Algoritmo Distribuido de Compresión de Datos. Una Experiencia Comparativa con Sockets y PVM

Lic. C. Russo[1], Lic. H. Ramón[2], A.C. A. Anderson[3],A.C. D. Dirazar[3], Ing.A. De Giusti[4]

Laboratorio de Investigación y Desarrollo en Informática[5]
Departamento de Informática - Facultad de Ciencias Exactas
Universidad Nacional de La Plata

Resumen

Es notoria la importancia actual de la compresión de datos en la transmisión de información en redes, especialmente información multimedial.

Aspectos como optimización del uso del canal, tiempos de respuesta, seguridad y reducción de tráfico, justifican las tareas de investigación y desarrollo en este tema, así como la evolución de los recursos tecnológicos disponibles para implementar nuevas técnicas.

En este contexto, se presentan resultados experimentales obtenidos de un algoritmo de compresión de datos utilizando procesamiento distribuido a través de sockets y PVM, teniendo como último objetivo mejorar los modelos de compresión de imágenes incluidos en Khoros.

Por último, se discuten las posibles implementaciones de estos algoritmos distribuidos sobre una verdadera arquitectura de procesamiento paralelo.

---

[1]Prof. Adjunto con Ded. Excl. LIDI. Dpto. de Informática, Facultad de Ciencias Exactas, UNLP.
E-mail crusso@ada.info.unlp.edu.ar
[2]JTP Ded. Excl. LIDI. Dpto. de Informática, Facultad de Ciencias Exactas, UNLP.
E-mail hramon@ada.info.unlp.edu.ar
[3]Analista de Computación. Alumno de Lic. en Informática, Facultad de Ciencias Exactas, UNLP.
E-mail dirazar@ada.info.unlp.edu.ar
[4]Inv. Principal CONICET. Profesor Tit. Ded. Excl., Dpto. de Informática, Facultad de Cs. Exactas, UNLP.
E-mail degiusti@ada.info.unlp.edu.ar
[5]Calle 50 y 115 Primer Piso, (1900) La Plata, Argentina, Teléfono 54-21-227707
E-mail lidi@ada.info.unlp.edu.ar

Introducción

Este es un complemento del estudio de paralelización de un algoritmo de compresión estático especificado en [Ande96].

Compresión

No solo nos preocupa el almacenamiento y el mantenimiento de los datos, sino también la disponibilidad en forma segura y rápida. El problema no acaba con optimizar el manejo local de los datos, debe resolverse también el problema de transmitirlos.

Es por eso que debemos poner énfasis en analizar la conveniencia de minimizar el tráfico de datos por la red, problema especialmente importante cuando el medio de comunicación es costoso (en tiempo y/o dinero).

Las técnicas de compresión intentan reducir la cantidad de bits requeridos para almacenar o transmitir la información, sin pérdida o con pérdida, dependiendo del tipo de dato que se esta comprimiendo (voz, imágenes, textos, etc.).

Básicamente se buscan distintos métodos de compresión de información para [Russ95]:

- Menor requerimiento de memoria para almacenamiento ó
- Reducción de tiempos en la transmisión de datos

En la actualidad se han desarrollado distintos métodos de compresión con las siguientes propiedades [Bora94]:

- Tipo de método: con pérdida o sin pérdida de información
- Nivel de complejidad desde el punto de vista computacional
- Rango de datos sobre el cual el método puede ser utilizado obteniéndose resultados satisfactorios.

Se presenta el algoritmo de Huffman estático con el objeto de comparar las implementaciones sobre sockets y PVM.

Sockets

Procesos en diferentes máquinas pueden comunicarse, pero los métodos a través de los cuales ellos establecen las comunicaciones suelen diferir dependiendo de los protocolos y el medio utilizados. Más aún los métodos pueden no permitir a los procesos comunicarse con otros procesos en la misma máquina. Para proveer métodos comunes para la comunicación interprocesos y para permitir el uso de protocolos sofisticados de red el sistema BSD introdujo un mecanismo conocido como **sockets**.

PVM

Es un sistema de software que permite utilizar una colección de computadoras heterogéneas como un único recurso computacional concurrente. En otras palabras, permite que un conjunto de procesadores heterogéneos conectados a través de una o varias redes sean vistos como una única máquina virtual.

Las computadoras individuales pueden ser multiprocesadores, con memoria compartida o local, supercomputadoras vectoriales, máquinas gráficas especializadas,

estaciones de trabajos escalares, que pueden estar interconectadas con ethernet, FDDI, etc.

Los programas de usuario escritos en C, C++ o FORTRAN acceden al PVM a través de rutinas de librería. La comunicación y el control de procesos son provistas a través de procesos *daemons* en cada host que pertenecen a la red PVM.

Una de las diferencias más importantes con la implementación dada en [Ande96] es que los servers son daemons, mientras que en esta (utilizando PVM) cada server es disparado por su cliente y termina con el.

Análisis del Problema

Codificación Huffman

La codificación de Huffman es probablemente el método más conocido de compresión de datos. La codificación de Huffman tiene aplicaciones prácticas: por ejemplo se utiliza en la última fase de compresión del método JPEG, en el método de compresión MNP-5 utilizado en los modems, etc.

Huffman trabaja de manera muy similar al código Morse, en la mayoría de los textos o imágenes, algunos símbolos y/o letras son más probables que otros. Esta observación sugiere un esquema de codificación en el cual a los símbolos más comunes se les asignen códigos cortos y a los símbolos menos frecuentes códigos largos.

La codificación de Huffman formaliza la idea de la longitud relativa de un símbolo de acuerdo a la probabilidad de ocurrencia del símbolo. La codificación estática requiere de una tabla con las probabilidades antes de comenzar la compresión de datos, esta puede ser generada de acuerdo a la fuente de entrada o contar con una fuente existente.

Algoritmo

El método abstracto de paralelización es el siguiente: Un proceso padre setea la cantidad de tareas (de acuerdo al tamaño del archivo y el tamaño de los **slots** {unidad de E/S física}), luego dispara M procesos hijos(M<=cantidad de tareas) que se encargan de realizar c/u de las tareas.

```
PROCESO_HIJO[1..M]::
 fin = FALSE;
 do {
  P(trabajos_pendientes);
  if ( !Hay_trabajos_pendientes )
        fin = TRUE;
  else
    Decrementar_trabajos_pendientes;
  V(trabajos_pendientes);
   if (!fin)
      procesar_slot();
 } while (!fin);
```

Se divide el archivo a comprimir en slots de igual tamaño. M procesos comprimen el archivo en forma concurrente. Las acciones que realiza cada uno de estos procesos son las siguientes:

- Tomar un slot.
- Generar el histograma del slot.
- Generar el código de Huffman para ese histograma.
- Codificar el slot.

Se crean M subprocesos que se comunican con servidores que realizan el procesamiento (generalmente en otros procesadores). La diferencia entre ambas implementaciones radica en que F+S (Fork+Sockets) asume que tiene los procesos servidores corriendo en todos los procesadores de la red mientras que en la solución PVM, la primera acción de cada subproceso es disparar su propio proceso servidor; luego el mecanismo de comunicación es el mismo para ambas implementaciones. Otra diferencia a destacar es que en F+S la vida de los procesos servidores es independiente del proceso principal, mientras que en PVM los procesos servidores terminan a medida que su cliente (el proceso que lo creó) termina.

La estructura del archivo comprimido es:

Tamaño del Archivo Original
Tamaño del Slot
Slots Comprimidos

y la de los Slots:

Offset del slot descomprimido en el archivo original
Tamaño en bytes del slot a distribuir
Slot a distribuir (sub-slot) o datos-excepción

Si ***Tamaño en bytes del slot a distribuir*** es cero los campos son:

Repeticiones del carácter
Carácter

Si ***Tamaño en bytes del slot a distribuir*** es distinto de cero la estructura del slot a distribuir es:

Tamaño del árbol
Árbol (inorder del código)
Tamaño de los datos comprimidos
Datos comprimidos

Las aplicaciones **servers** son las encargadas de atender los requerimientos de la aplicación principal. Existen 2 tipos de servicios:

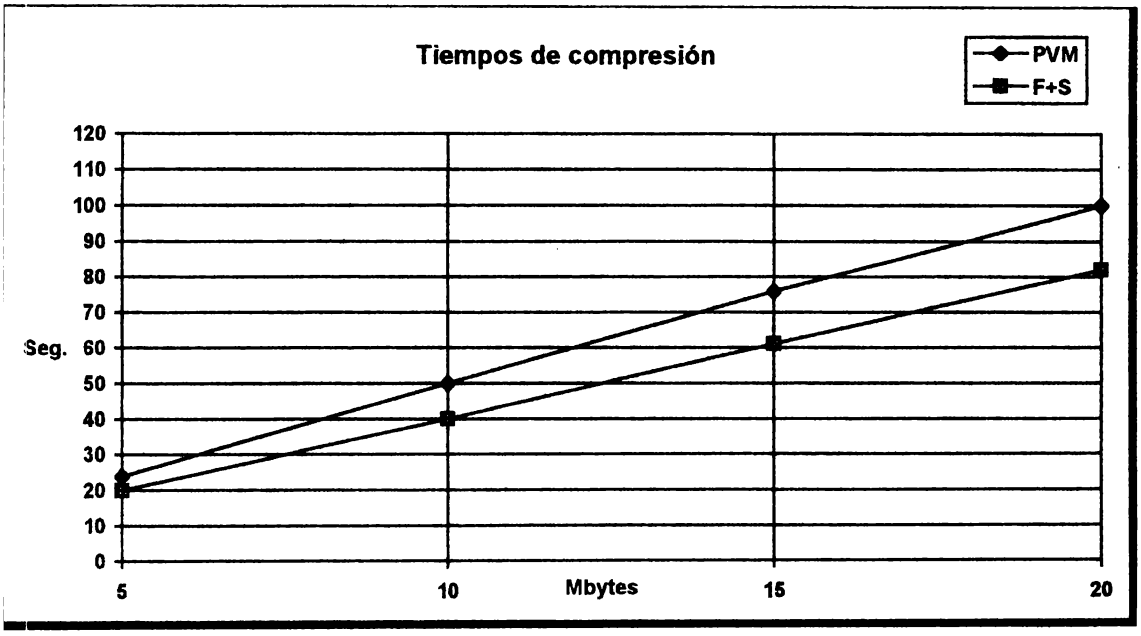
- Comprimir
- Descomprimir

Resultados Obtenidos

Los resultados experimentales del algoritmo se obtuvieron utilizando las mismas computadoras con similar carga de trabajo y obviamente con las mismas entradas, estas PC's tienen todas la misma arquitectura (Intel).

SOCKETS				PVM			
Tamaños (bytes)		Tiempos (seg.)		Tamaños (bytes)		Tiempos (seg.)	
Original	*Comp.*	*Comp.*	*Descom.*	*Original*	*Comp.*	*Comp.*	*Descom.*
20971520	16014427	82	68	20971520	16014427	100	89
15728640	11981695	61	52	15728640	11981695	76	68
10485760	7873631	40	34	10485760	7873631	50	43
5242880	3914234	20	17	5242880	3914234	24	22

El gráfico siguiente nos muestra los tiempos de compresión y se ve claramente overhead que existe en la solución con PVM.



Líneas de Trabajo

Para completar este trabajo nos falta integrar, a la red PVM, máquinas con dis arquitecturas, como SUN, HP9000, Indy, Macintosh, etc., dado que PVM provee la necesarias para la conversión de las diferentes representaciones de cada arquitectur

Dado que en nuestro Laboratorio se está realizando investigación en relac reconocimiento de patrones y tratamiento de imágenes, sería conveniente, integrar

ambiente Khoros algoritmos de compresión de imágenes con wavelets utilizando PVM para paralelizarlo.

Además estamos estudiando la aplicación de Message-Passing Interface [MPI] para comparar con el estándar y PVM.

Conclusiones

Con sockets obtenemos mejor performance en el tiempo de compresión debido a que no existe overhead, dado que las comunicaciones y asignaciones de procesos se hacen en forma manual.

Con PVM tenemos mejor abstracción en relación a las comunicaciones entre los diferentes procesos y en donde estos se están ejecutando, pero estas mejoras producen sobrecarga entre la aplicación y el PVM, pero con esta herramienta podemos concentrarnos en el algoritmo a desarrollar, sin preocuparnos por las comunicaciones entre procesos.

En el Laboratorio (LIDI) se encuentran los fuentes con los cuales se obtuvieron los resultados expuestos en este trabajo, están desarrollados en C bajo LINUX utilizando sockets, semáforos y PVM.

Bibliografía

[Ande96] *Paralelización de un Algoritmo de Compresión que Utiliza Diccionario Estático.* Russo, Ramón, Anderson, Dirazar, enviado a 2da CACIC.

[Andr91] *Concurrent Programming.* Gregory R. Andrews. The Benjamin Cummings Publishing Company Inc.

[Bora94] *Compresión de Imágenes en Dos Tonos.* Boraccia Marcos, Mas Carlos, Rodríguez Leandro. Trabajo de Grado, UNLP, 1994.

[Burn93] *Concurrent Programming.* Alan Burns, Geoff Davies. Addison Wesley.

[Chan88] *Parallel Program Design.* K. Mani Chandy, Jayadev Misra. Addison Wesley.

[Code92] *Introduction to Parallel Processing.* Bruno Codenotti, Mauro Leoncini. Addison Wesley.

[Geha89] *Concurrent Programming.* Narain Gehani, Andrews D. McGettrick. Addison Wesley.

[MPI] *MPI: The Complete Reference.* M. Snir, S. Otto, S. Huss-Lederman, D. Walker and J. Dongorra. The MIT Press, Cambridge, Massachusetts. London, England.

[Naio95] *Cells counting algorithms using Khoros.* Marcelo Naiouf, Carla Galimberti, Cecilia V. Sanz, Second International Congress of Information Engeneering, Bs. As., 1995.

[Nels91] *The Data Compresión Book.* Mark Nelson, Prentice Hall, 1991.

[Russ90] *Combinación de Algoritmos de Criptografiado y Compresión de Datos en Redes de Procesadores.* Claudia Russo, Gabriela Rosanova, Armando De Giusti, 2do Congreso de Informática y Telecomunicaciones de la Provincia de Buenos Aires, CINTEBA 90, 1990.

[Russ95] *Paralelización de Algoritmos de Compresión Fractal de Imágenes.* Claudia Russo, Hugo Ramón, Marcos Boraccia, Second International Congress of Information Engeneering, Bs. As., 1995.

[Umar93] *Distributed Computing and Client-Server Systems.* Amjad Umar, Prentice Hall, 1993.