Posibles Optimizaciones de Comunicaciones entre Procesos para Cómputo Paralelo en Clusters

Trabajo Final para la Especialización en Interconexión de Redes y Servicios

Facultad de Informática

UNIVERSIDAD NACIONAL DE LA PLATA

Autor. Roberto José Roque Paz

Director. Dr. Fernando G. Tinetti

Tabla de contenidos

Lista de figuras

Capítulo 1. Introducción

1.1. Problemática de las comunicaciones en los clusters basados en paso de mensajes

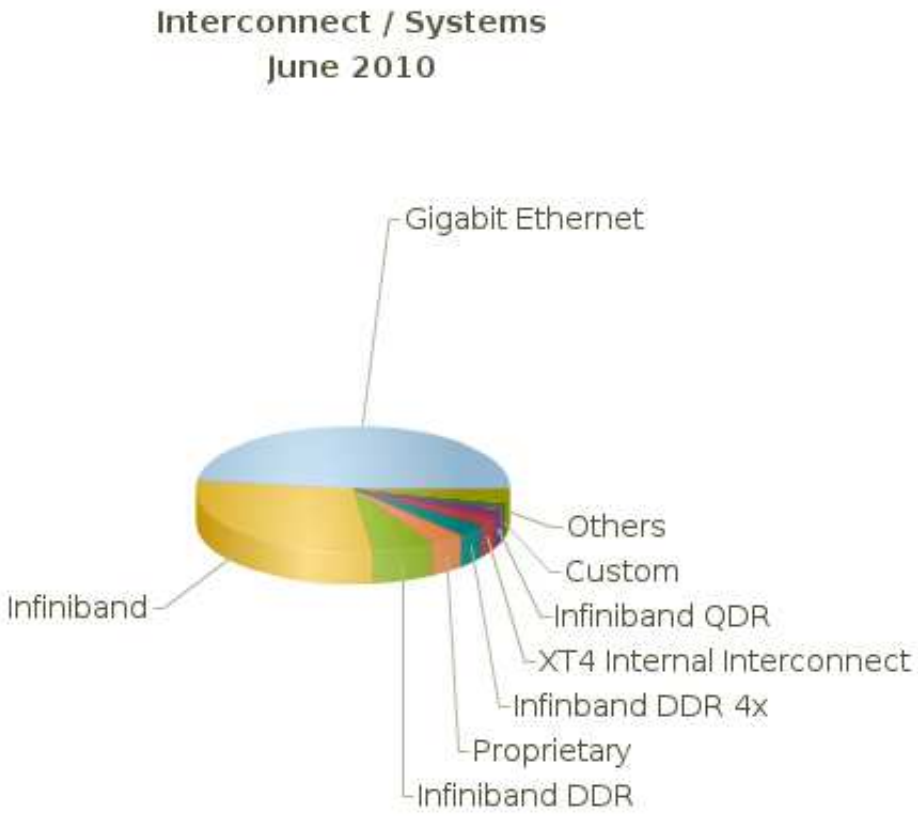
La constante evolución de la computación de alto rendimiento ha desarrollado múltiples alternativas, siempre en pos de un objetivo común: mejorar el rendimiento del cálculo paralelo. No existe un único modelo que provea una respuesta universal a esta problemática, sino que existen múltiples modelos, algoritmos, mecanismos, arquitecturas, etc., que son más o menos adecuados según el problema planteado.

A modo de ejemplo, podemos citar el caso de las arquitecturas de clusters basadas en nodos con memoria compartida. En épocas recientes, ha existido un gran empuje en las tecnologías "multi-core", ya que la mejora constante en los niveles de integración a nivel electrónico, permiten un mejor uso del espacio para múltiples elementos procesadores en una misma pastilla de silicio. Esto sin embargo, no se logra con la misma eficiencia para las áreas de almacenamiento, específicamente la memoria principal, memoria caché, etc. Esta dirección planteada por la evolución tecnológica, genera pastillas de silicio con cada vez mayor cantidad de elementos procesadores, con un área de memoria común compartida entre todos ellos. Este tipo de arquitectura de memoria compartida, favorece el uso de herramientas de programación en clusters como "OpenMP" por ejemplo, donde se privilegia el acceso compartido al área de datos [HPCMMM].

Otro escenario muy difundido, plantea el uso de elementos procesadores con memoria distribuida. En ese caso, los elementos procesadores están unidos típicamente por una red de datos de alta velocidad. Los datos son enviados entre los distintos nodos procesadores mediante algún mecanismo de paso de mensajes. Este tipo de escenarios son mejor aprovechados mediante el uso de herramientas como MPI, y es el que nos interesa particularmente para el análisis de este trabajo [HPCMMM].

Típicamente, un cluster de memoria distribuida basado en red, es un conjunto de computadoras interconectadas por una red de alta velocidad. En la actualidad, la tendencia en la tecnología de red empleada es Infiniband, aunque todavía existen muchos clusters basados en redes Ethernet: al mes de junio de 2010, casi la mitad de los sistemas relevados por el sitio de referencia "top500" están basados en Gb/s Ethernet [Top500].

Figura 1-1. Arquitectura de conexionado de los sistemas listados en top500. Junio de 2010.



Si además tenemos en cuenta que el programa utilizado por *top500* para medir el rendimiento del sistema es *Linpack*, y la ejecución de este último requiere del uso de una implementación de MPI (top500 sugiere "MPICH") y de la librería *CBLAS* para cálculo algebraico, cualquier elemento que permita aumentar el rendimiento de clusters que utilicen MPI sobre Gb/s Ethernet tendrá un gran universo de aplicabilidad.

En términos de velocidad de transferencia de datos, los tiempos empleados para trasmitir un dato desde un nodo procesador a otro, son varios órdenes de magnitud superior al tiempo que el procesador utiliza para realizar el cálculo. En otras palabras: en el mismo tiempo empleado para enviar un byte de datos, podríamos realizar miles de operaciones de cálculo. Es por ello que la transmisión de datos por red en un cluster debe estar plenamente justificada. Esto se logra mediante la elección inteligente del algoritmo paralelo empleado para solucionar nuestro problema de cómputo. De todas formas, muchos problemas de cálculo restringen la variedad en la elección de algoritmos a emplear, por lo que siempre es deseable una disminución en los tiempos de comunicación por red.

1.2. Uso de la red en MPI

Las implementaciones más difundidas de MPI están basadas en el uso de protocolos de la familia TCP/IP. Esta familia de protocolos es bien conocida. Entre sus características principales, podemos mencionar su versatilidad para adaptarse a distintos escenarios de red, lo que le ha posibilitado mantenerse en uso creciente sin modificaciones sustanciales, por más de 30 años.

Esta versatilidad, si bien adecuada para un uso genérico del protocolo, puede ser excesiva o contraproducente en un ámbito de clusters, debido a las características típicamente homogéneas y acotadas de este último. Muchas funcionalidades relativas a control de flujo, ruteo, etc. contempladas en el diseño de la familia de protocolos TCP/IP, pueden resultar innecesarias para su uso en la red local de un cluster. Como estas funcionalidades aportan una cuota de sobrecarga a la red, esto sugiere que se pueden realizar optimizaciones al protocolo para su uso específico en un cluster.

Ciertas optimizaciones de hecho existen. Estas optimizaciones se apoyan en condiciones de prueba o funcionamiento particulares, y es así como proveen mejoras en el rendimiento del cluster.

1.3. Caracterización de una red de datos en términos del rendimiento

Entre los distintos parámetros o valores numéricos que permiten cuantificar las propiedades de una red de datos, existen dos que permiten estimar su rendimiento, y estos son la *latencia* y el *ancho de banda*. En este caso, el rendimiento está considerado en términos de los tiempos de transferencia de datos entre los procesadores involucrados en el cluster. Estos parámetros deberán ser tenidos en cuenta si aspiramos a cuantificar las mejoras en el rendimiento de nuestro cluster.

La estricta definición de estos parámetros en el ámbito de un cluster, es la siguiente:

1. *Latencia*: es el tiempo que tarda una señal desde el comienzo de su transmisión en el nodo de origen, hasta el comienzo de la recepción en el nodo de destino. Este parámetro está condicionado por aspectos de la física (velocidad de la luz, por ejemplo), y suele ser difícilmente modificable o mejorable sin que medie un cambio de la tecnología empleada en la comunicación. En términos estrictos del dato empleado para el cálculo, este parámetro incluye la sobrecarga introducida por las distintas capas de los protocolos de red empleados, ya que estas capas no son útiles para el cálculo, y solo colaboran a aumentar la latencia.
2. *Ancho de banda*: es la cantidad de información que puede ser transferida por unidad de tiempo.

Roger Hockney propuso un modelo para estimar los valores de latencia y ancho de banda, a partir de mediciones sencillas en el cluster. Este modelo se utiliza en comunicaciones punto a punto, tal como es nuestro caso de

análisis para los comandos SEND/RECV de MPI. El modelo se vale de un mecanismo de *ping-pong* entre dos nodos, para medir el tiempo de ida y vuelta de un mensaje [Hockney].

Entre las ventajas de este método, podemos mencionar que no se requiere de una sincronización entre los relojes de los distintos nodos, ya que la medición del tiempo requerido se realiza en el primer nodo. Por otra parte, estamos asumiendo que no existen asimetrías en la comunicación en un sentido y en el otro, entre ambos nodos.

Según este modelo, el tiempo empleado para transmitir un mensaje de tamaño "s" está linealmente relacionado con su tamaño, y se puede estimar a partir de la siguiente fórmula:

$t(s) = \alpha + s/b$

α: latencia

b: ancho de banda asintótico

A su vez, *1/b* es el tiempo de transferencia por cada byte transferido. De esta manera, si medimos el tiempo empleado para transmitir mensajes de distinto tamaño "s", se pueden obtener los valores de *l* y *b*.

De la fórmula propuesta por Hockney, se desprende que la transferencia de datos entre dos nodos, va a tardar como mínimo un tiempo *l*. Tal como se refiriera al comienzo de la introducción, este valor de *l* representa un valor umbral: si el tiempo de cálculo invertido en los datos transferidos es menor a *l*, entonces no ganamos nada con transferir estos datos, y hubiese sido más eficiente realizar el cálculo de manera local. Este valor umbral nos indica la granularidad mínima que posee nuestro programa paralelo.

Cualquier análisis ulterior que se realice con fines comparativos utilizará estos parámetros para identificar las mejoras, y este método para obtener tales parámetros.

1.4. Otros elementos que afectan el rendimiento

Además del aumento en la latencia, existen muchos otros parámetros que afectan el rendimiento en un proceso de cálculo de alto rendimiento. Entre ellos podemos citar [HPCMMM]:

1. *Overhead*: sobrecarga producida por la gestión del paralelismo. Este tiempo no es empleado por el cálculo en sí, sino por la implementación en paralelo de un algoritmo que parte de una concepción secuencial (ej. sincronización y scheduling).
2. *Contention*: ciclos de tiempo malgastados en la espera para el acceso a un recurso compartido (ej. conflicto en el acceso a bancos de memoria, acceso compartido a canales de comunicación).

3. *Starvation*: paralelismo insuficiente, generalmente debido a una mala implementación del algoritmo, un inadecuado balanceo de carga entre los procesadores, o bien por el efecto de la ley de Amdahl.

Estos tres últimos aspectos mencionados, si bien no menos importantes a la hora de impactar en el rendimiento, no serán tenidos en cuenta para el análisis de este trabajo que se enfoca de manera exclusiva en el impacto de la red sobre el rendimiento.

1.5. Objetivos y antecedentes

Los objetivos planteados para este trabajo se enumeran a continuación.

General

- Identificar patrones y mecanismos de mejora en las comunicaciones de un cluster MPI

Específicos

- Manejo de herramientas para cuantificar el rendimiento de una implementación de MPI.
- Estimación de los efectos positivos en las comunicaciones, producidas por optimizaciones conocidas.
- Inferencia de posibles lineas de investigación respecto a aspectos que se puedan mejorar en la implementación elegida, con particular hincapié en las comunicaciones punto a punto.

Capítulo 2. Análisis general del uso de red en un cluster MPI

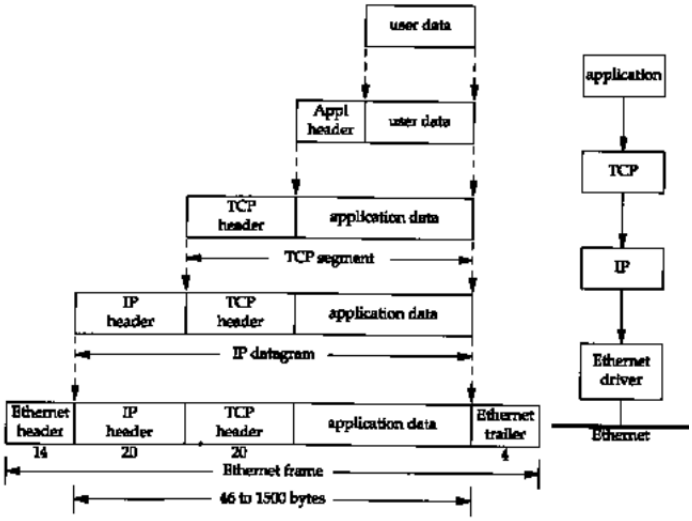
En este capítulo se describen los protocolos involucrados en una implementación de MPI, que utiliza infraestructura Gb/s Ethernet.

2.1. Modelo TCP/IP

Al momento de explicar el funcionamiento de una red, y más allá de la mención recurrente y habitual del modelo de capas OSI, en nuestro caso disponemos de un modelo mucho más específico y preciso: *el modelo de la familia de protocolo TCP/IP*. Este modelo se encuentra mencionado en numerosa bibliografía sobre el tema [STEVENS].

El modelo TCP/IP se presenta en la siguiente figura:

Figura 2-1. Modelo de capas de una red TCP/IP, y encapsulamiento de las distintas capas.



Este modelo se ha definido de manera formal en la RFC 1122 [RFC1222], y consta de 4 capas:

1. *Capa de aplicación*: define cuestiones asociadas a la presentación, codificación y flujo de mensajes de red de la aplicación.

2. *Capa de transporte*: proporciona lo necesario para detección y corrección de errores.
3. *Capa de red*: gestiona las conexiones a través de la red que serán utilizadas por las capas superiores.
4. *Capa de enlace*: proporciona los mecanismos para envío de datos a través del medio físico.

En sucesivos párrafos, se analizará el grado de injerencia de cada una de estas capas en el escenario planteado (Cluster MPI sobre una infraestructura Gb/s Ethernet).

2.1.1. Capa de *aplicación*

La capa de aplicación está provista por la implementación de MPI y sus servicios asociados. Además de los datos enviados entre los nodos para la realización de los cálculos, existe una una sobrecarga propia de la implementación de MPI utilizada.

2.1.2. Capa de *transporte*

Estadísticamente, en el caso de una LAN homogénea y construida con las especificaciones de certificación esperables en una instalación de esta naturaleza, es mucho más improbable que se presenten problemas de red que justifiquen la sobrecarga contemplada en cada segmento de protocolo TCP enviado. Es por ello que seguramente se puede llegar a prescindir del uso de todos los aspectos relativos a detección de errores, proveyendo en las capas superiores en todo caso (capa de aplicación), de mecanismos alternativos de reenvío de datos perdidos. La integridad de datos ya es controlada por la capa física (de enlace de datos).

2.1.3. Capa de *red*

En una LAN, el concepto de ruteo es innecesario, ya que las comunicaciones entre los nodos se valen simplemente del protocolo ARP para intercambiar información. Además, se puede prescindir del direccionamiento de red, en cuanto a que la capa física ya posee esta propiedad.

2.1.4. Capa de *enlace*

La capa de enlace está íntimamente ligada al hardware de comunicaciones empleado en la red. Es allí donde efectivamente se produce el fenómeno real de la comunicación de los datos. Es por ello que esta capa no puede ser evitada o salteada. Este protocolo tiene mayores restricciones respecto a las modificaciones que pueden realizarse en él, respecto a los de las capas superiores. Aún así se podrían aprovechar estas pocas variaciones para intentar prescindir de la sobrecarga introducida por las capas restantes.

2.2. Implementación de MPI en la capa de aplicación

En una instalación típica de las implementaciones de MPI más difundidas, la misma se vale de la aplicación SSH/RSH inicialmente, para la ejecución remota de las tareas en los distintos nodos que conforman el cluster. Una vez que la tarea está ejecutándose en cada nodo, se realizan conexiones cliente/servidor en puertos TCP dinámicamente determinados por la gestión del cluster, prescindiendo del uso de SSH/RSH.

Una vez establecidos los puertos, los datos entre los nodos son comunicados por la implementación de MPI utilizada. La sobrecarga introducida por la implementación de MPI es desconocida, y además, varía entre las distintas implementaciones de MPI más utilizadas. Las operaciones SEND/RECV a nivel de MPI establecen claramente el tipo de dato que se envía y la cantidad de elementos del mismo. Esto implica que el nodo receptor conoce la cantidad de bytes de información que va a recibir del nodo emisor durante la ejecución del par de operaciones SEND/RECV, así que la sobrecarga del protocolo debería ser prácticamente nula.

Es por esto último entonces, que envío de datos en las operaciones SEND/RECV debiera ser lo mínimo indispensable. Cada tarea en ejecución se bloquea al esperar el dato, a sabiendas del tipo de datos que debe esperar, y del tamaño del mismo. Asimismo no debiera existir compresión o codificación de los datos enviados, por cuanto eso requeriría un "gasto" en el uso de la CPU, que mejor debe reservarse para el cálculo paralelo impuesto por el algoritmo del programa en ejecución.

La existencia de esta cabecera además, implica un procesamiento de la misma tanto en el nodo emisor como en el receptor, por lo que si evitamos la cabecera, no solo favorecemos el espacio destinado para la transferencia de datos útiles para los nodos, sino que evitamos el desperdicio de tiempo de procesador en tareas que no son útiles para la resolución del problema original que intenta resolverse mediante el uso de un cluster.

2.3. Ahorro ideal de tráfico

Por lo expuesto en párrafos anteriores, podemos suponer que, en el caso más favorable, podemos prescindir de la cabecera utilizada en la capa IP y TCP. Si asumimos que el tamaño de la trama de la capa física es el más grande posible (1500 bytes), y los valores de las cabeceras a eliminar son el mínimo posible (20 bytes cada una), entonces podemos evitar, en lo que a transferencia de datos se refiere, 40 bytes por cada trama de datos enviada.

Además de esta relación entre los datos útiles y los datos totales transferidos, hay que tener en cuenta también el tiempo de procesamiento que utiliza cada nodo en encapsular la información de la aplicación que debe enviar, y remover este encapsulamiento que produce cada capa del protocolo TCP/IP para recuperar los datos recibidos. Estos tareas involucran múltiples cambios de modo usuario a modo kernel por parte del sistema operativo, para cada paquete recibido [Bhattacharya].

Es evidente que las aplicaciones de red se ven afectadas de manera directa por la forma en que está implementada la pila TCP/IP del sistema operativo utilizado por los nodos. Entre las distintas operaciones que debe realizar el kernel para procesar el datagrama se incluyen: procesamiento de las interrupciones involucradas, sobrecarga introducida por el controlador del dispositivo, copia entre buffers, cálculo de sumas de chequeo para controlar integridad, etc [Bhattacharya].

En tiempos recientes las velocidades empleadas en las redes locales ha crecido con una progresión mayor que la velocidad de los microprocesadores; solo basta recordar que no hace demasiado tiempo las redes Ethernet poseían una velocidad nominal de 10Mbp/s, mientras que en la actualidad podemos acceder a redes con velocidades de hasta 1 Gbp/s ó 10 Gbp/s (2 ó 3 órdenes de magnitud superior). Esto implica que el tiempo empleado para el procesamiento de la sobrecarga introducida por el protocolo de red, ha ganado más peso en proporción, respecto al tiempo empleado en la transmisión de los datos. Ante situaciones de carga alta, el procesamiento de los protocolos de red puede consumir una alta fracción de los recursos de cómputo disponibles [Bhattacharya].

Junto a la sobrecarga introducida por la pila del sistema operativo, debe considerarse además, la introducida por la implementación de MPI que se esté utilizando. Esta sobrecarga comprende, tanto el agregado de información de control que la implementación anexa a los datos verdaderamente útiles y necesarios para el cálculo paralelo, así como también el uso de protocolos y servicios adicionales para realizar las tareas de control y gestión. Esta sobrecarga varía con cada implementación de MPI.

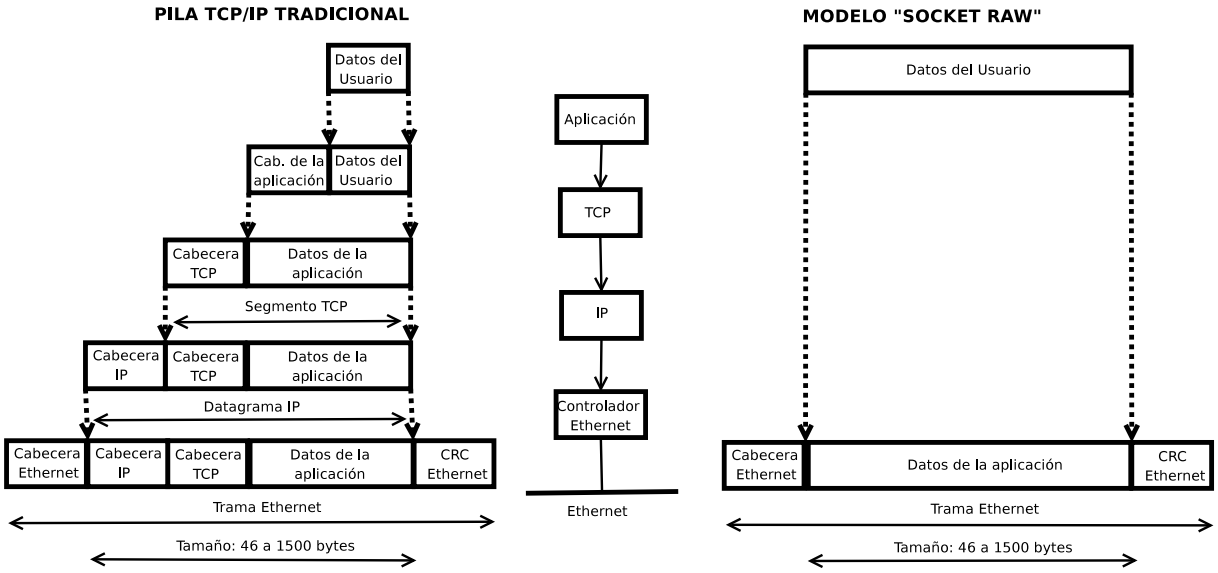
Más allá de analizar si el diseño de la pila empleada es eficiente, o bien su funcionamiento está optimizado para el escenario planteado por el problema a resolver, mucho mejor sería si pudiéramos evitarla por completo.

Por todos los elementos mencionados en párrafos anteriores, se justifica un análisis posterior con mayor nivel de detalle para intentar prescindir de las capas de red intermedias.

2.4. Modelo *Socket RAW*

El modelo *Socket raw* permite que una aplicación envíe y reciba paquetes de red, evitando el encapsulamiento introducido por la pila TCP/IP del sistema operativo. La aplicación se encarga del procesamiento del paquete de manera directa [Socket RAW]. En la siguiente figura se puede apreciar el área de aplicación de este modelo.

Figura 2-2. Area de aplicaci{on del Modelo "Socket Raw", respecto a "Berkley Sockets".



En nuestro modelo propuesto, el cluster MPI evitará el uso de capas intermedias en el envío de la información. La ventaja en el escenario de una red LAN Ethernet justifica esta exploración. De todas formas, a priori se observan algunas restricciones:

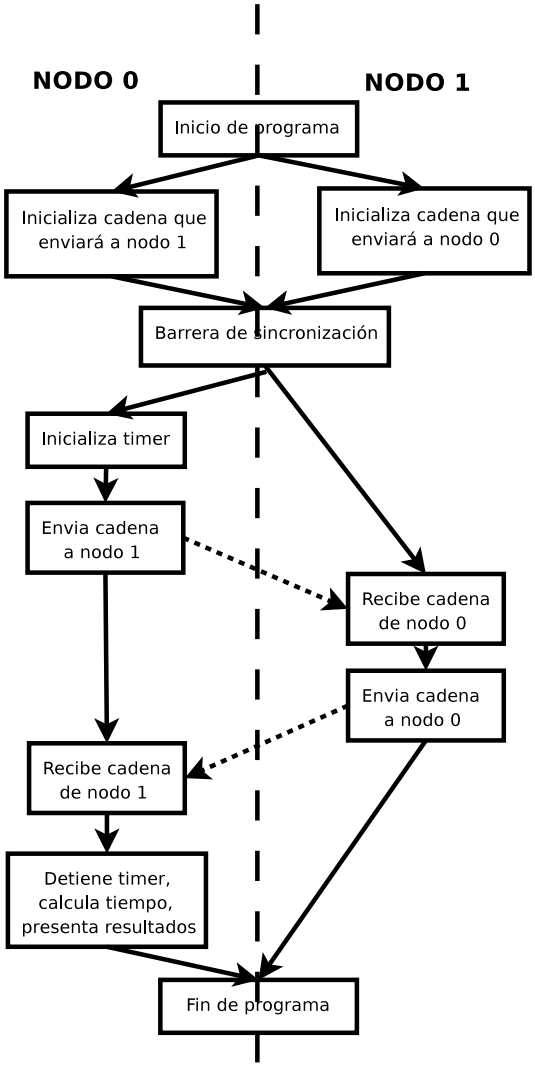
- *Rigidez en la asignación de canales de comunicación*: al carecer del concepto de puertos en el ámbito de la trama Ethernet, se debe disfrazar la asignación dinámica de puerto que realiza MPI. O bien se puede intentar una re-definición de esta funcionalidad mediante el uso del campo "type" en la cabecera Ethernet (existe un rango de uso "experimental"), o bien se agrega rigidez al diseño, de tal forma que la asignación de canales de comunicación entre los nodos del cluster no sea verdaderamente dinámica.
- *Aspectos de seguridad*: el uso de "socket RAW" exige que el código en ejecución goze de los permisos de superusuario en el ámbito del sistema operativo. Esto implicaría que el código a ejecutar, desarrollado en principio por un usuario con acceso limitado al sistema, sea lanzado en los distintos nodos con permiso de superusuario por el gestor de tareas del cluster. Por ello es que se debería implementar algún mecanismo de auditoría del código a ejecutarse.

2.5. Ejemplo de tráfico generado por un programa MPI

A los fines de mostrar el impacto que tienen las distintas capas de red del modelo TCP/IP en las comunicaciones de un cluster (y particularmente en las operaciones "Send" y "Receive" que es lo que nos atañe), se muestra la captura del tráfico de un sencillo programa, que implementa un mecanismo de "ping-pong" entre dos nodos

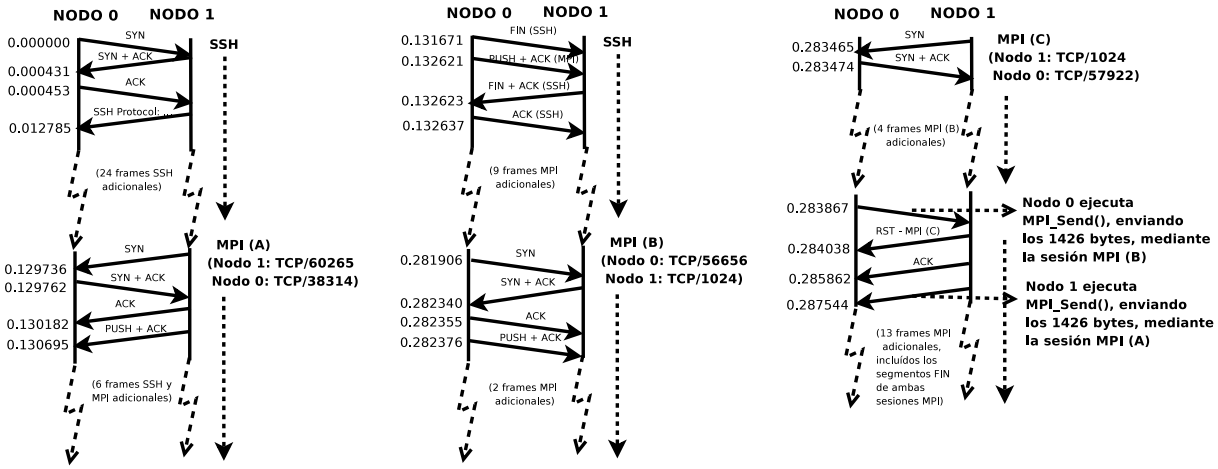
del cluster. Este tipo de mecanismo es la base de todas nuestras pruebas posteriores, para tratar de perfilar las características de la red. El diagrama de bloques del programa se muestra a continuación:

Figura 2-3. Diagrama de bloques del programa que realiza un "ping-pong" entre dos nodos.



Se procedió a ejecutar varias veces este programa de prueba. El tráfico generado se analizó con un capturador de red. La evolución temporal del tráfico capturado se muestra en la siguiente figura:

Figura 2-4. Tramas capturadas durante la ejecución del programa "ping-pong".



Del tráfico capturado, se desprenden las siguientes observaciones:

- Existe una gran cantidad de sobrecarga, producto de servicios asociados a la gestión del cluster. Particularmente, el protocolo SSH es utilizado de manera intensa en una primera etapa, donde el nodo raíz del cluster lanza la aplicación en los nodos esclavos. Una vez que todos los nodos integrantes de esta instancia de ejecución del programa están al tanto del puerto TCP donde los demás nodos esperan la información a nivel de MPI, las sesiones de SSH son finalizadas y el tráfico restante utiliza estos puertos TCP asignados de manera dinámica.

  El tráfico generado por SSH es generado solo al comienzo de la ejecución, por lo que su peso cobra menor relevancia mientras más tiempo demande la ejecución del programa MPI.

- En lo que hace al tráfico generado por MPI, la capa de aplicación utilizada es TCP. La sobrecarga introducida por esta capa es de 32 bytes (la información opcional de la cabecera ocupa 12 bytes en este ejemplo).
- La capa de red es IP y la sobrecarga introducida por esta capa es de 20 bytes.
- En lo que respecta al segmento específico donde se transmite la cadena de prueba empleada en el programa "ping-pong", se observa una sobrecarga introducida por la implementación de MPI utilizada, de 22 bytes.
- La implementación de MPI no codifica ni comprime la información enviada en la operación *MPI_Send()*. Probablemente la razón sea que, por definición, se debe gastar el mínimo tiempo de CPU posible en otras tareas que no sean el cálculo propuesto por el programa en sí, y las tareas de codificación o compresión suelen ser costosas para el procesador.

Asumiendo una ocupación completa del paquete a nivel de la capa física, se puede calcular qué porcentaje del tráfico se corresponde con información útil, y que porcentaje se corresponde a sobrecarga:

```
Tamaño útil = MTU - (Cabecera Aplicación) - (Cabecera Transporte) - (Cabecera Red)

Tamaño útil = 1500 - 22 - 32 - 20

Tamaño útil = 1426
```

Este es el valor máximo de datos que podemos enviar entre nodos, sin que se produzca fragmentación del datagrama (esto es, cuando la trama de la capa física se está utilizando con el tamaño más eficiente -valor de MTU-). Para el envío de dos secuencias de datos de 1426 bytes cada uno, que bien podrían entrar en dos tramas Ethernet, se obtienen los siguientes valores sumarios:

- 76 tramas Ethernet empleadas.
- 14339 bytes de datos transferidos.
- 0.291 segundos entre el primer y el último paquete capturado.

La sobrecarga es notoria.

2.6. Comparación de tiempos entre un programa *MPI* y un programa *SocketRaw*

En el ámbito de los clusters, existen herramientas conocidas para la medición de los parámetros de comunicación, como es el caso de *b_eff* por ejemplo. *b_eff* utiliza distintos mecanismos de comunicación en el cluster para determinar los valores asociados a los parámetros de red (ancho de banda y latencia) de la manera más fidedigna posible. Los métodos aplicados son generales, y pueden utilizarse cualquiera sea la topología o arquitectura de red que esté implementada en el cluster medido. De todas manera, nuestro particular interés es más específico: caracterizar nuestra red Ethernet para tráfico tipo send/recv exclusivamente.

Para determinar esta cuestión, se utilizará un programa específicamente adaptado. El programa se llama "mpitest", y está inspirado en los programas de prueba utilizados por *Andreas Schaufler*, en el contexto de sus mediciones de rendimiento de red en Linux, desarrollados para comparar las ventajas de una comunicación que utiliza *raw socket* vs. *UDP* [Schaufler].

Además, y con la intención de cuantificar la sobrecarga introducida por los distintos elementos mencionados en párrafos anteriores, se compararan los resultados con un programa que implementa un mecanismo de *ping-pong* mediante el uso de "socket raw": esto es, implementando las comunicaciones a nivel de la capa Ethernet específicamente.

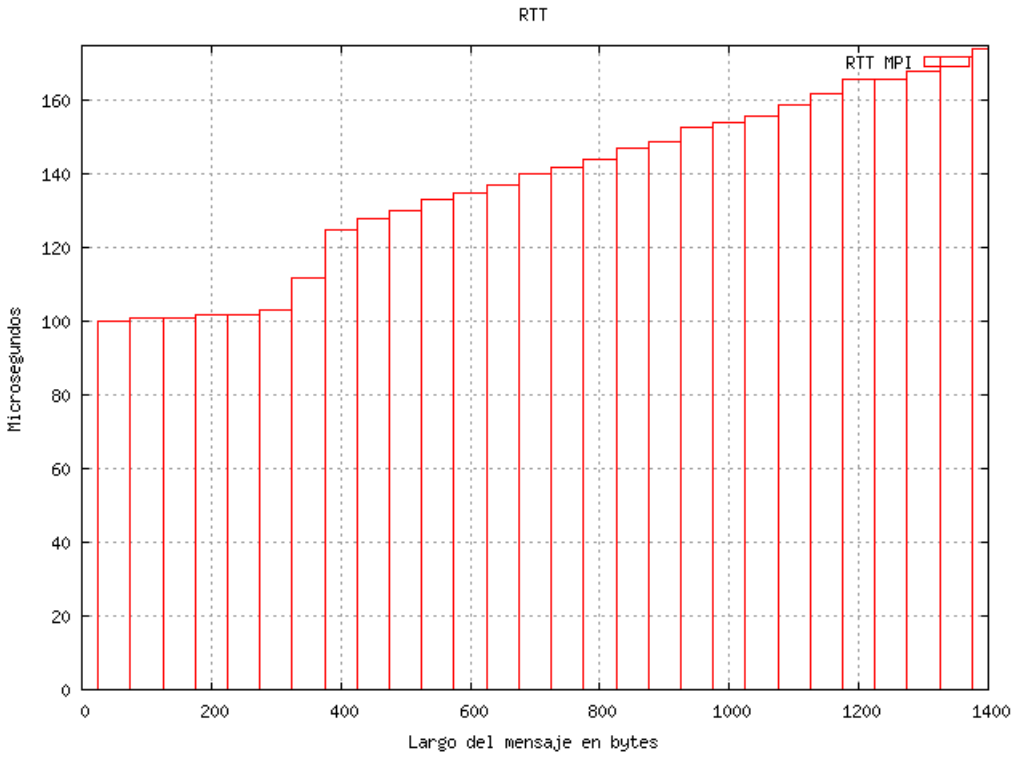
2.6.1. Utilización de mpitest

El programa *mpitest* utiliza un mecanismo de ping-pong, tan difundido en múltiples utilitarios de red. La idea del programa es la siguiente: se consideran dos nodos del cluster para la medición. El nodo de rango 0 genera un mensaje de un tamaño determinado. Envía repetidamente este mensaje al nodo de rango 1. Este último, en cada ocasión que recibe el mensaje, lo devuelve al nodo de rango 0. Cada vez que el nodo de rango 0 recibe la respuesta esperada, calcula el tiempo de ida y vuelta para ese mensaje. Una vez finalizada la repetición para un mensaje dado, el nodo 0 calcula un promedio de tiempo de ida y vuelta para ese mensaje de largo fijo.

El nodo 0 repite esta operación para mensajes de distinto largo, de tal manera que estima el tiempo promedio de ida y vuelta en función del largo del mensaje enviado. Tanto el tamaño del mensaje inicial, como el valor incremental con que aumenta el largo del mensaje, es de 50 bytes. El tamaño máximo considerado para el mensaje, es aquel que no supere el valor de MTU de la arquitectura de red, por cuanto se pretende evitar los efectos introducidos por la fragmentación (1500 bytes en nuestro caso). Las operaciones de envío y recepción utilizadas por ambos nodos, son *MPI_Send* y *MPI_Recv*.

La ejecución del programa en el cluster de prueba, arrojó los siguientes resultados:

Figura 2-5. Resultado de la ejecución del programa "mpitest" en el cluster de prueba.



Los tiempos obtenidos tienen en cuenta exclusivamente lo que tarda la cadena de caracteres en ir desde el nodo 0 al nodo 1, y volver. El cálculo del tiempo empleado se realiza internamente en el programa MPI, y por ende se excluye del cálculo a todo el tiempo utilizado por los servicios empleados en la gestión del cluster (SSH).

2.6.2. Utilización de socket raw

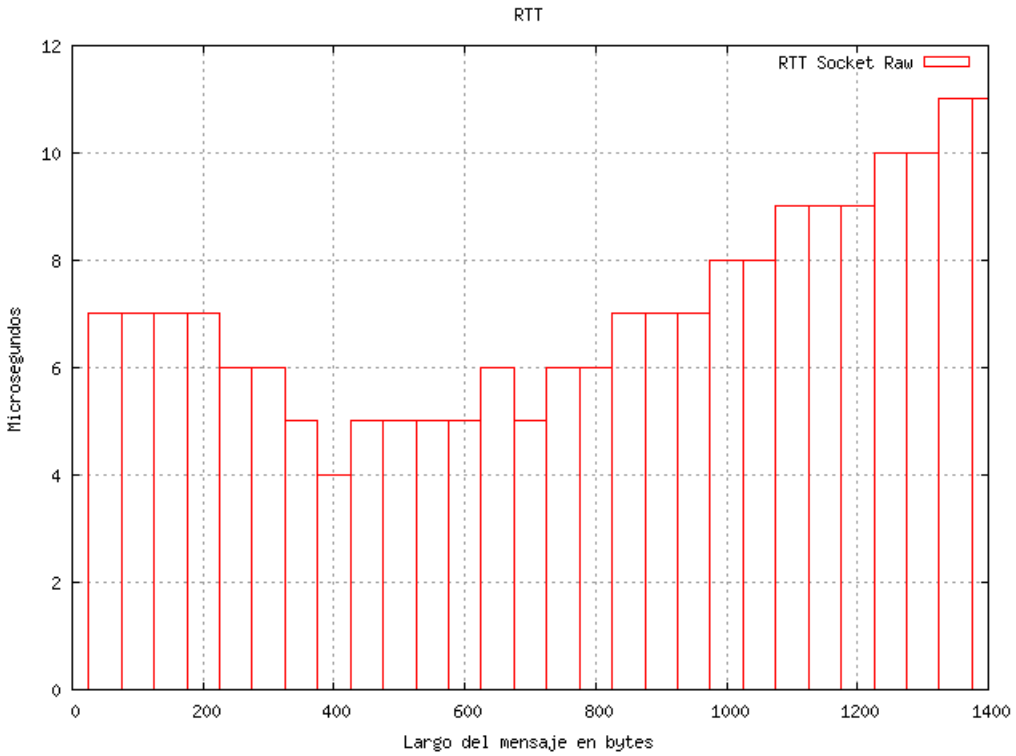
La utilización de *socket raw* implica el uso de 2 programas: un programa servidor y un programa cliente. El programa cliente es el que genera la primer trama con un mensaje constante, enviada desde el nodo 0. El programa servidor, que debe estar en ejecución en el nodo 1, recibe e identifica a esa trama, y envía su trama de respuesta. Nuevamente, cada vez que el nodo de rango 0 recibe la respuesta esperada, calcula el tiempo de ida y vuelta para ese mensaje. Una vez finalizada la repetición para un mensaje dado, el nodo 0 calcula un promedio de tiempo de ida y vuelta para ese mensaje de largo fijo.

La repetición de mensajes y la información mostrada en pantalla respeta el mismo formato que el programa *mpitest*. Como elemento adicional, debió agregarse un bucle de "calentamiento" en el código, para que los valores

de tiempo de los primeros paquetes no se vean demasiado afectados por la inicialización de buffers, paquetes, colas de mensajes, etc.

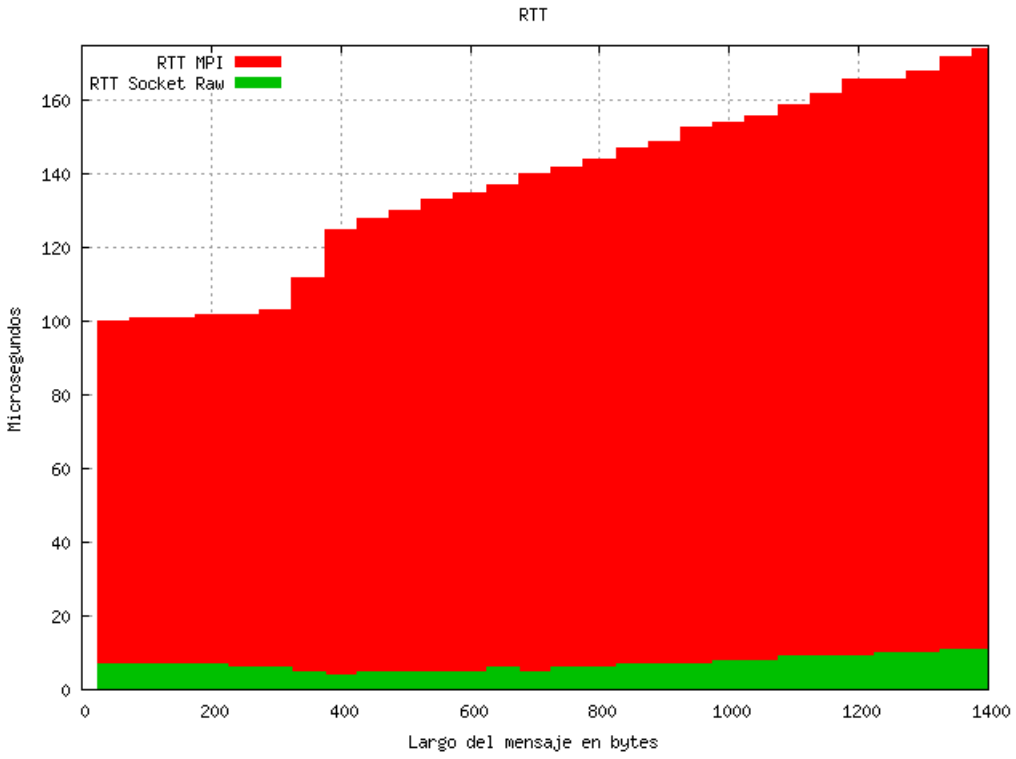
La ejecución del programa en el cluster de prueba, arrojó los siguientes resultados:

Figura 2-6. Resultado de la ejecución del programa "socket raw" en el cluster de prueba.



Comparando los dos gráficos, obtenemos:

Figura 2-7. Comparación de la ejecución del programa "MPI" vs. "socket raw"



La diferencia en los tiempos es notoria.

2.7. Conclusiones

Un vistazo rápido a los gráficos obtenidos en el punto anterior, permite apreciar el efecto de la sobrecarga introducida por la pila de TCP/IP en el uso de las comunicaciones. Para el caso de un mensaje de 1400 bytes por ejemplo, el envío de los datos mediante el uso de la técnica de *socket raw* solo emplea 11 microsegundos, mientras que el uso de MPI requiere de 174 microsegundos, o sea, aproximadamente unas 15 veces más.

Esta significativa diferencia justifica una investigación más profunda del código de una implementación MPI, con el objeto de mejorar esos 174 microsegundos y acercarnos lo más posible al valor obtenido con *socket raw*

Bibliografía de referencia

El siguiente material a sido consultado para el desarrollo de este proyecto:

[Top500] *Top500: Interconnect share for 06/2010*, stats (http://www.top500.org/stats/list/35/conn), charts (http://www.top500.org/charts/list/35/conn).

[Hockney] *The communication challenge for MPP: Intel Paragon and Meiko CS-2*, R. Hockney, 1994, Journal - Parallel Computing - Volume 20 Issue 3, March 1994.

[HPCMMM] *High Performance Computing: Models, Methods and Means*, CSC 7600 (http://www.cct.lsu.edu/csc7600/Home.html), Prof. Thomas Sterling, 2010, Department of Computer Science, Louisiana State University.

[Stevens] *TCP/IP Illustrated Vol. 1*, W. Richard Stevens, 1994, 0201633469, Addison-Wesley Professional.

[RFC1222] *Requirements for Internet Hosts -- Communication Layers*, RFC 1222 (http://tools.ietf.org/html/rfc1122).

[Opteron] *AMD Opteron Specifications and Reference Guide*, AMD.

[Schaufler] *Linux Network Performance. RAW ethernet vs. UDP*, http://aschauf.landshut.org/fh/linux/udp_vs_raw/index.html, Andreas Schaufler.

[Bhattacharya] *A Measurement Study of the Linux TCP/IP Stack Performance and Scalability on SMP Systems*, pdf disponible (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.6392&rep=rep1&type=pdf), Shourya P. Bhattacharya y Varsha Apte.

[Socket RAW] *Raw socket*, http://en.wikipedia.org/wiki/Raw_socket.

Advanced Programming in the UNIX Environment, W. Richard Stevens, 1992, 0201563177, Addison-Wesley Professional.

Programming Perl, Larry Wall, Tom Christiansen, y Jon Orwant, 1992, 0-596-00027-8, O'Reilly Media, Inc..

The C Programming Language, Brian Kernighan y Dennis Ritchie, 1988, 0-13-110362-8, Prentice Hall.

Linux Socket Programming by Example, Warren Gay, 2000, 0789722410, Que Corporation.