Mecánica Computacional Vol. XXI, pp. 2977–2990
S. R. Idelsohn, V. E. Sonzogni and A. Cardona (Eds.)
Santa Fe–Paraná, Argentina, October 2002

SIMULACIÓN COMPUTACIONAL BASADA EN BEOWULF: IMPACTO DE LA TECNOLOGÍA DE RED

Martín O. Silva[*], Carlos García Garino[*] y Armando De Giusti [**]

* LAPIC, Carrera Redes y Telecomunicaciones,
Instituto Universitario Tecnológico – UNC & CONICET
Casilla de Correo 947, 5500 Mendoza
msilva@prmarg.org cgarcia@itu.uncu.edu.ar

** LIDI, Facultad de Informática,
Universidad Nacional de La Plata & CONICET
degiusti@lidi.info.unlp.edu.ar

Key words: Beowulf, Simulación Computacional, cluster, PVM.

Abstract. *El propósito de este trabajo es presentar algunos elementos de la Tecnologías de Red LAN con el fin de analizar el impacto del medio de transmisión empleado en el rendimiento del cluster. Este trabajo se enfoca a tecnología Ethernet y en este contexto se brinda un análisis comparativo de redes compartidas vs. redes conmutadas. Otro aspecto importante que se sugiere en el trabajo es la posibilidad de utilizar más de una placa de red en el master o servidor con el fin de disminuir el overhead asociado a las comunicaciones. P. Finalmente se presentan ejemplos de aplicación para ilustrar los puntos discutidos en el trabajo.*

1 INTRODUCCION

En este trabajo se presentan algunos desarrollos que se han llevado a cabo en el LAPIC (Laboratorio para la Producción Integrada por Computadora) con el fin de poner en marcha una agrupación (*cluster*) de computadoras tipo PC, bajo sistema operativo GNU/Linux, conformando un sistema tipo Beowulf.

El objetivo final a alcanzar con este desarrollo es disponer de una herramienta capaz de realizar tareas de cálculo paralelo con el fin de utilizarla como herramienta de enseñanza en la Cátedra de Arquitectura de Computadoras (Carrera Redes y Telecomunicaciones) y a la vez disponer de una herramienta capaz de resolver de una manera eficiente, económica y robusta problemas de Mecánica Computacional. Con este fin se implementarán algunos paquetes de Álgebra Lineal y también códigos de elementos finitos no lineales con capacidad de grandes deformaciones.

En el trabajo se describe el *cluster*, que ya está implementado, el sistema tipo Beowulf y se analizan en este contexto algunas nociones propias de la Tecnología Ethernet con el fin de estudiar su impacto en las comunicaciones propias del Beowulf.

Los conceptos discutidos en el trabajo se ilustran con ejemplos de aplicación que dan lugar a las conclusiones del mismo.

2 AGRUPACIONES LINUX Y BEOWULF

Beowulf fue un héroe de la mitología germana de gran fortaleza y coraje. Sobrevive hasta nuestros días el mas antiguo escrito de la lengua inglesa que es justamente un poema acerca de él.

Fue el nombre de un proyecto que en 1994 se inició bajo el patrocinio del proyecto de Alto Rendimiento en Procesamiento y Comunicaciones (*HPCC, High Performance Computing and Communications*) de la NASA. Su objetivo era investigar el potencial de las agrupaciones de PC para llevar a cabo tareas de procesamiento importantes que van más allá de las capacidades de las estaciones de trabajo contemporáneas con un mínimo coste. Sterling y Becker[1] fueron los primeros en concebir una supercomputadora con función dedicada a partir de componentes de bienes de mercado masivos y ejecutando un software enteramente libre. Hoy en día el método utilizado en el proyecto Beowulf está ampliamente implementado y es, quizás, la tecnología más importante disponible para agrupaciones hasta el punto que las agrupaciones de arquitectura multicomputadora usadas para cálculo paralelo han heredado el nombre Beowulf.

2.1 Características de un Beowulf[2]

Las características principales de un Beowulf son las siguientes:

- ✓ Los componentes son elementos del mercado masivo.
- ✓ Procesadores dedicados (en vez de robar ciclos de estaciones de trabajo desocupadas).
- ✓ Una red privada dedicada (LAN o WAN o combinación entre redes).
- ✓ No tiene componentes de propietario.
- ✓ Fácil repetición para muchos fabricantes.

- ✓ E/S escalable.
- ✓ Una base de software disponible libremente.
- ✓ Uso de herramientas de ejecución distribuida libremente disponibles con mínimos cambios.
- ✓ Retorno del diseño y de las mejoras a la comunidad.

2.2 Características del Beowulf utilizado en el presente trabajo

Aunque los elementos del software Beowulf [2] se han implementado sobre varias plataformas diferentes, la opción más evidente como base es Linux, y la mayoría de las implementaciones Beowulf utilizan una agrupación de estaciones de trabajo Linux y PC Linux. Tal es el caso del presente prototipo en el que las máquinas están interconectadas en red y no poseen monitor, teclado ni mouse. El hardware empleado se describe más adelante.

Además del sistema operativo de base, este prototipo utiliza el Sistema de Archivos de Red (NFS) para tener en común el espacio de archivos de usuarios /home entre la estación maestra y sus esclavos, con el objeto de que ciertos archivos y programas sean de acceso transparente para todos ellos. Por motivos de eficiencia se eligió que las estaciones esclavas arranquen el sistema operativo desde su disco rígido local y no desde la estación maestra a través de la red, como es el caso de otras arquitecturas semejantes para Beowulf (*diskless*).

El sistema cuenta también con NIS(YP) (Network Information Service) para que los usuarios sean reconocidos en toda la agrupación y compartan a través de la misma el espacio de archivos de usuarios /home.

Existe una relación de confianza en la política de seguridad de la agrupación, es decir que un usuario válido puede ejecutar comandos y programas en forma remota (*rsh*) en cualquier estación de la agrupación sin necesidad de autenticación.

Para soportar computación paralela distribuida se utiliza PVM (Parallel Virtual Machine)[3], que funciona bajo GNU/Linux, están conectadas mediante una red de área local y el protocolo de comunicaciones empleado es TCP/IP.

Las doce estaciones esclavas se basan en micoprocesadores tipo PC Zenith Bull Z-Select 100. Las mismas poseen un microprocesador Intel 80486 DX2 a 66 Mhz, bus ISA, 16 MB de RAM, disco rígido de 340 MB y placa de red SMC-Ultra a 10 Mbits/s. La estación maestra es una Zenith Bull Z-Station 500, la cual posee un microprocesador Intel 80486 DX2 a 66 Mhz, buses PCI e ISA, 16 MB de RAM, disco rígido de 2 GB y placa de red SMC-Ultra a 10 Mbits/s para la conexión al *cluster* y una placa de red NE2000 genérica a 10 Mbits/s para la conexión a la red del ITU.

La tecnología de red empleada es Ethernet[4] , los cables son del tipo UTP categoría 5. Para materializar el bus o canal propio de ethernet se han dispuesto en el centro de la estrella un concentrador (hub) Kingston EheRx KNE-16TP/WG que posee 16 bocas bajo norma 10Base-T o bien un conmutador Nortel BayNetwork BayStack 450-12T que trabaja bajo 100BaseT.

Además del *cluster* descrito en los párrafos anteriores, el cual se dispone de manera dedicada, se cuenta con otro de producción basado en aproximadamente 20 equipos del tipo Pentium III y/o Durón, el cual se puede utilizar como complemento de tareas de enseñanza

durante el tiempo ocioso del laboratorio donde están instaladas.

Entre otras aspectos puede mencionarse que los equipos que constituyen el cluster son semejantes entre sí, lo que garantiza una gran homegeneidad. Los mismos se recuperaron al actualizarse el laboratorio de CAD-CAM-CAE del Instituto Tecnológico Universitario, lo cual ilustra la potencialidad de Beowulf y el espíritu del mismo. De esta forma se ha puesto en marcha un prototipo para investigación, desarrollo y enseñanza.

3 TECNOLOGÍAS DE RED ETHERNET

En este punto se discuten algunas nociones básicas de la tecnología de redes Ethernet, seguramente la más empleada en redes de área local (LAN), debido a su simplicidad y bajo costo. La sencillez de esta tecnología conlleva algunos inconvenientes asociados en ciertos casos al acceso al medio, lo que afecta su eficiencia global.

A partir de la introducción de conceptos bien conocidos, disponibles incluso en libros de textos, como son las nociones de acceso al medio, colisión, dominio de colisión, segmento, etc., se introducen las características distintivas de las redes compartidas y de las redes conmutadas.

En lo que se refiere al medio físico, capa 1 del modelo OSI, este trabajo se restringe al caso de cables de cobre del tipo par trenzado no blindado (UTP), de categoría 5, y que obligan a utilizar un concentrador (*hub*) para conformar la topología típica en estrella de estos casos para materializar el bus o canal propio de la topología lógica de Ethernet. Una alternativa, mucho más eficiente y costosa es reemplazar el concentrador por un conmutador.

3.1 Acceso al medio

Prácticamente toda la electrónica reside en la placa de red, a excepción de los concentradores y/o conmutadores necesarios para materializar la red. Cabe aclarar que los concentradores y conmutadores intentan mejorar el desempeño y rendimiento de un simple conexionado en topología de bus según las normas 10Base2 y 10Base5.

Las ventajas antes mencionadas se contrastan con un gran inconveniente: el acceso al medio. A diferencia de otras tecnologías el acceso al medio no está arbitrado en Ethernet, ya que se regula mediante el protocolo CSMA/CD (Múltiple acceso al medio con sensado de portadora y detección de colisiones).

Esta metodología permite transmitir a la vez a todas las estaciones (acceso múltiple) para lo cual debe escuchar la red (detección de portadora) y detectar las colisiones que pueden ocurrir en caso de que dos puestos transmitan simultáneamente (detección de colisiones). Esta forma de acceder al medio impide que las placas de red transmitan y reciban datos a la vez, por lo cual deben funcionar en la modalidad *half-duplex*.

Se suele decir que las estaciones compiten por el acceso al medio conformando de esta forma una red compartida. Una alternativa, mucho más eficiente y costosa, que permite incrementar el ancho de banda y disminuir en gran medida las colisiones, o directamente eliminarlas, consiste en reemplazar los concentradores por conmutadores (*switches*). Estos equipos que operan a nivel de la capa de enlace o capa 2 del modelo OSI simulan circuitos

virtuales punto a punto entre los equipos y el conmutador, permitiendo además que se utilice el medio para transmitir y recibir datos a la vez, en la modalidad *full-duplex*.

La discusión de ventajas y desventajas del empleo de redes compartidas versus redes conmutadas es un tema de interés actual en la tecnología de redes LAN y en la referencia[5] se ha recopilado información de interés.

3.2 El rendimiento de 802.3

Ethernet ha sido estandarizado por IEEE en la recomendación 802.3. Un estudio del desempeño de 802.3 puede verse en Tanenbaum[6] del que podemos sintetizar que para k estaciones listas para transmitir, con una probabilidad p de que lo hagan durante una ranura de contención, la probabilidad A que de una estación adquiera el canal durante esa ranura es de

$$A = kp(1-p)^{k-1}$$

A es máxima cuando $p=1/k$, con $A \rightarrow 1/e$ conforme $k \rightarrow \infty$. La probabilidad de que la ranura de contención tenga exactamente j ranuras de alojamiento en él es de $A(1-A)^{j-1}$, por lo que el número medio de ranuras por contención está dado por

$$\sum jA(1-A)^{j-1} = 1/A$$

Ya que cada ranura tiene una duración de 2τ, la ranura media de contención, w, es $2\tau/A$. Suponiendo una p óptima, el número medio de ranuras de contención nunca es mayor que e, por lo que w es, cuando mucho, $2\tau e \cong 5.4\tau$. Si la trama media tarda P segundos en transmitirse, cuando muchas estaciones tienen tramas por enviar,

$$\textit{Eficiencia del canal} = P/(P+2\tau/A)$$

Cuanto mayor sea la longitud del cable, mayor será la ranura de contención. Al no permitir más de 2.5 Km de cable y cuatro repetidores entre dos transceptores, el tiempo de ida y vuelta puede fijarse en 51.2 μseg, que a 10 Mbps corresponde a 512 bits o 64 bytes, el tamaño mínimo de la trama.

3.3 Comunicación entre estaciones y servidores

Otro punto conflictivo que suele producirse en las arquitecturas tipo cliente/servidor es el cuello de botella que aparece cuando varias estaciones necesitan comunicarse simultáneamente con el servidor. En el caso de un *cluster* tipo Beowulf este problema aparece, por ejemplo, en el contexto de arquitecturas SIMD[7], cuando se deben sincronizar los distintos procesadores y transmitir datos al maestro.

El problema mencionado es característico de aplicaciones comerciales como son los sistemas de Bases de Datos y existen algunas publicaciones, que sugieren diferentes maneras de aliviar el problema ya sea en presencia de concentradores o bien de conmutadores.

Una de las tendencias actuales es conformar las llamadas *Granjas de Servidores*, con el fin de aumentar el ancho de banda disponible para el servidor y disminuir el cuello de botella. La idea básica es utilizar varias placas de red, o bien una multiplaca, para comunicar el servidor al

concentrador o, preferentemente, a un conmutador. De esta manera se puede aumentar notablemente el ancho de banda disponible para el acceso al servidor y, además, trasmitir y recibir datos a la vez. Los conmutadores suelen poseer disponibilidades para sincronizar y agrupar estos canales en uno solo para facilitar su administración.

3.4 Tiempos relativos en un ambiente cliente-servidor distribuido

En la tabla se muestra una jerarquía típica de recursos de computación en un ambiente cliente-servidor distribuido. Dado que los tiempos de acceso son diferentes, para resaltarlo se relaciona 60 ns a 1 segundo.

Tipo de dispositivo	Tiempo típico de servicio	Relativo al segundo
Memoria de acceso aleatorio	60 ns	1 s
Llamada a procedimiento	1 μs	16 s
Memoria expandida	25 μs	7 m
RPC local	100 μs	28 m
Disco estado sólido	1 ms	5 h
Disco “cacheado”	10 ms	2 d
Disco magnético	25 ms	5 d
Disco via LAN serv alta vel	25 ms	5 d
Disco via LAN serv típico	35-50 ms	7-10 d

Tabla 1

4 PVM - PARALLEL VIRTUAL MACHINE

PVM [3] es un paquete de software que permite a una colección heterogénea de computadoras seriales, paralelas y vectoriales interconectadas mediante una red, actuar como si fuera una única computadora paralela, también llamada *máquina virtual*. Para poder llevar a cabo su función se vale del *pasaje de mensajes* entre las computadoras que forman la agrupación

4.1 El modelo de mensajes de PVM [3]

Los *daemons* de PVM y las tareas pueden componer y enviar mensajes de longitudes arbitrarias conteniendo datos. Estos datos pueden ser convertidos utilizando XDR[8] (eXternal Data Representation) al pasar entre máquinas con formato de datos incompatibles. Los mensajes se marcan al momento de ser enviados con un código entero definido por el usuario y puede ser seleccionado para recepción por su dirección fuente o marca.

El remitente de un mensaje no espera un reconocimiento por parte del destinatario, sino que continua tan pronto como el mensaje haya sido enviado a la red y el área de almacenamiento

del mensaje pueda borrarse o reusarse en forma segura. Los mensajes se almacenan en el punto de destino. PVM despacha los mensajes en forma confiable suponiendo que el destinatario existe. Se preserva el orden de los mensajes desde cada remitente a cada destinatario en el sistema, si una entidad envía varios mensajes a otra, serán recibidos en el mismo orden.

4.2 Los protocolos de comunicación de PVM

La comunicación en PVM está basada en TCP[9], UDP[10] y en *sockets* de Unix. Los *daemons* de PVM se comunican entre sí utilizando *sockets* UDP. UDP es un servicio de despacho no confiable que puede perder, duplicar o reordenar paquetes, por lo tanto se utiliza un mecanismo de reconocimiento y reintento. UDP también limita la longitud del paquete, por lo tanto PVM fragmenta los mensajes largos.

Por otra parte una tarea se comunica con sus *daemons* PVM y con otras tareas mediante *sockets* TCP. Se usa TCP porque despacha datos en forma confiable. UDP puede perder paquetes aun dentro de la misma máquina. Un despacho no confiable requiere reintentos en ambas puntas y como las tareas no pueden interrumpirse mientras están procesando para realizar operaciones de entrada/salida, no es posible utilizar UDP. La versión 3.3 de PVM introdujo el uso de flujos *sockets* de Unix como una alternativa a TCP para comunicaciones locales para mejorar la latencia y la tasa de transferencia.

4.3 Las funciones de PVM para el pase de mensajes

Hasta la versión 3.3 de PVM las únicas funciones para el pase de mensajes disponibles eran pvm_send() y pvm_recv(). A partir de esa versión se han agregado algunas mas.

El envío de un mensaje requiere tres pasos: primero debe inicializarse un buffer de PVM mediante una llamada a la función pvm_initsend(). Segundo, el mensaje debe ser "empaquetado" desde el espacio de datos del usuario al buffer de PVM utilizando cualquier combinación de las rutinas que comienzan con pvm_pk*(). PVM se ocupa de cualquier codificación y fragmentación de datos. Tercero, el mensaje completo se envía a otro proceso con pvm_send().

La recepción del mensaje involucra dos pasos. Primero, el mensaje entrante debe ser aceptado por pvm_recv(), que hace una recepción bloqueante, o por pvm_nrecv() que hace una recepción no-bloqueante. Segundo, una vez que el mensaje ha llegado debe ser "desempaquetado" dentro del espacio de datos del usuario con una combinación de las funciones pvm_upk*(). Durante la inicialización del buffer de PVM, el usuario puede elegir entre tres formas diferentes de empaquetar los datos en este buffer, dependiendo del parámetro pasado a pvm_initsend(). El modo de empaquetado por omisión es PvmDataDefault. El dato es empaquetado desde el espacio de usuario al buffer de PVM y se codifica de acuerdo con el formato XDR. Este modo permite la comunicación sobre una red heterogénea (es decir, un conjunto de computadoras de las cuales al menos dos no tienen el mismo formato de datos).

Un segundo modo es PvmDataRaw; este modo es similar al *default*, pero se salta el paso de

codificación. Entonces PvmDataRaw solo puede usarse entre máquinas de formatos de datos compatibles. Siempre es mas eficiente usar PvmDataRaw en PVM, cuando se dispone de sistemas de procesamiento paralelo masivo (MPP). También es eficiente en el caso del hardware utilizado en el presente trabajo que se basa en una red homogénea.

4.4 Las funciones pvm_psend() y pvm_precv()

A partir de la versión 3.3 de PVM es posible enviar y recibir mensajes en un solo paso utilizando las funciones pvm_psend() y pvm_precv(). Los mensajes procesados por estas rutinas deben intercambiarse entre máquinas con formato de datos compatible. Además, ya que no hay empaquetamiento, los datos enviados deben estar contiguos en el espacio de memoria del remitente. En otras palabras, pvm_psend() puede usarse para enviar un arreglo de un tipo de datos dado a un determinado destino, lo cual es un tipo de mensaje muy común en una aplicación paralela. Esta característica no puede usarse entre máquinas con formato de datos incompatibles ya que no se resuelven las diferencias en la codificación de datos.

5 LOS PROGRAMAS UTILIZADOS

Para medir el ancho de banda y tiempos de latencia se utilizaron los programas “bwtest” y “timing”, escritos en lenguaje C, provistos en el paquete PVM. El primero mide tiempos transcurridos de ida y vuelta (RTT- *roundtrip time*), medidos desde una tarea remitente a otra destinataria, mediante un “ping-pong” de mensajes. La función utilizada para obtener la hora del reloj de la máquina es pvmgetclock(), y el envío en tres etapas de los mensajes en forma estándar de PVM implica el llamado a las funciones pvm_initsend(), pvm_pack(), pvm_send() y la recepción el llamado a las funciones pvm_recv(), pvm_unpack() y genera la respuesta nuevamente con pvm_initsend(), pvm_pack() y pvm_send(), para que finalmente se complete la ida y vuelta con pv_recv() y pvm_unpack desde la tarea remitente que inició el pasaje de mensajes. Esta forma estándar es la que en PVM se denomina PvmDataDefault y es efectuada por el programa “dbwtest” (una variante de “bwtest”).

Para efectuar las mediciones sin el empaquetamiento de datos, es decir en la modalidad PvmDataRaw, se emplea el programa “rbwtest”, y el pasaje de mensajes en tres pasos.

El programa “pbwtest” trabaja en un solo paso, y llama a las funciones pvm_psend(), pvm_precv(), pvm_psend() y pvm_prec().

Los tres programas van incrementando el tamaño del mensaje desde 0, 8, 80, 800, ... hasta 800000 bytes.

El programa “timing” está estructurado de una manera diferente: él mismo genera una tarea destinataria llamada “timing_slave” cuyo propósito es hacer eco de todo lo que reciba de “timing” y éste va generando mensajes de distintas longitudes; “timing” luego de enviar un mensaje con una llamada a pvm_send() espera su eco con pvm_recv(). De esta manera se genera mucho tráfico en la red, sobretodo al comienzo de la ejecución en la que los mensajes son muy cortos.

Cabe acotar que en este modelo de mensajes, con distintos tamaños medidos en la aplicación, sufren un incremento, sobrecarga o *overhead*, que se debe al pasaje por cada capa

del modelo TCP/IP, que agregan su propia información. En consecuencia a nivel de la capa 2 (Ethernet) la trama incluye sobrecargas debidas al encapsulamiento. En general, la longitud de los mensajes empleados en estos ejemplos superan a la de la trama Ethernet por lo que el protocolo se ve obligado a fragmentar los mismos para enviarlos.

Esta situación favorece aún más la aparición de colisiones, y a la vez es una aproximación más cercana a la práctica basada en cálculos paralelos, que la efectuada solamente en el medio de transmisión.

6 EJEMPLOS

Con el fin de ilustrar las ideas discutidas se realizan una serie de experimentos con el cluster, que permiten medir colisiones, ancho de banda y tiempo de roundtrip (RTT).

Con el propósito de aumentar el tráfico en la red y forzar su capacidad, se ejecutaron varias copias de estos programas en forma simultánea. De esta manera se incrementa la competencia por el uso del medio de comunicación y se incrementan las colisiones en el mismo.

Con el fin de contrastar el rendimiento del cluster bajo redes compartidas y redes conmutadas, se realizó el experimento utilizando un *hub* y luego un *switch* bajo las mismas condiciones. y poder así cuantificar en qué medida se mejora el desempeño de la red en el pasaje de mensajes.

6.1 Calculo de Colisiones

Para medir las colisiones que aparecen en la red se emplean los programas *dbwtest*, *rbwtest*, *pbwtest* y *timing* que se han descrito en el punto anterior.

Las Tablas 2 y 3 cuantifican las colisiones registradas durante la simulación utilizando un *hub* y un *switch* respectivamente. Tal como era de esperar, en el caso de redes conmutadas, el número de colisiones baja notoriamente pero no se anula.

Número de colisiones con hub				
Nº de tareas	Dbwtest	rbwtest	Pbwtest	Timing
2	13	8	2	74
4	184	236	181	2354
6	34180	34793	36458	12370
8	48345	68494	65956	24462
10	82808	110460	105558	36397
12	128564	155837	152971	53967

Tabla 2: Cálculo de colisiones para la red compartida.

Número de colisiones con switch				
Nº de tareas	Dbwtest	Rbwtest	Pbwtest	Timing
2	4	8	4	432
4	7	5	4	2718
6	13	18	10	5297
8	8	5	3	8881
10	13	4	5	6082
12	10	10	11	10085

Tabla 3: Cálculo de colisiones para la red conmutada

Es importante destacar que en el experimento se emplearon placas *half-duplex*, de esta manera permanecen dominios de colisión de orden dos conformados por las bocas del switch y del PC. En el caso de placas full-duplex las colisiones deberían disminuir todavía más o directamente desaparecer.

El gráfico de la Figura 1 muestra cómo aumentan las colisiones en el *hub* a medida que se incrementan las tareas, según los datos de la Tabla 1. Sin embargo se observa que no se evidencian diferencias significativas entre las distintas modalidades del envío de mensajes, es decir entre el pasaje de mensajes en tres pasos o en uno solo, y empaquetando y sin empaquetar. No se grafican los resultados de la Tabla 2 por ser sus valores casi nulos con respecto a los de la Tabla 1.

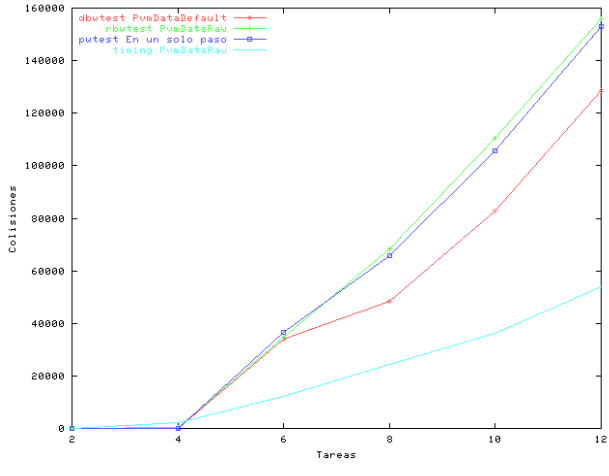

Figura 1: Colisiones en función del número de tareas – Red Compartida

6.2 Medición del RoundTrip

Con el propósito de medir este efecto se utilizó los mismos programas del ejemplo anterior con la excepción de *timing*. Los programas se ejecutaron tanto en redes compartidas como en redes conmutadas.

En la Figura 2 podemos ver que el tiempo de ida y vuelta del pasaje de mensajes aumenta considerablemente con el incremento de las tareas, para un tamaño de mensajes de 800000 bytes para el caso de un *hub* en la red. Cuando se emplea un *switch* el rendimiento de la red es constante.

También se observa una latencia propia del *switch* en los casos de poco tráfico, ya que se verifican tiempos similares o más elevados que en el caso del *hub*, aspecto que debe confirmarse con experimentos específicos.

También en este caso se observa que los modos de pasar los mensajes de PVM prácticamente no influyen en los tiempos registrados, al punto que se superponen las curvas de "rbwtest" y "pwtest" para el caso *switch*. Estos resultados son comparables con los obtenidos por Sterling y Becker en un Beowulf de similares características, en el que se muestra el rendimiento del mismo pero con una doble red Ethernet, una en 10baseT con *hub* y la otra en 10base2, y utilizando paquetes desde 64 a 16384 bytes.

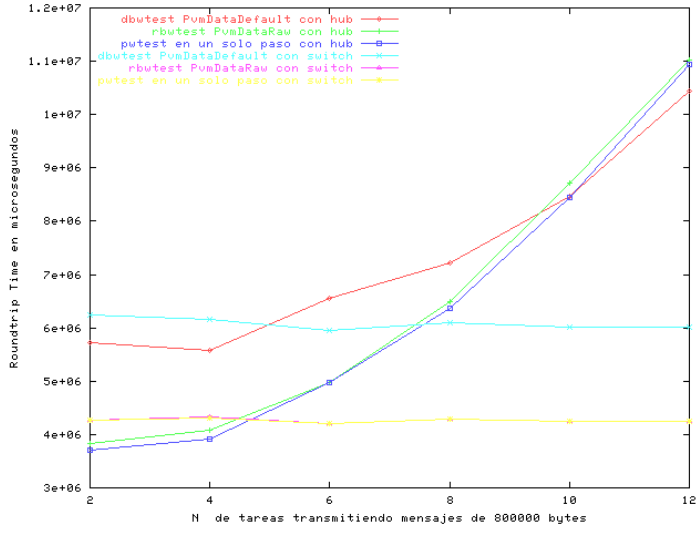

Figura 2: Evolución del tiempo de RoundTrip en función del número de tareas

6.3 Cálculo del Ancho de Banda

Para medir el ancho de banda también se utilizan los programas *dbwtest*, *rbwtest*, *pbwtest*, tanto para redes compartidas como para redes conmutadas.

La Figura 3 muestra el ancho de banda en función del número de tareas, para un tamaño de mensaje de 800000 bytes. En la misma se observa como decae el ancho de banda disponible en el caso de la red compartida. Para el caso de la red conmutada este parámetro se mantiene prácticamente constante.

Nuevamente se observa la presencia de la sobrecarga introducido por la electrónica activa del *switch* cuando la red está poco utilizada. Tampoco en este caso se observa influencia de las distintas formas de enviar mensajes en los resultados obtenidos. Estos resultados son comparables con los obtenidos por Sterling y Savarese[11] utilizando un hub y tamaños de mensajes más chicos que en el presente trabajo.

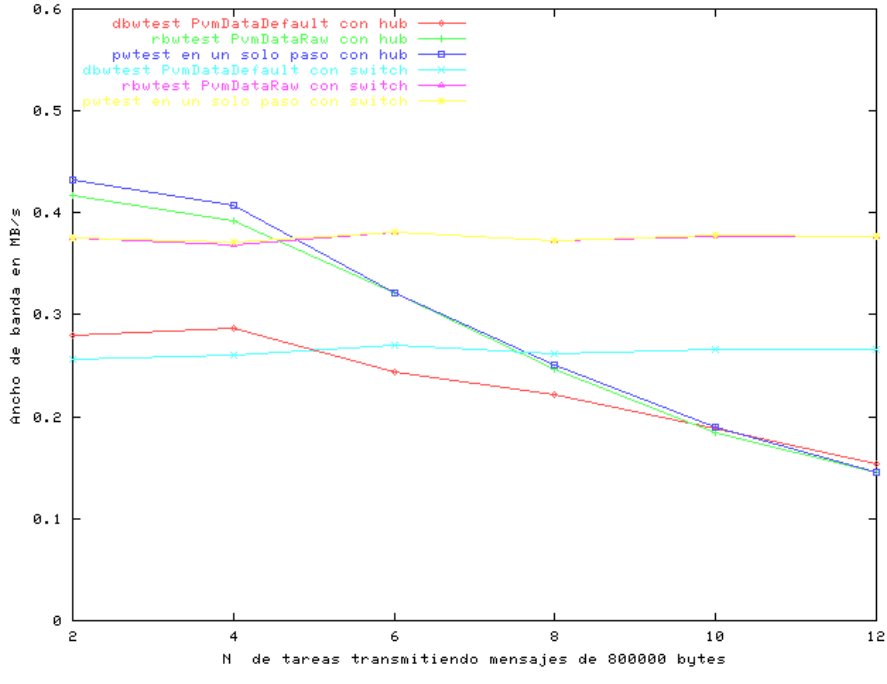

Figura 3: Ancho de Banda en función del número de tareas.

En la Figura 4 podemos observar el ancho de banda en Megabytes logrado en función del tamaño del mensaje PVM, medido para 12 tareas. Como se ha señalado anteriormente para el

caso de mensajes muy grandes, mayores que la trama máxima, Ethernet se ve obligado a fragmentar y esto favorece a la aparición de más colisiones. Se deben reenviar tramas y consecuentemente se disminuye el ancho de banda. En el caso de la red conmutada no se observa este efectro.

Por otra parte no se observan diferencias notorias en el modo que tiene PVM de empaquetar los mensajes, ya que los tiempos relativos de empaquetamiento son despreciables con respecto a los de transmisión en la red.

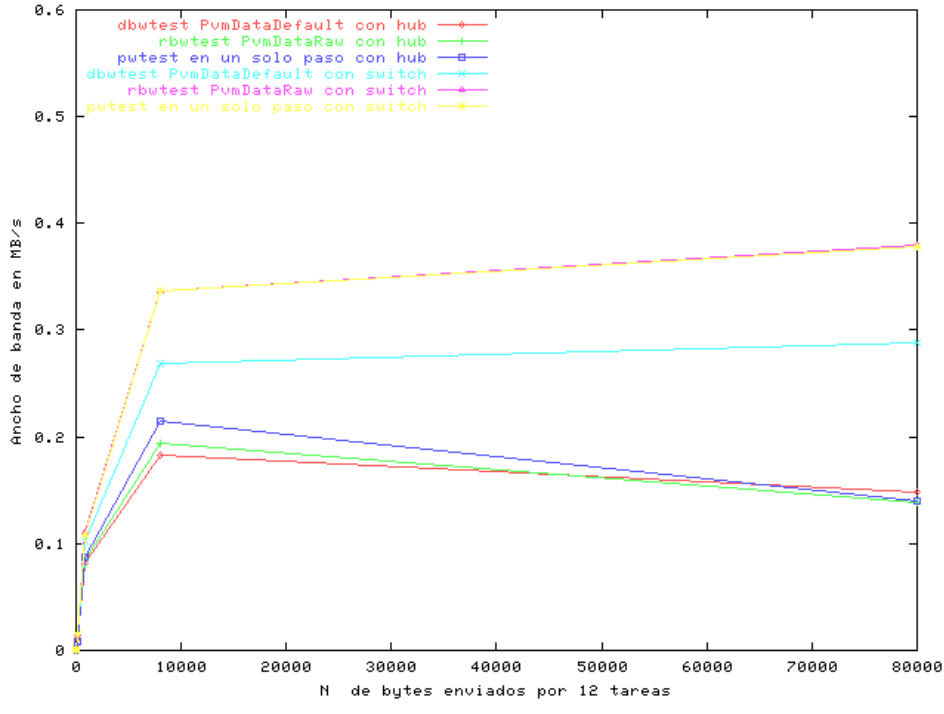

Figura 4: Ancho de Banda en función del número de tareas.

El ancho de banda medido, con el programa timing, alcanza 6.44 Mbits/seg., que no alcanza el valor nominal de 10 Mbits/segundo, como se ha mostrado en el punto 3.2, siguiendo a Tanenbaum, y lo reporta el trabajo de Yommi y Sonzogni[12].

7 CONCLUSIONES

El presente trabajo a servido de esfuerzo de validación, ya que ha permitido corroborar

empíricamente distintos enunciados de la Teoría de Redes.

La implementación de este tipo de *clusters* es posible aún con hardware antiguo.

Los experimentos realizados permiten confirmar mediante experimentos rigurosos las presunciones empíricas acerca del mejor rendimiento de las redes conmutadas frente a las redes compartidas.

En el caso de redes compartidas se deberían utilizar topologías semejantes a las de las granjas de servidores, con el fin de segmentar el dominio de colisión y consecuentemente disminuir las colisiones, aspecto que se ilustrará en el momento de la presentación.

Estos resultados se han obtenido con agrupaciones altamente homogéneas y parece importante investigar el comportamiento de otro tipo de agrupaciones.

8 REFERENCIAS

[1] D. Becker, T. Sterling, D. Savarese, J. Dorband, U. Ranawak, C. Packer. "Beowulf: a parallel workstation for scientific computation". Proceedings of the 1995 International Conference on Parallell Procesing (ICPP), August 1995, pp. 11-14.

[2] Ridge, D., *et al.* "Beowulf: Harnessing the power of Parallelism in a Pile-of-PCs" Proceedings IEEE Aerospace, 1997.

[3] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: A Users´ Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994. Also available electronically, the url is http://www.netlib.org/pvm3/book/pvm-book.html.

[4] C. Spurgeon. "Ethernet: the definitive guide". Ed. O'Reilly. ISBN 1-56592-660-9.

[5] C. García Garino: "Redes Compartidas vs Redes Conmutadas". Informe Interno RyT-ITU-0100, Mendoza, 2000.

[6] A. S. Tanenbaum. "Computer networks". ISBN 0-13-349945-6. Ed. Prentice-Hall. 1996.

[7] M.J. Flynn: "Some Computer Organizations and Their Effectiveness", IEEE Trans. On Computers, vol. C-21, pp948-960, september, 1972.

[8] Sun Microsystems, Inc. XDR: External Data Representation Standard. RFC 1014, Sun Microsystems, Inc., June 1987.

[9] J. Postel. Transmission Control Protocol. RFC 793, Information Sciences Institute, September 1981.

[10] J. Postel. User Datagram Protocol. RFC 768, Information Sciences Institute, September 1981.

[11] T. Sterling, D. Savarese, D. Becker, B. Fryxell, K. Olson. "Communication Overhead for Space Science Applications on the Beowulf Parallel Workstation". Proceedings of the Fourth IEEE Symposium on High Performance Distributed Computing (HPDC), August 1995, pp. 23-30.

[12] A. Yommi, V. Sonzogni. "Evaluación de una red de cálculo distribuido usando PVM". Actas del Congreso Mecom 99, Mecánica Computacional, Volumen Especial, AMCA, Santa Fe, 1999.