá la siguiente secuencia: 192.168.0.100 + nn. La razón de seguir esta regla, es por la facilidad de seguir aumentando nuevos nodos al cluster, de modo que se vuelva escalable.

2.4.5. Servicios

Verificar que los siguientes servicios estén habilitados:

chkconfig -add sshd
chkconfig -add nfs
chkconfig -add rexec
chkconfig -add rlogin
chkconfig -level 3 rsh on
chkconfig -level 3 nfs on
chkconfig -level 3 rexec on
chkconfig -level 3 rlogin on

Se deberá remover algunos servicios que no son usados, por ejemplo:

chkconfig -del atd
chkconfig -del rsh
chkconfig -del sendmail

2.4.6. SSH

Como un nivel de seguridad se recomienda usar ssh. Como usuario root debes modificar el archivo /etc/ssh/sshd_config en la linea:

#RSAAuthentication yes

#AuthorizedKeysFile .ssh/authorized_keys

Se deberá descomentar (eliminar los #), rebootear la computador para que surtan efectos los cambios y logearse como el usuario "wolf". Se deberá generar las llaves públicas y privadas para usarlo con ssh, esto se realiza con el siguiente comando:

ssh-keygen -b 1024 -f .ssh/id_rsa -t rsa -N ""

A continuación se mostrara un mensaje donde te indicar la creación satisfactoria de una llave pública y privada, los cuales serán "*id_rsa*" e "*id_rsa.pub*" respectivamente, en la carpeta "*/home/wolf/.ssh*"

Copia el archivo "*id_rsa.pub*" hacia un archivo llamado "*authorized_keys*" en la carpeta .ssh (será usado posteriormente en los nodos slaves). Modificamos la seguridad de los archivos y la carpeta .ssh

*chmod 644 .ssh/auth**
chmod 755 .ssh

Finalmente agregar las siguientes líneas en tu archivo "*.bash_profile*":

export LAMRSH='ssh -x'
ssh-agent sh -c 'ssh-add && bash'

2.4.7. MPICH

Como implementación de MPI se usara el producto MPICH 1.2.7.p1 [5]. El proceso de instalación de describe en el Anexo A [6]. El MPICH se instalo en el directorio /usr/local/mpich-1.2.7.p1/ch_p4

2.5. Configuración del Nodo Slave

Se instalara el Sistema Operativo Linux (SUSE 9.0). Se recomienda instalar el software necesario y los servicios de red, especialmente NFS y ssh. Si se necesita algún tipo de desarrollo dentro del cluster Beowulf, se adicionara herramientas de desarrollo de lenguaje C y FORTRAN para la compilación de las herramientas de MPI, HPL y GOTO BLAS. El servicio de NFS no es indispensable, pero es un buen mecanismo para copiar archivos entre los nodos y para automatizar los procesos de instalación.

Como se va a usar MPICH (una implementación de MPI) es necesario bajar los servicios de Firewall en los slave porque MPICH usa en forma aleatoria puertos de comunicación entre los nodos. Se debe seguir con detenimiento el proceso de instalación de MPICH que viene en la documentación de los instaladores (Anexo 1). Debes de ejecutar el MPICH sin necesidad de ser root, es decir otro usuario y repetir el mismo proceso en todos los nodos slave. Se construirá cada nodo del Cluster con el mismo usuario y clave, y no deberán ser root.

2.5.1. Hosts

En forma similar al nodo master, la modificación del archivo ***"/etc/hosts"***, se realizara de la misma forma, modificar directamente la línea que dice:

127.0.0.1 wolf00 localhost.localdomain localhost

Deberá decir:

127.0.0.1 localhost.localdomain localhost

Luego deberás adicionar los otros nodos que están considerados en tu Cluster. Aunque esto no es requerido para la operación del Cluster, es recomendable porque

facilita la referencia a cualquier computadora del Cluster por medio de alias o nombres.

192.168.0.101 wolf00 # master 1
192.168.0.101 wolf01 # slave 1
192.168.0.102 wolf02 # slave 2
192.168.0.103 wolf03 # slave 3

En este caso se considera que el Cluster esta formado por 4 computadoras, una de las cuales es el Master y los demás son los Slaves

2.5.2. Grupos

Adicionar el grupo "***beowulf***" y el usuario "***wolf***" en cada nodo slave, de forma similar a como se realizo en el nodo master. El proceso se describe a continuación:

groupadd beowulf
usermod –g beowulf wolf

Y adicionar la siguiente línea en el archivo profile de tu usuario (/home/wolf/.bash_profile)

umask 007

Con esta ultima acción, cualquier archivo o directorio creado por el usuario "wolf" podrá ser modificado solamente por el grupo "beowulf"

2.5.3. NFS

Se desea montar la carpeta NFS que es compartida por el nodo master (wolf0), para tal fin como root modificamos el archivo /etc/fstab, agregando la línea:

wolf00:/mnt/wolf /mnt/wolf nfs rw, hard,intr 0 0

A continuación se debe crear el directorio donde se montara la carpeta del nodo master:

mkdir /mnt/wolf
chmod 770 /mnt/wolf

2.5.4. Direcciones IPs

Las IP de los nodos slave se asignaran siguiendo el criterio definido en el punto 3.3.4, por lo tanto se usara una red con formato 192.168.0.nn, debido a que es una red privada de propósitos generales. Los nodos slaves empiezan a partir de nn mayor o igual a 101, y los nombres de los nodos slaves se denominaran wolfnn, donde nn será mayor o igual a 01. La razón de seguir esta regla, es por la facilidad de seguir aumentando nuevos nodos al cluster, de modo que se vuelva escalable.

2.5.5. Servicios

Verificar que los siguientes servicios estén habilitados:

chkconfig -add sshd
chkconfig -add nfs
chkconfig -add rexec
chkconfig -add rlogin
chkconfig -level 3 rsh on
chkconfig -level 3 nfs on
chkconfig -level 3 rexec on
chkconfig -level 3 rlogin on

Se deberá remover algunos servicios que no son usados, por ejemplo:

chkconfig -del atd

chkconfig -del rsh
chkconfig -del sendmail

2.5.6. SSH

En el $HOME del usuario wolf crear la carpeta ".ssh" y copiar dentro del directorio el archivo "*authorized_keys*" del nodo master. Logearse via ssh desde el nodo master al nodo slave , la primera vez pedira la clave del usuario wolf, las siguientes veces la clave ya no será solicitada.

2.5.7. MPICH

Se instalara el MPICH como se realizo en el nodo master. Ver punto 3.3.7

2.6. Resumen

En el presente capitulo se basa en la implementación de un cluster Beowulf con 2 computadoras. El sistema operativo empleado es SUSE 9.0 (en el lado master) y RedHat 9.0 (en el lado slave). La asignación de IP en la red es dado bajo el criterio: 192.168.0.[100+nn], donde nn es el nodo del cluster Beowulf. El master tendrá la IP 192.168.0.100, el slave 192.168.0.101.

La configuración de software en el nodo master y slave es similar:

- Se crea el usuario wolf y el grupo beowulf.
- La ejecución remota desde master al slave se realiza a través del SSH (se hace uso de llaves de autenticación).
- Se instala una implementación de MPI, el producto MPICH 1.2.7.p1
- Se desinstala los servicios que no serán usados por el cluster Beowulf en cada nodo

CAPITULO III

ANÁLISIS Y PRESENTACIÓN DE LOS RESULTADOS

3.1. Introducción

En el presente capitulo se usara la especificación MPI para realizar aplicaciones distribuidas básicas en el cluster y finalmente se ejecutara un benchmarking para medir el rendimiento del cluster Beowulf implementando previamente.

3.2. Uso de MPI para calcular números primos.

Se implementara una aplicación para calcular los números primos, usando una sola computadora y luego se adaptara la aplicación para ejecutarlo en el Cluster. EL lenguaje de programación es el C estándar. Se calculara los números primos basándose en el siguiente algoritmo:

"Todo número N es primo si y solo si no es divisible entre los números correspondientes entre 2 y la raíz cuadrada de N"

Ejecución del algoritmo de calculo de número primos sin clúster:

Se muestra el código para calcular los números primos:

```
#include <stdio.h>
#include <mpi.h>

/**
 *  Verifica si un número es primo o no
```

```
 */
int is_prime(int n) {
      //
    int j,ret = 1;

    // algoritmo para saber si un número
    // es primo o no
    for(j=2; j<(int)sqrt(n); j ++ )
            if( n % j == 0 )  ret = 0 ;
      //
    return ret;
}

/**
 *  Programa principal
 */
int main(int argc, char **argv){

    int n=1;              // número a analizar
    int primes=0;         // contador de numeros primos
    int chunk=100000;     // cada que cierto número se
                          //mostrara informacion

    while(1){

      // si un numero es imprimo incrementa el contador
      if(is_prime(n++)) primes++;

      // imprime resultados temporales
      if(!(n%chunk))
            // imprime resultados parciales
            printf("%d primes < %d\n", primes, n);
    }

    //
    return 0;
}
```

Ejecución de algoritmo del calculo de números primos con clúster:

El código fuente anterior se adapta para poder usarlo en el Cluster, el código modificado se muestra a continuación:

```
#include <stdio.h>
#include <mpi.h>

/**
 *  Verifica si un número es primo o no
 */
int is_prime(int n) {
      //
    int j,ret = 1;

    // algoritmo para saber si un número
```

```
    // es primo o no
    for(j=2; j<(int)sqrt(n); j ++ )
            if( n % j == 0 )  ret = 0 ;
      //
    return ret;
}

/**
 *  Programa principal
 */
int main(int argc, char **argv){

    int rank;
    int size;

      // inicializa valores del cluster
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);


    int n2;
    int primesh=0;// almacena numero de primos parciales
    int primes;   // almacena numero de primos totales
    int chunk=100000;
    int n1 = rank*chunk; // asigna rango de iteracion


    while(1){

      n2 = n1 + chunk;
      if( n1==0 ) n1 = 1;

      // analiza rango en cada nodo del cluster
      int n;

      for( n=n1; n<n2 ; n++){
          if(is_prime(n++))
                  primesh++;
      }

      // envio resultado de cada nodo slave
      // al nodo master
      MPI_Reduce(&primesh, &primes, 1 , MPI_INT,
              MPI_SUM, 0 , MPI_COMM_WORLD);

      // prepara el siguiente rango a analizar
      n1 += size*chunk;

      // imprime resultados parciales
      if(!rank)
          printf("%d primes < %d\n", primes, n1);
    }

    MPI_Finalize();

    return 0;
}
```

3.3. Benchmarking

Para realizar el Benchmarking de nuestro cluster usaremos la herramienta ***HPL (High Performance Linpack)*** [7], el proceso de instalación del HPL [8] y sus requisitos [9] se encuentran en los Anexos B y C respectivamente .

La herramienta HPL tiene un archivo de configuración (HPL.dat) en el cual nos permite personalizar los parámetros de configuración de nuestro benchmarking [10], se plantea las siguientes arquitecturas:

- Clúster de 1 computadora
- Clúster de 2 computadoras

3.3.1. Benchmarking de un clúster de 1 computadora

Los parámetros de configuración se detallan a continuación:

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
8            device out (6=stdout,7=stderr,file)
2            # of problems sizes (N)
5000 10000 Ns
3            # of NBs
80 104 122   NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
1            Ps
1            Qs
16.0         threshold
3            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
```

```
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

Los resultados obtenidos se muestran a continuación:

Tabla 3.1 – Característica de un cluster con 1 nodo
100 Mbs con Switch - 1 NICs por nodo

CPU	Pentium 4 - 3GHz - 512Mb
OS	Linux 9.0 Suse (Kernel 2.4.21-99-smp46)
Compilador C	gcc (gcc-3.3.1-24)
MPI	MPICH 1.2.7.p1
BLAS	GotoBLAS (libgoto_nortwood32p-r1.00.so)
HPL	1.0a

Tabla 3.2 - Rendimiento (Gflops)
Tamaño del Problema en 1 nodo

Grid	NBs	5000	10000
1x1	80	1,241	---
1x1	104	1,236	---
1x1	122	1,279	---

Se observa que la máxima velocidad de procesamiento se obtuvo para un tamaño de problema de 5000 y un NB de 122 : ***1,276 Gflops*** . (Un Gigaflops es un millón de operación de puntos flotantes)

3.3.2. Benchmarking de un clúster de 2 computadoras

Los parámetros de configuración se detallan a continuación:

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
8            device out (6=stdout,7=stderr,file)
4            # of problems sizes (N)
5000 10000 25000 50000  Ns
3            # of NBs
80 104 122   NBs
0            PMAP process mapping (0=Row-,1=Column-major)
2            # of process grids (P x Q)
1 2          Ps
```

```
2 1             Qs
16.0            threshold
3               # of panel fact
0 1 2           PFACTs (0=left, 1=Crout, 2=Right)
2               # of recursive stopping criterium
2 4             NBMINs (>= 1)
1               # of panels in recursion
2               NDIVs
3               # of recursive panel fact.
0 1 2           RFACTs (0=left, 1=Crout, 2=Right)
1               # of broadcast
0               BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1               # of lookahead dept
0               DEPTHs (>=0)
2               SWAP (0=bin-exch,1=long,2=mix)
64              swapping threshold
0               L1 in (0=transposed,1=no-transposed) form
0               U  in (0=transposed,1=no-transposed) form
1               Equilibration (0=no,1=yes)
8               memory alignment in double (> 0)
```

Los resultados obtenidos se muestran a continuación:

Tabla 3.3 – Característica de un cluster con 2 nodos
100 Mbs con Switch - 1 NICs por nodo

CPU - master	Pentium 4 - 3GHz - 512Mb
CPU - slave	Pentium 4 - 2GHz - 2Gb
OS-master	Linux 9.0 Suse (Kernel 2.4.21-99-smp46)
OS-slave	Linux 9.0 Redhat (Kernel 2.4.20-8)
Compilador C - master	Gcc (gcc-3.3.1-24)
Compilador C - slave	gcc (gcc-3.2.2-5)
MPI	MPICH 1.2.7.p1
BLAS	GotoBLAS (libgoto_nortwood32p-r1.00.so)
HPL	1.0a

Tabla 3.4 - Rendimiento (Gflops)
Tamaño del Problema en 2 nodos

Grid	***NBs***	***5000***	***10000***
1x2	80	2,127	2,655
1x2	104	2,137	2,659
1x2	122	2,115	2,665
2x1	80	1,817	2,396
2x1	104	1,719	2,393
2x1	122	1,683	2,321

Se observa que la máxima velocidad de procesamiento se obtuvo para un gris de 1x2, un tamaño de problema de 10000 y un NB de 122: ***2,665 Gflops*** (un Gigaflops es mil millones de operación de puntos flotantes)

3.4. Resumen

El presente capitulo esta dedicado al empleo del lenguaje de programación MPI. Se divide en 2 partes:

La primera parte es la implementación de un algoritmo de cálculo de números primos usando el lenguaje de programación C, luego se implementa una solución empleado MPI, el cual usa el procesamiento paralelo de todo el cluster.

En la segunda parte se realiza un benchmarking del cluster usando una herramienta: HPL (High Performance Linpack), el cual nos permite configurar parámetros de afinamiento del cluster para obtener el máximo rendimiento del cluster para aplicaciones paralelas. El valor obtenido es de 2.665 Gflops (un Gigaflops es mil millones de operaciones de punto flotantes)

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. Para un rendimiento adecuado en un Clúster Beowulf sobre Linux, el servidor master debe ser el nodo con mayor capacidad de recursos (velocidad de CPU, memoria, etc). Los demás nodos slave no necesariamente tienen que tener la misma configuración de hardware y software, pero se recomienda que para uniformizar la carga de procesamiento en todo los slave, tengan similares configuraciones de hardware y software.
2. En todo Clúster Beowulf la red debe ser dedicada exclusivamente para el Clúster, si se usara una red compartida dentro de un dominio, el rendimiento del Clúster podría degradarse enormemente.
3. Para facilitar la configuración se debe crear el mismo usuario en todos los nodos y los directorios de instalación de los productos deben ser los mismos.
4. La cantidad de memoria es un elemento que mejorara el rendimiento del clúster, porque Linux es un sistema operativo que emplea la memoria eficientemente para la ejecución de procesos.
5. De las pruebas del Benchmarking se obtienen las valores óptimos para un procesamiento del Cluster Beowulf. El tamaño del problema deberá ser de 10000 y un grid de 1x2, para obtener un procesamiento de 2.665 Gflops. Para una mayor capacidad de procesamiento, se deberá aumentar un nodo adicional al cluster.

RECOMENDACIONES

1. El Sistema Operativo usado es Linux, pero existen diferentes distribuciones, por lo tanto se recomienda usar una sola distribución para todos los nodos de ese modo se facilitara el mantenimiento del Clúster.

2. Se recomienda una Red de 100 Mbps para la conexión de los nodos del Clúster.

3. No es necesario que todos los nodos slave tengan un monitor o teclados, solo es necesario en el master, porque desde ahí se puede configurar los demás nodos slave en forma remota (usando el servicio de telnet.).

4. No se debe instalar servicios innecesarios en los sistemas operativos porque degradaran el rendimiento, debido a que usaran recursos de memoria y procesador.

ANEXO A

INSTALACIÓN DE MPICH

La implementación de MPI usada en el presente informe es MPICH versión 1.2.7.p1, los pasos para la instalación y uso del MPICH sobre la distribución de SUSE 9.0 se describen a continuación:

1. Se considera que el usuario "**wolf**" existe.
2. Download MPICH 1.2.7 p1 (**mpich.tar.gz**) desde http://www.mpi.com y grabarlo en la carpeta **$HOME/install/**
3. Desempaquetar MPICH en la carpeta **$HOME/dev/**
   [wolf@master] cd $HOME/dev
   [wolf@master] md5sum $HOME/install/mpich.tar.gz
   [wolf@master] tar –xzvf $HOME/install/mpich.tar.gz.
4. Entrar en el directorio donde está desempaquetado MPICH
   [wolf@master] cd mpich
5. Configurar MPICH
   [wolf@master] ./configure --with-device=ch_p4
   --prefix=/usr/local/mpich-1.2.7.p1/ch_p4
6. Compilar MPICH
   [wolf@master] make
7. Instalar MPICH (debe ingresar al sistema previamente como root)
   [wolf@master] su
   [wolf@master] make install
8. Adicionar a la variable PATH de tu **.profile** /usr/local/mpich-1.2.7.p1/ch_p4/bin

ANEXO B

INSTALACIÓN DE GOTO BLAS

BLAS (Basic Linear Álgebra Subprograms) es un conjunto de librerías que implementan una multiplicación de matrices en forma secuencial (DGEMM) que es utilizado por el HPL bechmark (High Performance Linpack), a continuación se detalla la instalación de la versión GOTO BLAS sobre la distribución SUSE 9.0 de Linux.

1. Se considera que el usuario "**wolf**" existe.
2. Download GOTO BLAS (**libgoto_p4_512-r0.6.gz**) desde http://www.gotoblas.com y grabarlo en la carpeta ***$HOME/install/***
3. Desempaquetar GOTO BLAS en la carpeta ***/usr/local/mathlib/goto*** (previamente ingresar al sistema como root)

   [wolf@master] su

   [wolf@master] mkdir –p /usr/local/mathlib/goto

   [wolf@master] gunzip $HOME/install/libgoto_p4_512-r0.6.gz
4. Adicionar a tu LD_LIBRARY_PATH : ***/usr/local/mathlib/goto***
5. Download xerbla.f desde www.cs.utexas.edu/users/kgoto/libraries a $HOME/install
6. Copiar y compilar xerbla.f

   [wolf@master] cp $HOME/install/xerbla.f
   *   /usr/local/mathlib/goto*

   [wolf@master] cd /usr/local/mathlib/goto

   [wolf@master] g77 -c xerbla.f
7. Verifica que se genere el compilado ***xerbla.o***

ANEXO C

INSTALACIÓN DE HPL (High Performance Linpack)

HPL es el benchmark más utilizado para medir el rendimiento de un sistema y ser catalogado en la lista de los ***top500 list***, el cual es la lista de los 500 sistemas más potentes en el mundo. A continuación se detalla como instalar el HPL en la distribución SUSE 9.0 de Linux.

1. Se considera que el usuario "**wolf**" existe.
2. Download HPL (***hpl.tgz***) desde http://www.hpl.com y grabarlo en la carpeta ***$HOME/install/***
3. Desempaquetar HPL en la carpeta ***$HOME/dev***

   [wolf@master] cd $HOME/dev

   [wolf@master] md5sum $HOME/install/hpl.tgz

   [wolf@master] tar -zxvf $HOME/install/hpl.tgz
4. Entrar en el directorio donde está desempaquetado HPL

   [wolf@master] cd hpl
5. Configurar HPL (Crear un archivo de configuración)

   [wolf@master] cp setup/Make.Linux_PII_CBLAS

   Make.Linux_P4_goto
6. Realizar algunos cambios en el archivo de configuración ***Make.Linux_P4_goto***

   ARCH = Linux_P4_goto

   TOPdir = $HOME/dev/hpl

   CC = mpicc

   LINKER = mpif77

   LINKFLAGS = $(CCFLAGS) -lm

   MPdir =

   MPinc =

   MPlib =

LAdir = /usr/local/mathlib/goto

LAlib =$(LAdir)/libgoto_p4_512-r0.6.so $(Ladir)/xerbla.o

HPL OPTS =

7. Compilar HPL

   [wolf@master] cd $HOME/dev/hpl

   [wolf@master] make arch=Linux_P4_goto

8. Se genera un ejecutable (***xhpl***) y un archivo de configuración (***HPL.dat***) en el directorio ***$HOME/dev/hpl/bin/Linux_p4_goto***

BIBLIOGRAFÍA

[1] Thomas Sterling. "Beowulf Cluster Computing with Linux" MIT Press, 2003.

[2] Charles Bookman. "Linux Clustering: Building and Maintaning Cluster Linux" New Riders Publishing, 2002.

[3] Salim Hairi – Manish Parashar." Tools and Environments for Parallel and Distributed Computing", John Wiley & Sons, 2004.

[4] The Beowulf HOWTO. *http://tlpd.org/HOWTO/html_single/Beowulf-HOWTO/*

[5] MPICH-A Portable Implementation of MPI
http://www-unix.mcs.anl.gov/mpi/mpichl/

[6] Instrucciones para la Creación de un Cluster LAM-MPI.
http://www.generacio.com/cluster/cluster1.html

[7] HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers.
http://www.netlib.org/benchmark/hpl/

[8] Building and running HPL with GOTO BLAS on Pentium-4/Linux
http://bohnsack.com/writing/libgoto-hpl/

[9] Texas Advanced and Computer Center.
http://www.tacc.utexas.edu/resources/software/

[10] High performance Linux clustering, Part 2: Build a working cluster
http://www-128.ibm.com/developerworks/linux/library/l-cluster2/