Aplicación de PVM a la carga de datos para el análisis OLAP

- DRAFT -

Ignacio Larrañaga, malito:ilarra@fing.edu.uy, http://www.fing.edu.uy/~ilarra
Grupo de Concepción de Sistemas de Información, http://www.fing.edu.uy/~csi
Instituto de Computación – Facultad de Ingeniería – Universidad de la Republica

Abstract

En este documento se presenta un trabajo realizado con la herramienta de programación paralela PVM sobre el área de procesos de extracción transformación y carga (ETL). Se muestra una herramienta con la cual se permite modelar grafos basados en la idea de "*flujo de datos*" para la resolución de problemas en el área de ETL y ejecutarlos en forma centralizada o distribuida. Se presenta también un problema de ejemplo en el área de generación de redundancia para el análisis OLAP y su resolución con la herramienta, además del análisis de tiempos y speed-up del proceso.

Introducción

El objetivo general de este trabajo es formar experiencia en el trabajo con la herramienta PVM [PVM94] para resolver procesos que actualmente se resuelven con herramientas de Extraction-Tranformation-Loading (ETL). Las herramientas ETL son programas de software que realizan la extracción de datos de varias fuentes, su limpieza, adecuación e inserción dentro de un data warehouse. Los procesos de ETL pueden ser modelados utilizando grafos. En este trabajo, se presenta una herramienta que recibe grafos de estas características y permite ejecutarlos de forma distribuida utilizando PVM.

La herramienta implementada se aplicará en un ejemplo de generación de redundancia masiva en un data warehouse relacional para análisis OLAP. El esquema del data warehouse es el obtenido por sucesivas aplicaciones de la primitiva "**P9 Aggregate Generation**" presentada en [AM2000]. Para esto se modelará un grafo que representará una forma de generar la redundancia, esto claramente puede ser pensado como un proceso de ETL. El hecho de ser un problema de un alto potencial de crecimiento en por ejemplo dos líneas: cantidad de datos para procesar y dificultad inherente por la cantidad de dimensiones y jerarquías involucradas es lo que lo hace interesante como ejemplo.

En lo que sigue primero se presentaran las áreas de trabajo relacionadas, luego se pasara directamente a la herramienta implementada deteniéndose en la implementación distribuida (en la sección "Implementación Distribuida") y en la implementación de la primitiva P9 (en la sección "Implementación del AGREGATE OPERATOR (P9)"). A continuación se presentara el ejemplo y los resultados obtenidos, se comentaran las posibles líneas de trabajo futuro y se finalizara con las conclusiones.
Se incluyen tres apéndices uno con la implementación del ejemplo en una de sus variantes expuesto en forma completa y otro con los comandos disponibles en la herramienta. El ultimo apéndice por otro lado mostrara las tablas de los resultados obtenidos.

Trabajos Relacionados

La implementación de la herramienta propuesta se realizo en base a una modificación al trabajo presentado en [BBGS] y [MG2000] para adaptarla al contexto y alcance del presente trabajo. Sobre la temática presentada en estos dos artículos se realizo un análisis previo al comienzo del proyecto que se presenta en [IL2001].

A modo de resumen [BBGS] y [MG2000] podemos decir que lo que ahí se presenta es un prototipo para procesos paralelos de ETL, basado en el encare de *flujo de datos* (el cual señala ha sido tomado por la comunidad de base de datos), montado sobre un cluster de computadores.

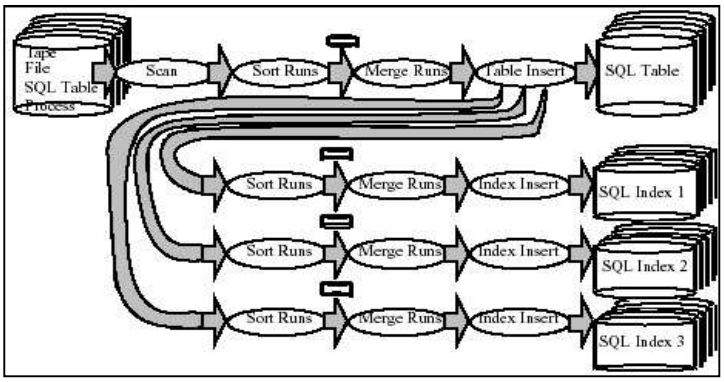

Figura 1: Ejemplo de grafo de definición.

En el encare basado en el *flujo de datos*, la información reside en archivos o bases de datos en algunos discos, cintas u otros dispositivos de memoria. Los algoritmos buscan subconjuntos de esta información, y o bien depositan sus resultados en dispositivos de almacenamiento o retornan sus respuestas a un array de programas de aplicación. Un algoritmo de flujo de datos se describe como un grafo dirigido.
Los nodos del grafo, llamados operadores, son programas secuenciales. Cada operador lee sus *"streams"* de entrada (*"dataflows"*), transforma la información, y produce uno o mas *"streams"* secuenciales de salidas de datos. Las aristas del grafo muestran los flujos de datos entre operadores. En la Figura 1 se presenta un ejemplo de un proceso modelado con esta técnica.

Sobre la herramienta PVM, lo que podemos comentar es que se trata de una solución para atacar problemas de programación paralela. Es una framework es decir, da soporte para resolver este tipo de problemas, no los resuelve[(1)].
PVM compuesto por una "*maquina virtual*" (MV), que para expresarlo en pocas palabras es una herramienta de control distribuido de procesos, similar a lo que seria una "scheduler" del sistema operativo. Y por otro lado las bibliotecas de programación necesarias para interactuar con la MV y los demás procesos que nos permiten comunicarlos dentro de la arquitectura propuesta.
Esta MV se conforma por procesos gemelos que corren en todos los nodos de la red que a su vez la definen. Brinda servicios a los procesos que trabajan bajo su ambiente y se ejecuta en todos los nodos (maquinas) que la conforman, de ahí su nombre.

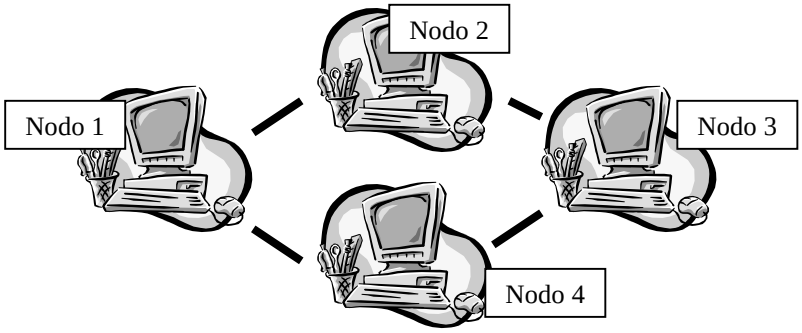

Figura 2: Nodos en una maquina virtual PVM.

Para el control de esta infraestructura se tiene una consola desde la cual se pueden monitorear los procesos que ejecutan dentro de la maquina virtual, así como alterar la configuración, por ejemplo agregando o removiendo una maquina, etc.

[1] Por mas información sobre PVM consultar [PVM94] y [WPVM].

La herramienta

La herramienta que se implemento ataca tres problemas, que son, la definición del proceso de extracción, transformación y carga (ETL)[2], la ejecución centralizada y la ejecución distribuida.

Para enfrentar el primer problema se da acceso a una serie de comandos que permiten definir los distintos elementos del proceso. Esto según se extrae del documento serian, la definición del grafo del proceso con los tipos de operaciones que cada nodo ejecuta y los formatos de los registros intercambiados en cada arista[3].

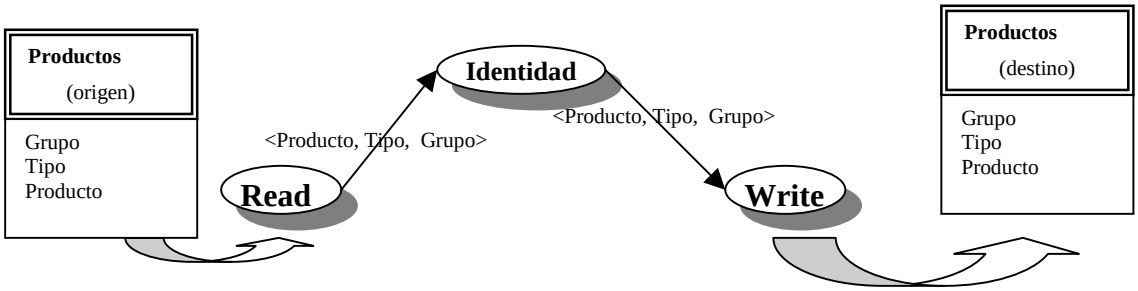

Figura 3: Ejemplo de proceso.

Supongamos el siguiente ejemplo para fijar ideas. Sea un conjunto de productos identificados por un código teniendo a su vez un tipo de producto asociado y un grupo.
La tabla relacional origen de los datos seria entonces de tuplas formadas por <Producto, Tipo, Grupo>. Supongamos a su vez la transformación mas simple posible, la identidad. Y finalmente dado que estamos aplicando la identidad la tabla relacional destino será del mismo formato que la tabla origen. Gráficamente esto se ve en la Figura 3.

Para darle la información a la herramienta disponemos entre otros de los siguiente comandos:

- "***define record format***". Que nos permite definir los formatos de registro que se usaran para intercambiar la información entre los distintos nodos del sistema (Producto, Tipo, Grupo), así como leer y almacenar la información de archivos. Estos formatos son dados por un nombre y un conjunto de triplas formadas por "*nombre*", "*tamaño*" y "*tipo*" de cada atributo (p.e. "Formato1" = {<Producto, 10, String>, <Tipo, 4, Entero>, <Grupo, 4, String>}).

  En sintaxis de la herramienta[4, 5]:
  define record format Formato1 { Producto, 10, String; Tipo, 4, Integer; Grupo, 4, String};

- "***define nodes***". Que nos permitirá definir los nodos que habrá involucrados en el grafo que define el proceso (Read, Identidad y Write). Estos serán dados por conjuntos de duplas "*nombre*" y "*operador*" (p.e. {<"Read", Read>, <"Identidad", Identidad>, <"Write", Write>}). Los operadores son los que nos permitirán definir la operación a realizar en cada nodo (p.e. el nodo de nombre "Identidad" realiza la operación de identidad, es decir copia toda su entrada en la salida).

  En sintaxis de la herramienta[5]:
  define nodes {
  NodoRead, read operator (W:\cprojects\data\ventas-origen.txt);
  NodoIdentity, identity operator;
  NodoWrite, write operator (W:\cprojects\data\ventas-destino.txt)
  };

[2] Obviamente aquí se esta abusando de la palabra ETL ya que lo que se encuentra realizado simplemente ataca un sector acotado y reducido de los problemas en el área.
[3] Estamos tomando una solución simplificada de la propuesta original.
[4] Hemos de notar que aunque se hablo del grafo primero y de la definición de formatos posteriormente, esto en la practica se realiza en forma inversa ya que como veremos las definiciones de formato son necesarias para la definición de las aristas.
[5] Por mas información ver en el "Apéndice – Comandos de la Herramienta" el comando **define**.

Los operadores que tenemos disponibles para trabajar y por lo tanto la funcionalidad que podremos colocar en cada nodo es:

- **READ OPERATOR**. Lo que nos permitirá es leer la información de un archivo dado y colocarla en las aristas de salida para que otros nodos la utilicen. Como se ve recibe como parámetro el nombre del archivo del cual leerá los datos.

- **WRITE OPERATOR**. Realiza la operación inversa a la anterior, toma su entrada de las aristas que le llegan y la vuelca en el archivo que se le indica. También recibe como parámetro el nombre del archivo del cual grabara los datos.
  Aunque uno intuitivamente pensaría que tanto en este como en el operador anterior es necesario recibir el formato de los datos que se leerán o escribirán por una decisión de diseño este se da con las aristas que salen del nodo, esto simplemente porque parece mas flexible que el otro enfoque.

- **IDENTITY OPERATOR**. Copia la información de todas las aristas de entrada en las aristas de salida y obviamente no recibe parámetros.

- **AGGREGATE OPERATOR**. Toma todas sus entradas, y realiza la agregación de estos colocándolos luego en todas sus salidas. La inspiración de este operador es claramente la primitiva "**P9 Aggregate Generation**" presentada en [AM2000] y recomendamos referirse a esta referencia por mas información. De igual forma a continuación nos extenderemos un poco mas sobre la resolución en particular.

- "***define edges***". Que definirá las aristas que unen estos nodos para formar el grafo. Estas aristas están dadas por un conjunto de cuádruplas "*nombre*", "*nodo origen*", "*nodo destino*" y "*formato*" (p.e. {<"AristaRead–Identidad", Read, Identidad, "Formato 1">...}).

  En sintaxis de la herramienta[(5)]:
  define edges {
  AristaRead-Identidad, NodeRead, NodeIdentity, Formato1;
  AristaIdentidad-Write, NodeIdentity, NodeWrite, Formato1
  };

Con estos comandos y algunos detalles mas especificamos el proceso que queremos realizar[(6)], pasemos ahora a como lo ejecutamos. Aquí como ya mencionamos tenemos dos posibilidades: ejecución centralizada y ejecución distribuida.
Que queremos decir con ejecución centralizada?. Bueno nos referimos a que un solo proceso del sistema operativo realizara toda la tarea y obviamente en una sola maquina. Esto fuera de detalles de implementación no es mas que un algoritmo que hace una recorrida del grafo y va calculando los resultados.
Dada la implementación actual la única restricción que impondremos sobre el grafo (en el caso centralizado) es que este sea un árbol (o bosque), dado que la recorrida se realiza con invocaciones recursivas y estas no están preparadas para tomar entrada de varias fuentes. Por las dudas remarcamos que en el contexto distribuido esto no es así.

Para llevar a cabo la ejecución centralizada basta simplemente que una vez definido lo anterior ejecutemos el comando "**execute alone**".

Implementación Distribuida

Bueno y ahora el punto mas interesante, que es lo que sucede con la ejecución distribuida?. Aquí entraremos ahora si en un poco mas de detalle sobre como se implemento.
Primero que nada lo que ya indicamos sobre la definición del proceso es exactamente igual y a los efectos de la persona que lo ejecuta solo basta con utilizar ahora el comando "**execute**" en vez del otro.

La diferencia es que ahora en vez de utilizarse el proceso actual para realizar el procesamiento, se levantaran N procesos (N es la cantidad de nodos del sistema) representando los nodos del sistema. Luego de estar estos en ejecución mediante las facilidades de PVM se le enviaran los datos sobre la estructura

[6] Ver en el apéndice un ejemplo completo.

del grafo que les interesa (básicamente el sub-grafo que lo tiene a el y a los nodos que se encuentran a un salto de distancia, además de las aristas de salto). Desde ese momento en adelante cada nodo simplemente empezara a ejecutar el operador que se le halla indicado.

Debemos agregar un aspecto interesante, que es que los nodos no tienen porque enviarse a cualquier maquina o arquitectura de la red, sino que mediante las funcionalidades de PVM esto puede controlarse. La herramienta hace uso de este tipo de funcionalidades para ofrecer cosas como que en la definición de los nodos podemos utilizar una cláusula "on machine {nombre de maquina}" u "on architecture {nombre de la arquitectura".

Continuando con el tema de operadores, si el operador que se indico es un "**READ OPERATOR**" se abrirá el archivo correspondiente y como se explicara a continuación se enviaran los datos a los nodos consecutivos. Si el operador es "**WRITE OPERATOR**" lo que se valla recibiendo se ira colocando en el archivo que se indico en la definición del operador. El caso de "**IDENTITY OPERATOR**" es si se quiere aun mas sencillo, simplemente cuando algo llega se reenvía.
El caso un poco mas complejo es el de "**AGGREGATE OPERATOR**". En este caso todo lo que se recibe, es colocado en un vector donde si se constata que el elemento ya existía (mediante igualdad en los campos fijos) se le aplica la operación de agregación que se dio en la definición del operador (esto lo veremos en "Implementación del AGREGATE OPERATOR (P9)") a los campos que corresponda. Luego de haber terminado de realizar toda la agregación recién se comienza entonces a propagar los datos calculados.

El aspecto que nos resta es la comunicación entre procesos, para esto se implemento la idea de RIVER[7], esto explicado en pocas palabras es un ambiente de comunicación en el cual unos procesos colocan datos utilizando lo que se define como un "*sink"* y sacando datos de lo que se llama un "*source*".
Cada operador recibe internamente para trabajar todos los "*sources*" de donde obtener datos y los "*sinks*" donde volcarlos. Obviamente hay excepciones, el operador de lectura no recibe nada por lo que no se les pasan los "*sources*", así mismo el de grabación no recibe los "*sinks*".

Los "*sources*" y los "*sinks*" en la implementación en definitiva no son mas que puntos de acceso a las rutinas de PVM, enmascaradas con estos nombres. Cuando se coloca en un "*sink*" un registro simplemente se empaquetan con PVM todos los campos del registro y se le envían al proceso receptor, pero este sin embargo los ve como saliendo de un "*source*" que internamente realiza la operación inversa, es decir recibe y des-empaqueta.

La implementación también para ser un poco mas viable agrega el hecho de que arma mensajes mas grandes que un registro al momento de enviar, y en el caso de un nodo que envía el mismo dato a múltiples nodos utiliza multicast en lugar de envíos simples.

Implementación del AGREGATE OPERATOR (P9)

Este es el operador que levanta el valor de la aplicación, no porque su implementación sea novedosa o distinta sino porque abandona la línea de lo poco practico para entrar en el área de resolución de problemas reales como mostraremos en el ejemplo.

Para refrescar lo que hace esta primitiva mencionaremos que básicamente dada una relación de medida[8], esta genera otra, en la cual la información es resumida (o agrupada) por un conjunto dado de atributos.
La entrada de esta primitiva según se define en [AM2000] es:

- source schema : $R(A_1, ..., A_n) \in Rel_M$[9]
- Z, conjunto de atributos / $card(Z) = k$ (medidas)
- $\{e_1, ..., e_k\}$, expresiones de agregación
- $Y / Y \subset \{A_1, ..., A_n\} \wedge Y \subset (Att_D(R) \cup Att_M(R))$, atributos que serán removidos
- instancia origen : r

[7] De ahí el nombre de la herramienta.
[8] El conjunto de las "measure relations" son las "crossing relations" que tienen al menos un atributo de medida. A su vez el conjunto de las "crossing relations" son las relaciones que representan relaciones o combinaciones entre los elementos de un grupo de dimensiones.
[9] Rel_M es el conjunto de relaciones de medida.

Y en la implementación simplemente se cambia levemente el conjunto Y que se toma por los atributos que permanecen en vez de los que se remueven (solamente por un tema de facilidad al momento de implementar).

La sintaxis concreta con la que se da el comando es[10]:
'agregate operator' '(' [{Atributos} / record format '(' {Atributos de RF} ')'] ';' {Atributos a Agregar} ')'
Donde:

- 'Atributos' Es una lista de identificadores de atributos con la forma: {Atributo} [',' {Atributo}]*, donde:
  - 'Atributo' Es un atributo definido en el formato de registro de entrada o salida del nodo.
- 'Atributo de RF' Es idéntico a 'Atributos' salvo que cada atributo en realidad es el nombre de un formato de registro.
- 'Atributos a Agregar' Es una lista de identificadores de atributos con la forma: '(' {Atributo}',' {Agregación} ')' [',' '(' {Atributo}',' {Agregación} ')']*, donde:
  - 'Atributo' Es el definido anteriormente y
  - 'Agregación' es uno de los posibles métodos de agregación (p.e. 'sum'), use 'list agregators' para mas información.

El Ejemplo

Para mostrar la aplicación en funcionamiento lo que se planteo fue seleccionar un problema en concreto, y plantearse utilizar la herramienta para resolverlo. La elección del problema no fue para nada trivial, dado que para mostrar la herramienta no había mucho problema, pero si en realidad se desea hacer interesante y que la versión paralela tenga posibilidades de ser mas eficiente que la no paralela no puede elegirse cualquiera.

El problema que se atacara es el de generación de redundancia para el tema de análisis OLAP. Para esto se planteo un problema típico de ejemplo que en nuestro caso pasa por el análisis de ventas en función de la cantidad vendida y el importe. Para analizar esta información se plantearon como dimensiones: el supervisor del vendedor que realizo la venta, el cliente que la compro, el producto que compro y en que fecha lo hizo. Estas dos ultimas dimensiones además con jerarquías conformadas por grupo de producto, tipo de producto y producto en el primer caso y año, mes y día en el segundo. El siguiente esquema estrella intentara mostrar el problema gráficamente:

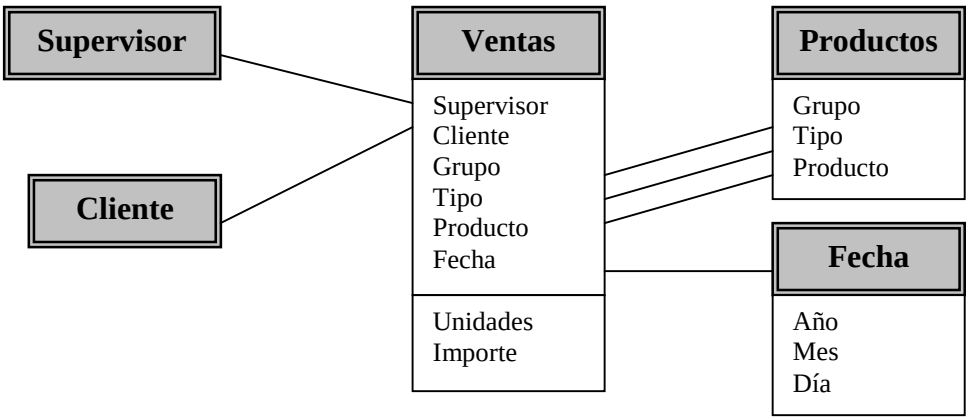

Figura 4: Diagrama estrella del ejemplo propuesto.

Repasando lo que queremos hacer (ahora ya con el diagrama), se trata de generar las agregaciones necesarias para tener pre-calculada la redundancia de los posibles puntos de análisis sobre el diagrama anterior. Estos son los distintos cruzamientos de las dimensiones antes presentadas.
Por ejemplo, una opción puede ser, mirar todos los supervisores, analizando todos los clientes y viendo todos los productos (sin jerarquizar por grupo y tipo) por año. Esto involucra agregar la información[11] por año, es decir eliminar el detalle por meses y días de los datos originales.

[10] Dentro de la aplicación esta disponible con 'help define nodes agregate operator', por mas información referirse al .
[11] De aquí la utilización de la primitiva de agregación (P9).

Sin tomar en cuenta las jerarquías de las dimensiones Producto y Fecha, estimando también que de las fechas conviene tomar como nivel inferior de detalle los años nos queda para analizar posibilidades en función de cuatro variables S (Supervisor), C (Cliente), P (Productos), A (Año)[12].

Ahora nuestro problema pasa a como plantear todos nuestros posibles cruzamientos de dimensiones (es decir donde calcular redundancia). Bueno una estrategia puede ser la siguiente: dadas cuatro variables en juego si las colocamos todas significara que estamos analizando las cuatro disgregadas, es decir sin resumir ninguna de las variables (si obviamente de los datos de entrada ya estamos resumiendo dado que el nivel de detalle era superior, pero para simplificar no colocaremos una agregación aquí). Esto podría ser un primer objetivo (SCPA) en nuestro problema donde todas estas variables se encuentran sin manipular.
El siguiente paso podría ser resumir una sola de ellas a la vez y mantener las otras intactas, lo que daría origen a: CPS, SPA, SCA y SCP. Así podríamos ahora hacerlo con dos por lo que tendríamos: CP, SA, PA, SC, CA, SP y finalmente tres: C, P, A, S.

Si miramos un poco en detalle esto ultimo vemos que nos lleva a trabajar con combinatorias quedándonos la formula: $C^4_4 + C^4_3 + C^4_2 + C^4_1 = 1 + 4 + 6 + 4 = 15$ que nos da la cantidad de cruzamientos posibles para este caso[13].
Ahora el otro problema que enfrentamos es como organizar el grafo del proceso para realizar los cálculos. Lo primero que podemos ya intuir es que cada una de las agregaciones se realiza en un nodo utilizando el operador de agregación que presentamos en anteriormente, ahora como conectamos estos nodos?, los conectamos?.

Una alternativa es un grafo totalmente desconexo (obviando los nodos de lectura y escritura) que representaría que cada nodo lee los datos realiza la agregación y graba el resultado. Rápidamente podemos deducir que hay una forma aparentemente mas inteligente de reaprovechar el trabajo de los otros nodos, mirando el problema global de calcular todas las redundancias seria hacer una **partición funcional** del problema. Esto da entrada a nuestra primera propuesta de grafo:

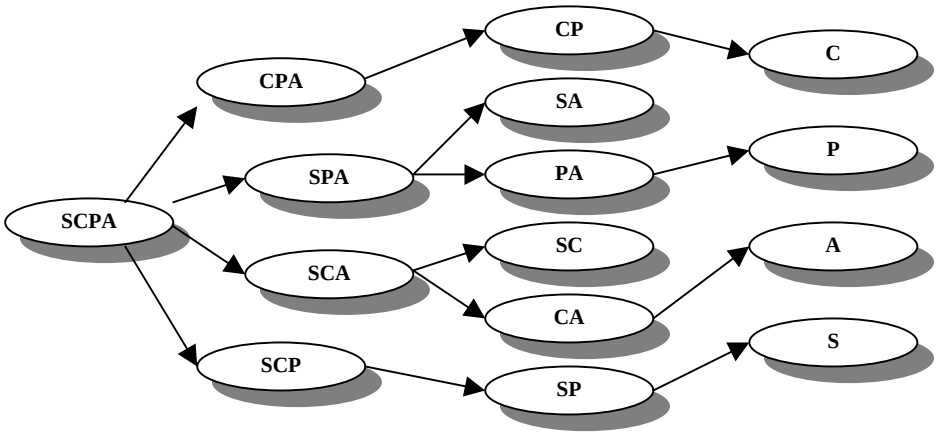

Figura 5: Grafo del proceso con partición funcional (pipeline)

Otra posibilidad que también rápidamente se nos puede venir a la mente es aplicar la técnica de **partición de dominio** esto como veremos en las líneas futuras podría facilitarse ampliamente, pero no se incorporo para la presente versión. La idea a seguir es simple, se pueden partir los datos del dominio de entrada y crear múltiples nodos para resolver el problema y luego juntar el resultado.
En nuestro ejemplo no tiene diferencia a los efectos del resultado final calcular sub-agregaciones y luego realizar una agregación final, pero a los efectos prácticos esto puede tener ventajas de performance porque los volúmenes de datos para cada nodo son mas manejables.

[12] Esto se realizo dado que era inmanejable afrontar el problema con mas de estas variables para el marco de este trabajo, y si se hubiera hecho así su aporte en relación no seria significativo.
[13] Esto de una manera similar salvo que eliminando situaciones inconsistentes por el hecho de tener jerarquías nos hubiera permitido poner una cota superior en la cantidad de casos del problema original en el orden de los 255 casos que hacia inmanejable el problema.

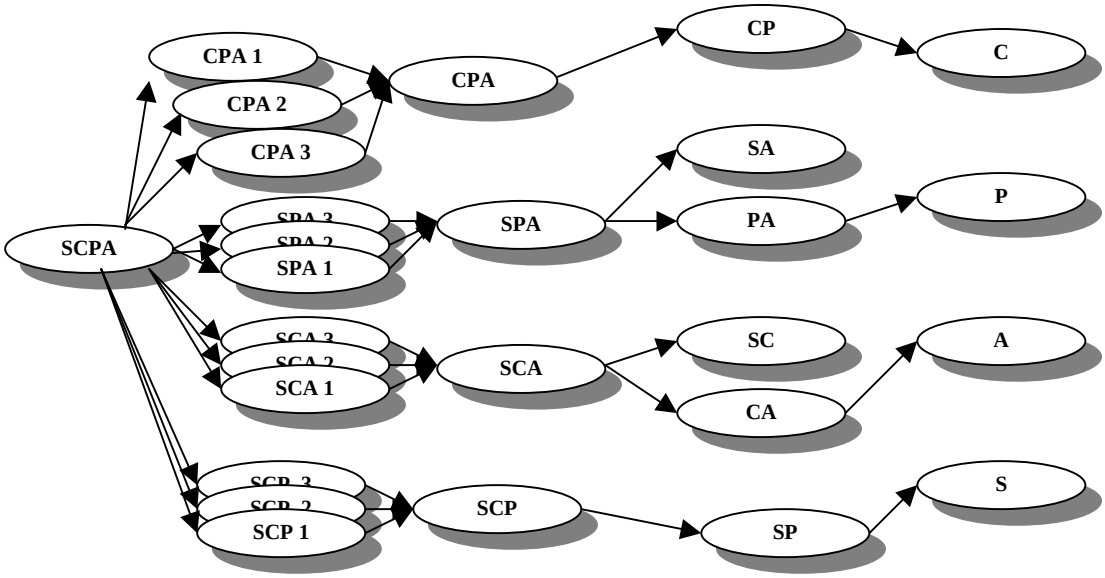

Figura 6: Grafo alternativo para el problema con partición de dominio.

Para redondear fuera de cual sea el grafo que se elija para ejecutar la solución (dado que ambos producen el mismo resultado), agregándole el nodo de lectura y los de grabación correspondientes tendremos nuestra solución para resolver el ejemplo.

Los Resultados

Presentados ya los problemas pasamos a mostrar y comentar los resultados obtenidos. Para empezar aclararemos que a pesar que presentamos dos posibles alternativas de grafos para el proceso solo pudimos realizar mediciones para uno de ellos[14], además de esto a pesar que comenzamos con un rango mas extenso de tamaños de entrada (~10Mb) solo llegamos hasta trabajar con 6000Kb.

Aquí la acotación mas general tiene que ver con que dado que una de las fortalezas de la herramienta es poder cambiar rápidamente la arquitectura de la solución una vez propuesta una versión inicial, la variedad de casos que podríamos tener seria importante. Por ende fuera del alcance de este estudio. Esto nos llevara a imponer una serie de hipótesis de forma de poder cerrar el presente trabajo, de las cuales ya presentamos algunas en el párrafo anterior.

Nuevamente con el fin de acotar el campo introducimos una hipótesis al problema que será que los datos de entrada siempre estarán locales al equipo en el cual se realiza el procesamiento. A su vez los datos de salida también se depositaran en el equipo que realizo el procesamiento evitando nuevamente su pasaje por la red[15].

Los equipos con los que se contaba eran:

- Equipo A: Intel Pentium MMX, 133 Mhz, 128Mb de RAM.
- Equipo B: Intel Pentium MMX, 200 Mhz, 96Mb de RAM.
- Equipo C: Intel Pentium MMX, 166 Mhz, 128Mb de RAM.
- Equipo D: Intel Pentium MMX, 200 Mhz, 128Mb de RAM.

Los cuales se distribuyeron de la siguiente forma:

- Prueba No Paralela: A;
- Prueba Paralela 1 equipo: A;
- Prueba Paralela 2 equipos: A, B;
- Prueba Paralela 3 equipos: A, B, C;
- Prueba Paralela 4 equipos: A, B, C, D.

[14] Principalmente la dificultad radicaba en el tiempo que tomaba realizar las mediciones.
[15] Notar que esto claramente es solo por la introducción de una hipótesis particular, la herramienta de ninguna manera restringe a que esto sea necesario.

La red con la que se realizaron las pruebas es una LAN Ethernet de 10Mb, con muy poca carga dado que las pruebas se realizaban en horario nocturno prácticamente contando con todo el ancho de banda disponible.

Un aspecto importante que no podemos pasar por alto y que posiblemente sea ya una pregunta pendiente es como finalmente se distribuyen los nodos en las maquinas en el caso de que hallan varias. Tomando en cuenta los Equipos mencionados anteriormente estos serian los grafos correspondientes a las opciones con mas de un equipo[16]:

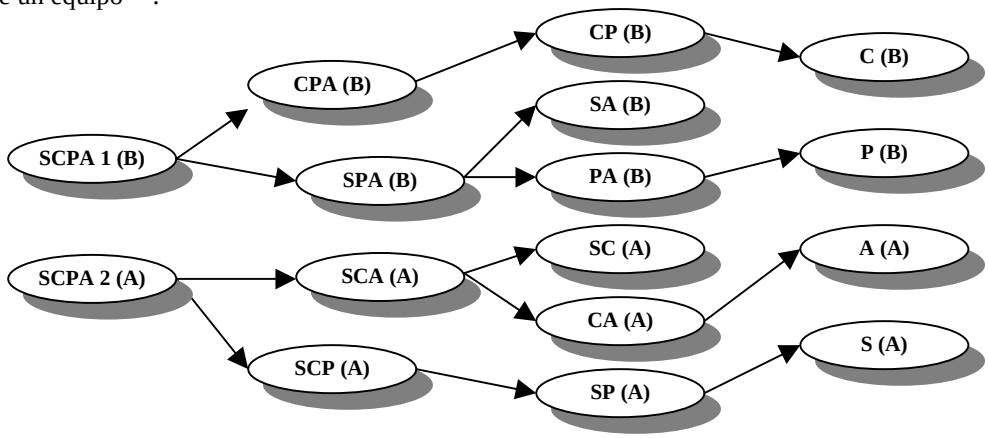

Figura 7: Distribución de nodos en los equipos (en el caso de 2 maquinas)

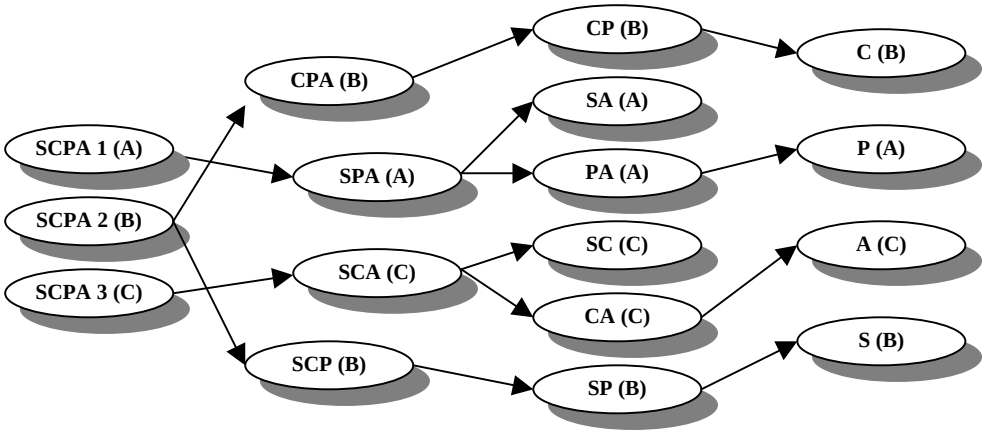

Figura 8: Distribución de nodos en los equipos (en el caso de 3 maquinas)

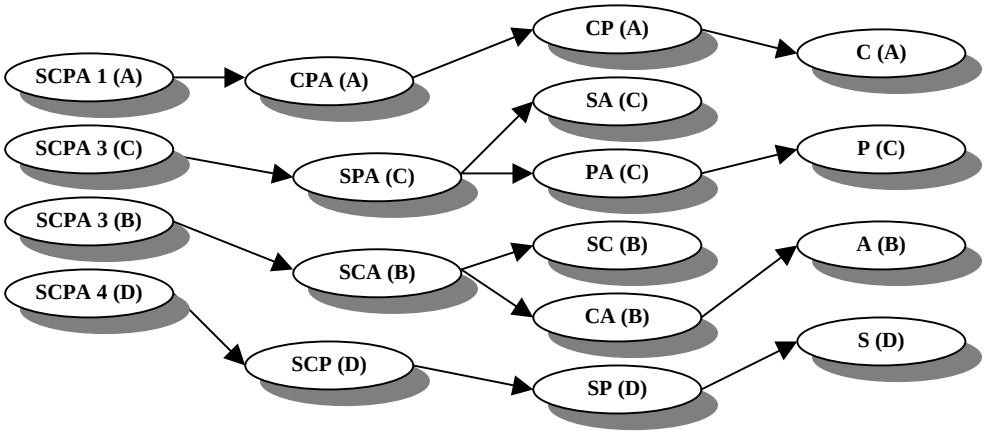

Figura 9: Distribución de nodos en los equipos (en el caso de 4 maquinas)

[16] Vale recordar que los nodos de grabación de la salida de la agregación que no se dibujan, o sea el "write operator" asociado a cada uno de los nodos de interés se coloca en la misma maquina que el nodo de calculo siguiendo lo expresado en la hipótesis anteriormente.

Tomando en cuenta la hipótesis introducida anteriormente y mirando los distintos grafos vemos que el objetivo perseguido fue minimizar el uso de la red, lo cual si se ve con detenimiento fue logrado. Luego de que el trabajo es dividido, todo el procesamiento de ahí en adelante se realiza en la misma maquina.

Para cada uno de los casos que se expondrá a continuación se tomaron 5 mediciones y lo que se presentara será el resultado promedio[17].

Tamaño	No Paralela	Paralelo (1 equipo)	Paralelo (2 equipos)	Paralelo (3 equipos)	Paralelo (4 equipos)
1K	102	40448	23518	18708	9580
10K	1124	29254	25012	18446	9996
100K	12566	41656	29176	28076	15140
200K	23354	51322	35394	32240	18378
300K	32418	63186	38460	31352	22850
400K	41138	72832	36724	32368	25716
500K	53860	81342	39720	36204	26880
600K	65340	92176	43710	37800	29846
700K	72798	103952	48522	43830	35404
800K	84366	112916	53190	45478	34450
900K	95000	123736	57114	48334	41018
1000K	105446	138336	61804	51994	41884
2000K	216878	249952	113300	89462	66260
3000K	295908	382016	167184	115498	88046
4000K	387126	471470	221110	153562	115892
5000K	503922	569250	273672	182706	148948
6000K	610166	630006	322238	217870	180784

Tabla 1: Tabla de resultados de la mediciones (clocks).

En base a estos resultados podemos ver el Speed-Up para los distintos tamaños, en la Ilustración 1, como vemos a medida que aumenta el tamaño del problema el speed-up acompaña este crecimiento.

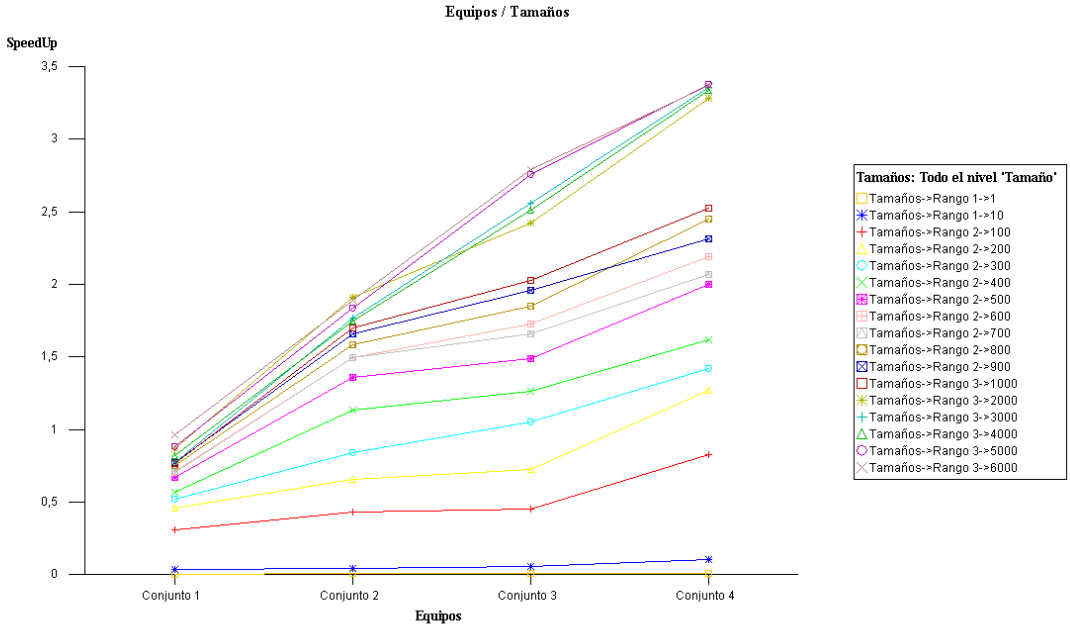

Ilustración 1: Grafica del speed-up del proceso para todos los tamaños[18].

[17] La tabla con los valores correspondientes se presenta en el .

[18] Solo se están mostrando los problemas categorizados como Paralelos, porque para los Secuenciales no tendría sentido (esto implica seleccionar en la herramienta O3 que muestre solo los Problemas Paralelos).

En esta Ilustración los diversos conjuntos son los formados por los distintos grupos de maquinas (Conjunto 1 -> 1 Maquina, Conjunto 2-> 2 Maquinas, etc.). Cabe señalar para esta y las siguientes ilustraciones que el "Conjunto 1" se refiere a aplicar la solución paralela pero solo trabajando con un maquina, la solución secuencial aunque también se ejecuta en un procesador no utiliza los mecanismos de la paralela, propios de la separación en procesos y comunicación de estos. En este ultimo aspecto obviamente la propuesta secuencial en un procesador es mucho mas performante que la paralela, esto se vera constantemente en las graficas.

Para facilitar su análisis sin embargo podemos separar en 3 rangos de datos (que es la otra parte de la descripción que se ve en la Ilustración):

- Rango 1: 1K a 10K, muy pequeña cantidad de datos.
- Rango 2: 100K a 900K, pocos datos.
- Rango 3: 1000K a 6000K, volúmenes un poco mayores de datos.

Con esta clasificación tenemos:

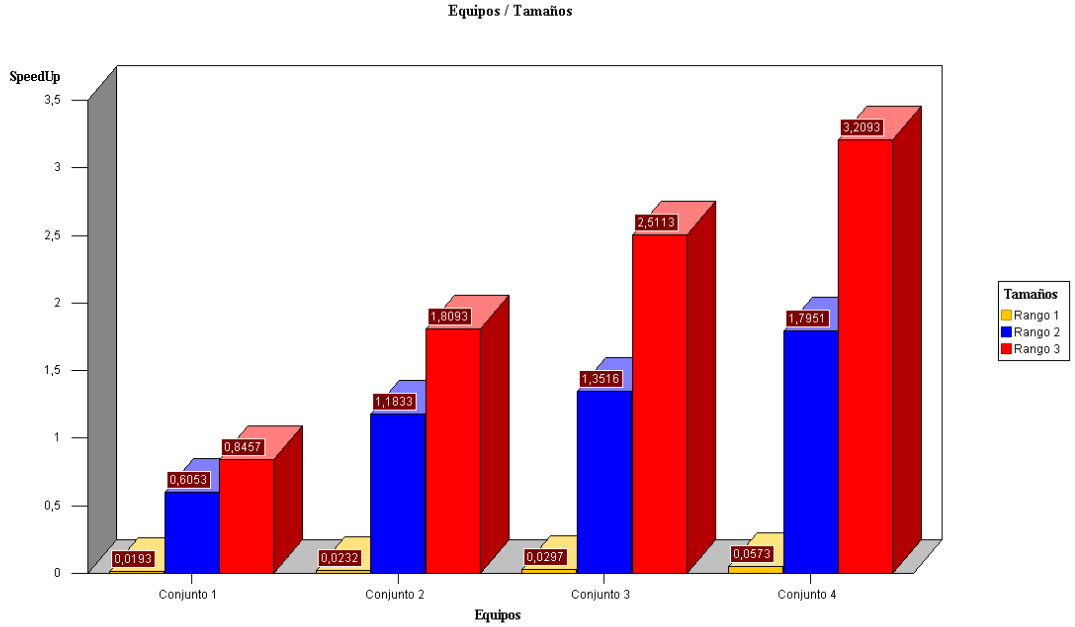

Ilustración 2: Análisis de Speed-Up considerando los grupos.

donde se puede ver que en el rango 1 (muy pequeña cantidad de datos) no tiene sentido aplicar paralelismo, el valor de este esta siempre bajo la unidad, recordando el calculo[19] significa que el tiempo de ejecución en N equipos es siempre inferior al secuencial.
Por el contrario en los otros dos rangos esto ya no es así, sino que la aplicación del paralelismo en general mejora el Speed-Up y por tanto el tiempo de ejecución se reduce a medida que se colocan mas equipos. Otra observación que se puede ver incluso mejor aun cambiando a grafico a líneas[20], es que el rango 2 esta por debajo del rango 3 y se aleja de el a medida que aumenta la cantidad de equipos. Esto tampoco es raro porque indicaría que a mayor tamaño de entrada la pendiente del grafico es mayor y por lo tanto se gana mas al agregar mas equipos; esto también podía verse en la Ilustración 1 dado que si miramos con detenimiento cada grafica tiene mayor pendiente que la anterior (en general).

En el Apéndice – Tablas de Resultados y en particular en la Tabla 8 se muestran los valores mas claramente.

---

[19] Speed-Up=[Tiempo Secuencial]/[Tiempo Paralelo](N). Por mas información referirse sobre el calculo de Speed-Up referirse a [MPER].

[20] Adjunto a este informe se entregara entre otras cosas el cubo multidimensional con el que se realizo el análisis, obviamente la discusión de las dimensiones y medidas que ahí se tienen escapa al alcance de este documento. Es importante resaltar que es muy sencillo (como era de esperarse en una herramienta OLAP) variar las representación y ángulos con los cuales se enfoca el problema. En esta línea se agregara información observada en la herramienta O3 que fue la que se utilizo, dejando a criterio del lector observarla abriendo los archivos adjuntos o no (por mas información remitirse a [O3]).

La observación del Speed-Up refleja también que es lo que sucede con los tiempos de ejecución secuencial con respecto a los paralelos, esto precisamente, los tiempos de ejecución es lo que se grafica a continuación:

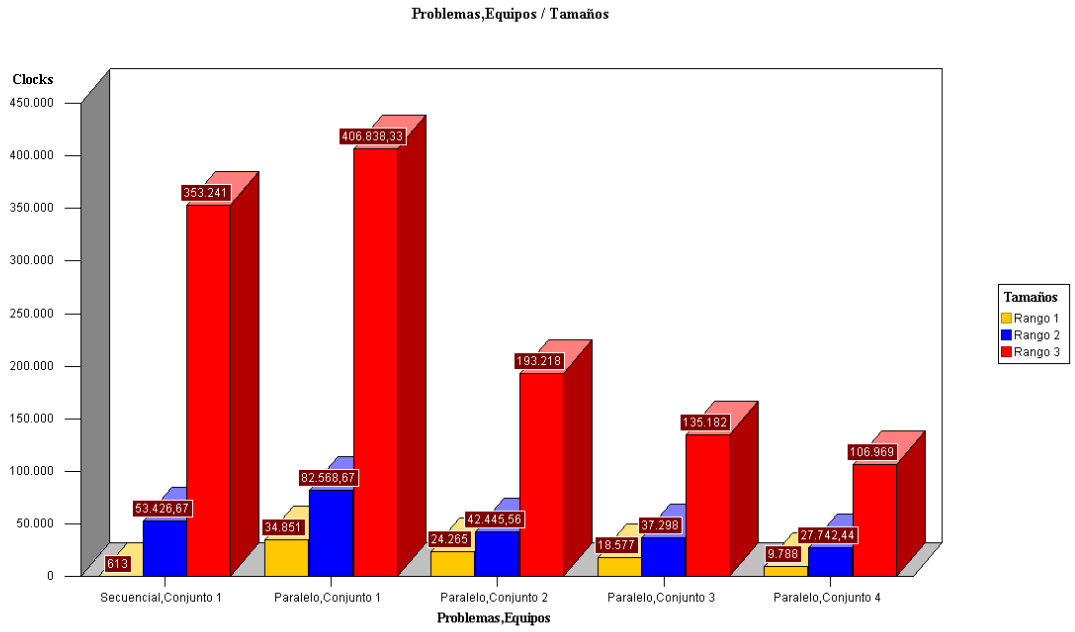

Ilustración 3: Tiempos de ejecución secuenciales vs. paralelos.

Dos visiones adicionales que agregamos ahora son la Eficiencia y Eficacia (sobre las cuales se puede encontrar mas documentación en [MPER]):

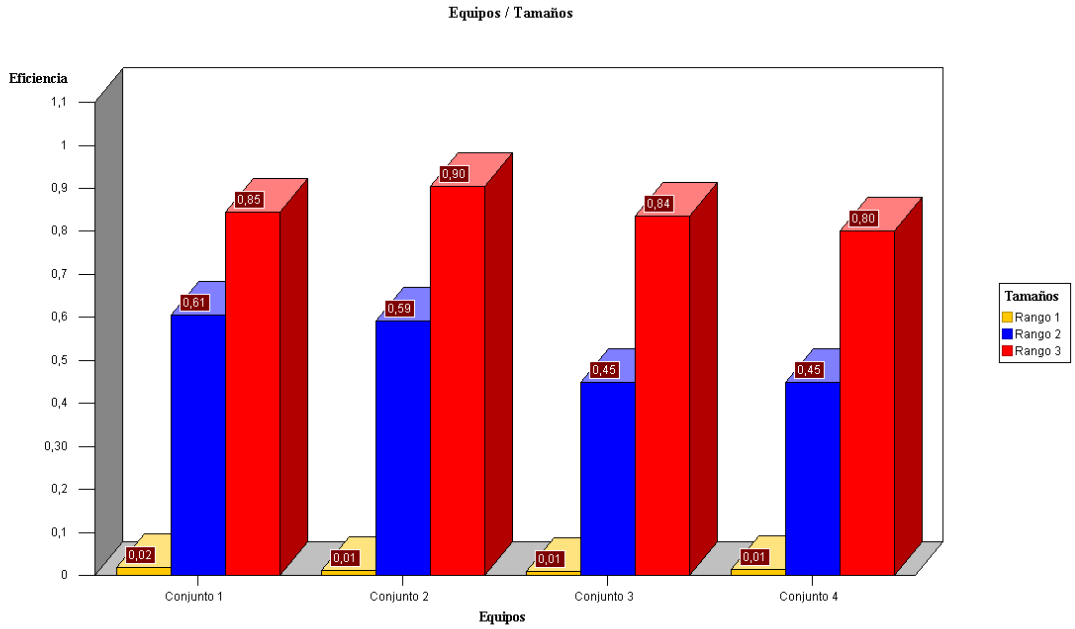

Ilustración 4: Grafico de la Eficiencia.

La eficiencia según se explica en [MPER] nos permite ver cual es el porcentaje con que contribuye cada procesador en la computación global, como vemos en general este va decreciendo a medida que agregamos procesadores, sin embargo no confundamos como veremos luego de la siguiente ilustración esto no quiere decir que no sea conveniente agregar otro procesador.

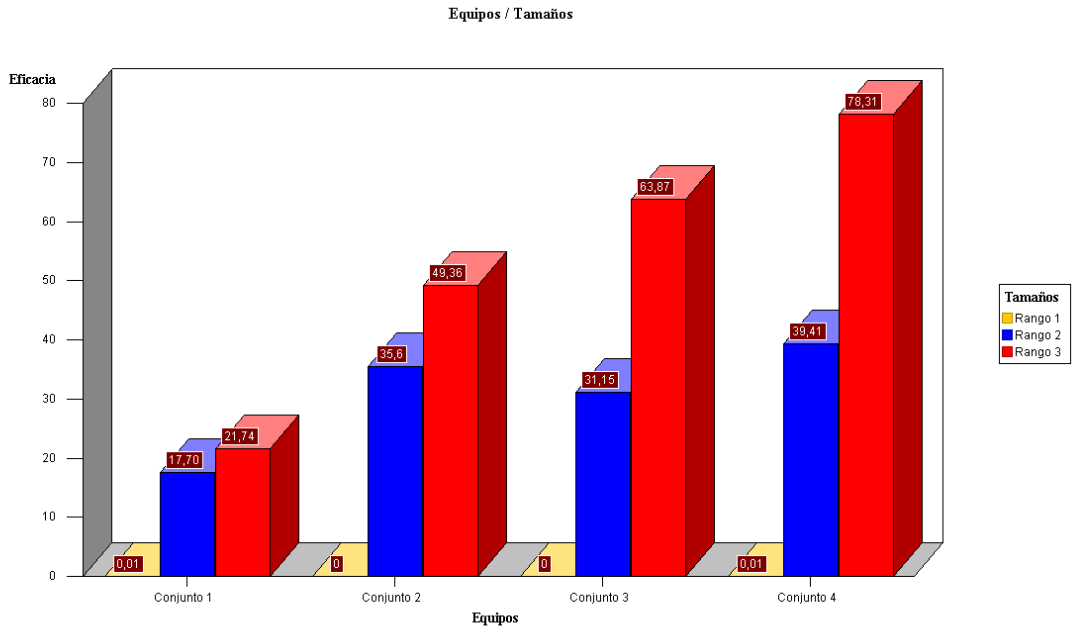

Ilustración 5: Grafico de la Eficacia.

Observando el grafico de la Eficacia y rescatando el resultado comentado en [MPER] que indica que el primer máximo de la curva corresponde con un punto optimo operacional, podemos ver que en el rango de problemas 100K – 900K encontramos un máximo operacional utilizando 2 equipos. Sin embargo en el rango de problemas 1000K – 6000K y llegando hasta trabajar con 4 equipos por lo menos no hemos encontrado un máximo (confirmado, pues 4 equipos podría serlo dependiendo de lo que suceda con 5), lo cual nos sugiere que seguir agregando equipos mejorara la relación costo / beneficio[21].

Líneas de Trabajo Futuro

Podemos presentar varias líneas de trabajo, muchas de ellas como veremos si bien se planteaban en [BBGS] o en [MG2000] escaparon al alcance de este trabajo. Entre ellas tenemos:

Nuevos operadores: Contamos con cuatro operadores (lectura, escritura, identidad y agregación) la mayoría de estos fueron considerados básicos para cerrar un primer acercamiento a la solución dentro del alcance del proyecto, pero claramente se podría trabajar por ejemplo en implementar todas las primitivas que se presentan en [AM2000] con lo cual se aumentaría considerablemente el poder de la herramienta.

Particiones de dominio automáticas: Uno de los puntos en los que vimos se podía ampliar la herramienta de una forma muy practica trata el tema de facilitar en ciertos casos realizar particiones de dominio automáticas. Obviamente según el tipo de problema esta puede ser una solución viable o no, pero en caso que lo sea esto parece una buen característica que la herramienta podría incorporar.

Ejecución por maquina mono proceso: Cuando se plantea el grafo y en particular una serie de nodos se envían a ejecutar a una misma maquina, tenemos un overhead que podría ser fácilmente quitado realizando una ejecución de ese conjunto de nodos de forma centralizada, es decir dentro de un mismo proceso. Se trataría de hacer que ese sub-grafo actuara como un solo nodo hacia el resto del sistema.

Mejorar las estrategias de comunicación: Si bien se resolvió el problema de la comunicación de los nodos y el pasaje de información, e incluso se mejoro frente a una primera versión en la que se transmitía un registro por mensaje, es claro pensar que analizando mas en profundidad el tema se podría mejorar. Esto ultimo principalmente porque no se hizo mucho hincapié en esto al momento de resolverlo.

Mejorar el análisis de los resultados obtenidos: Si bien se realizo un análisis de los datos obtenidos bastante razonable a nuestro entender es claro que este podría haber sido explotado mucho mas. Ya sea agregando mas casos, tomando en cuenta otras configuraciones de equipos (mas potentes), examinando las líneas que se dejaron fuera por cuestiones de alcance en este trabajo, como ser otras posibles distribuciones del grafo, etc. Una mejora adicional pero en esta misma línea también seria facilitar mecanismos automáticos para registrar los tiempos de ejecución, incluso aumentando la cantidad de

[21] El caso de el rango 1K – 10K no lo analizamos por la escasa variación de los valores, aunque no se halla realizado un estudio adecuado del error introducido en las mediciones parece convincente que todos los valores +/- el error estén en el mismo rango.

información, por ejemplo incluyendo el tamaño de datos enviado por la red, la cantidad de comunicaciones, etc.

Conclusiones

Sobre los objetivos iniciales planteados de cobrar experiencia en la herramienta PVM a nuestro parecer esto se alcanzo, ya que se llegaron a manejar bastantes áreas (creación de procesos, comunicación, sincronización, etc.), e incluso se llego a pasar por esos detalles que uno no descubre hasta que le da problemas y encuentra solución en un recóndito lugar del manual.

Se implementó la primitiva P9 y se utilizó en forma practica para plantear una solución a un problema de generación de redundancia. Además potencialmente no parece irreal decir que el uso del paralelismo pueda mejorar procesos de carga similares a estos aunque no se tengan datos concretos.

También se puede notar al leer las posibles líneas de trabajo que hay buen material por donde seguir si se quiere, e incluso algunos de los puntos parecen no estar lejos de poderse completar en un corto plazo.
Un aspecto que resulto interesante que vale la pena resaltar y que se constato luego de haber finalizado el desarrollo, fue la facilidad para cambiar de estrategia al momento de diseñar el proceso de transformación y así rápidamente poder probar otra nueva alternativa para resolver el mismo problema.

Sobre el análisis de tiempos del problema como ya se adelanto un poco mas atrás los resultados fueron bastante prometedores, efectivamente sucedió que a medida que se incorporaban equipos a la maquina virtual el tiempo necesario para resolverla bajaba. Mejor que esto que puede considerarse como un resultado particular el Speed-Up del problema si bien no era lineal (pendiente 1) tenia pendiente positiva. También pudimos constatar que a medida que aumentamos el tamaño del problema esta pendiente crecía en relación, aunque es natural intuir también que esto solo ocurrirá hasta cierto momento denota un resultado positivo del estudio.

No obstante lo positivo de los resultado antes mencionados debemos aclarar también el carácter restrictivo de las hipótesis impuestas, ya que como vemos se minimizo el pasaje de datos por la red, haciéndolo casi nulo. Los datos siempre se encontraron en el equipo en que se ejecutaba la operación, los resultados se colocaban también en este equipo, etc.

Sin embargo es justo también decir que lo que se planteo fue un escenario posible y se trabajo sobre este, si bien podrían existir muchas alternativas que no se consideraron, tratar de atacarlas todas hubiera sido irrazonable para el alcance de este trabajo. Como lo hubiera sido también intentar una respuesta muy general al problema como ser "*el paralelismo siempre mejora el proceso o no*", en lugar de esto lo que se logro desde nuestro punto de vista fue "*en este caso y con las hipótesis dadas el paralelismo mejora la solución sensiblemente*".

Apéndice – Ejemplo completo de un proceso

Presentaremos aquí un ejemplo completo que se menciono en el documento y que fue utilizado como prueba de la aplicación. Ya anteriormente se presento el diagrama estrella de análisis que se realizo pero no el formato de todos los datos involucrados[22].

Nombre de Campo	**Tamaño**	**Tipo**
Supervisor	10	String
Cliente	10	String
Grupo	10	String
Tipo	10	String
Producto	10	String
Día	2	Integer
Mes	2	Integer
Año	4	Integer
Unidades	10	Integer
Importe	10	Float

Tabla 2: Tabla de definición de atributos del ejemplo.

Lo primero que presentaremos serán los formatos de registro necesarios para intercambiar la información de los nodos[23]:

```
define record format FormatoDeMedidas { Units,10,Integer; Amount,10,Double };

define record format S { Supervisor,10,String };
define record format C { Client,10,String };
define record format P { Group,10,String; Type,10,String; Product,10,String };
define record format A { Year,4,Integer };

define record format FormatoS extends S, FormatoDeMedidas;
define record format FormatoC extends C, FormatoDeMedidas;
define record format FormatoP extends P, FormatoDeMedidas;
define record format FormatoA extends A, FormatoDeMedidas;

define record format FormatoSC extends S, C, FormatoDeMedidas;
define record format FormatoSP extends S, P, FormatoDeMedidas;
define record format FormatoSA extends S, A, FormatoDeMedidas;
define record format FormatoCP extends C, P, FormatoDeMedidas;
define record format FormatoCA extends C, A, FormatoDeMedidas;
define record format FormatoPA extends P, A, FormatoDeMedidas;

define record format FormatoSCP extends S, C, P, FormatoDeMedidas;
define record format FormatoSCA extends S, C, A, FormatoDeMedidas;
define record format FormatoSPA extends S, P, A, FormatoDeMedidas;
define record format FormatoCPA extends C, P, A, FormatoDeMedidas;

define record format FormatoSCPA { Supervisor,10,String; Client,10,String; Group,10,String;
Type,10,String; Product,10,String; Day,2,Integer; Month,2,Integer; Year,4,Integer } extends
FormatoDeMedidas;
```

Como vemos es posible utilizar entre otras cosas la noción de herencia entre formatos de registro para facilitar su especificación. A continuación será necesario definir los nodos:

```
define nodes {
        NodoSCPA, read operator (W:\cprojects\data\Ventas-140249L.txt);

        NodoS, agregate operator (Supervisor; (Units, sum), (Amount, sum));
```

[22] Cabe resaltar que el formato presentado en la tabla será el que a continuación dará base a los formatos de registro definidos por eso se incluye especialmente.

[23] Código textual sacado de la distribución de la herramienta que se envía como ejemplo.

NodoC, agregate operator (Client; (Units, sum), (Amount, sum));
NodoP, agregate operator (Group, Type, Product; (Units, sum), (Amount, sum));
NodoA, agregate operator (Year; (Units, sum), (Amount, sum));

NodoSC, agregate operator (record format (S,C); (Units, sum), (Amount, sum));
NodoSP, agregate operator (record format (S,P); (Units, sum), (Amount, sum));
NodoSA, agregate operator (record format (S,A); (Units, sum), (Amount, sum));
NodoCP, agregate operator (record format (C,P); (Units, sum), (Amount, sum));
NodoCA, agregate operator (record format (C,A); (Units, sum), (Amount, sum));
NodoPA, agregate operator (record format (P,A); (Units, sum), (Amount, sum));

NodoSCP, agregate operator (record format (S,C,P); (Units, sum), (Amount, sum));
NodoSCA, agregate operator (record format (S,C,A); (Units, sum), (Amount, sum));
NodoSPA, agregate operator (record format (S,P,A); (Units, sum), (Amount, sum));
NodoCPA, agregate operator (record format (C,P,A); (Units, sum), (Amount, sum));

SaveNodoS, write operator (W:\cprojects\data\calculated\nodeS.txt);
SaveNodoC, write operator (W:\cprojects\data\calculated\nodeC.txt);
SaveNodoP, write operator (W:\cprojects\data\calculated\nodeP.txt);
SaveNodoA, write operator (W:\cprojects\data\calculated\nodeA.txt);

SaveNodoSC, write operator (W:\cprojects\data\calculated\nodeSC.txt);
SaveNodoSP, write operator (W:\cprojects\data\calculated\nodeSP.txt);
SaveNodoSA, write operator (W:\cprojects\data\calculated\nodeSA.txt);
SaveNodoCP, write operator (W:\cprojects\data\calculated\nodeCP.txt);
SaveNodoCA, write operator (W:\cprojects\data\calculated\nodeCA.txt);
SaveNodoPA, write operator (W:\cprojects\data\calculated\nodePA.txt);

SaveNodoSCP, write operator (W:\cprojects\data\calculated\nodeSCP.txt);
SaveNodoSCA, write operator (W:\cprojects\data\calculated\nodeSCA.txt);
SaveNodoSPA, write operator (W:\cprojects\data\calculated\nodeSPA.txt);
SaveNodoCPA, write operator (W:\cprojects\data\calculated\nodeCPA.txt)

};

Aquí también pudimos ver otro aspecto que facilita la utilización que es utilizar la definición de formatos de registro para especificar los nodos que se desea mantengan en la agregación. Finalmente definiremos las aristas del grafo:

define edges {

AristaSCPA-SCP, NodoSCPA, NodoSCP, FormatoSCPA;
AristaSCPA-SCA, NodoSCPA, NodoSCA, FormatoSCPA;
AristaSCPA-SPA, NodoSCPA, NodoSPA, FormatoSCPA;
AristaSCPA-CPA, NodoSCPA, NodoCPA, FormatoSCPA;

AristaSCA-SC, NodoSCA, NodoSC, FormatoSCA;
AristaSCP-SP, NodoSCP, NodoSP, FormatoSCP;
AristaSPA-SA, NodoSPA, NodoSA, FormatoSPA;
AristaCPA-CP, NodoCPA, NodoCP, FormatoCPA;
AristaSPA-PA, NodoSPA, NodoPA, FormatoSPA;
AristaSCA-CA, NodoSCA, NodoCA, FormatoSCA;

AristaCP-C, NodoCP, NodoC, FormatoCP;
AristaSP-S, NodoSP, NodoS, FormatoSP;
AristaPA-P, NodoPA, NodoP, FormatoPA;
AristaCA-A, NodoCA, NodoA, FormatoCA;

AristaSaveS, NodoS, SaveNodoS, FormatoS;
AristaSaveC, NodoC, SaveNodoC, FormatoC;
AristaSaveP, NodoP, SaveNodoP, FormatoP;
AristaSaveA, NodoA, SaveNodoA, FormatoA;

```
        AristaSaveSC, NodoSC, SaveNodoSC, FormatoSC;
        AristaSaveSP, NodoSP, SaveNodoSP, FormatoSP;
        AristaSaveSA, NodoSA, SaveNodoSA, FormatoSA;
        AristaSaveCP, NodoCP, SaveNodoCP, FormatoCP;
        AristaSavePA, NodoPA, SaveNodoPA, FormatoPA;
        AristaSaveCA, NodoCA, SaveNodoCA, FormatoCA;

        AristaSaveSCP, NodoSCP, SaveNodoSCP, FormatoSCP;
        AristaSaveSCA, NodoSCA, SaveNodoSCA, FormatoSCA;
        AristaSaveSPA, NodoSPA, SaveNodoSPA, FormatoSPA;
        AristaSaveCPA, NodoCPA, SaveNodoCPA, FormatoCPA
};
```

Esto define los formatos de registro, nodos y aristas del grafo necesarios para correr un ejemplo y generar los archivos con las agregaciones calculadas. Cabe resaltar que los caminos ya sea del archivo origen y destino claramente deben adaptarse a la instalación en particular.

Una vez realizado esto y colocado correctamente el archivo de origen de datos (según lo especificado en el NodoSCPA) estamos en condiciones de utilizar el comando "**execute**" mencionado en el "Apéndice – Comandos de la Herramienta" si lo que deseamos es hacer correr la ejecución en forma distribuida o el comando "**execute alone**" si lo que queremos es que la ejecución se realice centralizada.

Apéndice – Comandos de la Herramienta

Los comandos disponibles en la aplicación son[24]:

- **define** : Define alguno de los elementos definibles del sistema (nodes, etc.).
  - Opciones: '**record format**', '**nodes**', '**edges**'.
- **list** : Lista los elementos ya definidos según el parámetro.
  - Opciones: '**record format**', '**nodes**', '**edges**', '**operators**', '**agregators**'.
- **help** : Presenta este help o el help particular de un comando u elemento.
- **quit** : Termina la aplicación.
- **file** : Lee el contenido de un archivo y lo ejecuta en el interprete.
- **execute** : Ejecuta en función de la información dada al sistema, en modo distribuido, es decir lanzando un proceso por operador en las distintas maquinas de la maquina virtual.
- **execute alone** : Ejecuta en función de la información dada al sistema, en modo autónomo, es decir un solo proceso que ejecuta toda la secuencia en esta maquina.

Apéndice – Tablas de Resultados

Aquí presentamos las tablas de mediciones para los cinco problemas expuestos, versión no paralela y paralela con 1, 2, 3 y 4 equipos.

Tamaño	M1	M2	M3	M4	M5	Promedio
1K	170	60	110	60	110	102
10K	1160	1150	1050	1160	1100	1124
100K	12360	12410	12410	12860	12790	12566
200K	23620	23010	22630	22850	24660	23354
300K	32460	31810	32130	32680	33010	32418
400K	40650	40200	39930	41360	43550	41138
500K	52070	51360	53380	56190	56300	53860
600K	59760	66240	66850	66020	67830	65340
700K	69090	68220	75030	75250	76400	72798
800K	79150	81180	86400	87170	87930	84366
900K	88320	91950	97600	98540	98590	95000
1000K	98090	103040	107930	108430	109740	105446
2000K	201080	215140	217340	230970	219860	216878
3000K	277270	293850	299510	302920	305990	295908

[24] Por mas información sobre la sintaxis de un comando en particular se recomienda usa el comando help mencionado aquí mismo.

4000K	365690	386950	390910	392000	400080	387126
5000K	476320	494330	505100	518060	525800	503922
6000K	600440	632800	642240	660870	514480	610166

Tabla 3: Tabla de mediciones para el problema no paralelo[25].

Tamaño	M1	M2	M3	M4	M5	Promedio
1K	42790	39330	40760	41190	38170	40448
10K	44710	26470	25210	24120	25760	29254
100K	56140	36190	37130	39820	39000	41656
200K	65750	47620	47570	47450	48220	51322
300K	78490	59480	58990	59320	59650	63186
400K	85580	70030	70030	68220	70300	72832
500K	99910	77390	76780	75680	76950	81342
600K	111720	87390	87990	86890	86890	92176
700K	118920	97160	96830	96510	110340	103952
800K	136820	106170	107100	107110	107380	112916
900K	149840	117540	116660	117540	117100	123736
1000K	159940	130340	147530	126610	127260	138336
2000K	278960	243370	241670	242990	242770	249952
3000K	381840	368770	345590	436930	376950	382016
4000K	499880	468570	462580	465170	461150	471470
5000K	571340	574140	573150	553100	574520	569250
6000K	642950	643070	605500	636310	622200	630006

Tabla 4: Tabla de mediciones para el problema paralelo con 1 equipo.

Tamaño	M1	M2	M3	M4	M5	Promedio
1K	22900	24230	22840	24110	23510	23518
10K	27460	25810	24830	26260	20700	25012
100K	34660	25050	30260	29660	26250	29176
200K	32850	37070	33290	37130	36630	35394
300K	38450	36910	38340	38830	39770	38460
400K	47080	33610	33120	34760	35050	36724
500K	44600	38720	38280	38060	38940	39720
600K	45700	43560	43500	42680	43110	43710
700K	48880	48780	48000	48780	48170	48522
800K	53820	53390	52290	54600	51850	53190
900K	57680	56130	57350	57510	56900	57114
1000K	62890	61850	60850	62180	61250	61804
2000K	117260	112430	112000	112480	112330	113300
3000K	172690	168290	166260	162750	165930	167184
4000K	219540	226020	218820	216520	224650	221110
5000K	279020	276990	271060	271990	269300	273672
6000K	323080	321590	321590	320210	324720	322238

Tabla 5: Tabla de mediciones para el problema paralelo con 2 equipos.

Tamaño	M1	M2	M3	M4	M5	Promedio
1K	21810	17800	16360	18780	18790	18708
10K	19610	17960	17910	17630	19120	18446
100K	33340	24660	26800	26850	28730	28076
200K	32900	35750	32410	30540	29600	32240
300K	28450	35760	26910	28400	37240	31352
400K	30870	32240	33890	32270	32570	32368
500K	37290	36150	37670	36360	33550	36204
600K	39270	37180	36750	38070	37730	37800
700K	43990	41850	46690	43770	42850	43830
800K	45090	44650	46800	44430	46420	45478
900K	48280	47840	49160	48280	48110	48334

[25] Los valores M1, ..., M5 de las columnas indican el numero de medicion, recordemos que se hicieron 5 mediciones para cada valor.

1000K	53560	51520	51680	51580	51630	51994
2000K	91230	87990	89420	88870	89800	89462
3000K	116880	118530	112050	112980	117050	115498
4000K	154560	150830	154120	153790	154510	153562
5000K	177580	183670	183670	183730	184880	182706
6000K	214160	218880	218550	223710	214050	217870

Tabla 6: Tabla de mediciones para el problema paralelo con 3 equipos.

Tamaño	M1	M2	M3	M4	M5	Promedio
1K	10270	8900	9840	9500	9390	9580
10K	10820	9780	9720	9780	9880	9996
100K	15760	14230	15380	15330	15000	15140
200K	18560	19120	17080	17740	19390	18378
300K	20650	22790	25600	22190	23020	22850
400K	25210	31580	23070	22740	25980	25716
500K	28340	27080	25210	26690	27080	26880
600K	30270	27570	31640	28780	30970	29846
700K	34100	37790	39930	30260	34940	35404
800K	35540	32850	32570	35970	35320	34450
900K	40920	38390	39660	43610	42510	41018
1000K	41740	40040	44320	37190	46130	41884
2000K	67120	60910	70140	62720	70410	66260
3000K	90690	87710	87330	87110	87390	88046
4000K	122370	113210	113470	114030	116380	115892
5000K	152090	148080	147590	148460	148520	148948
6000K	186970	174610	183290	180870	178180	180784

Tabla 7: Tabla de mediciones para el problema paralelo con 4 equipos.

Los datos individuales de cada medición no fuero objeto de estudio, pero sin embargo rápidamente se puede notar que la primera medición generalmente insume mas tiempo que el resto y que los valores de tiempo necesarios para realizar la operación se van reduciendo a medida que se repite la prueba.

Esto se debe a que las mediciones no son independientes entre si en muchos casos, en el sentido que se realizaron solicitando las 5 ejecuciones sucesivas al mismo proceso (en sintaxis de la herramienta 5 sentencias “execute” o “execute alone” ejecutadas una tras otra).
Además de esto el Sistema Operativo puede estar utilizando algún tipo de cache sobre los datos de entrada (los archivos) de forma de optimizar las próximas ejecuciones, sin contar otro tipo de optimizaciones que también puedan estarse realizando.

Para finalizar a continuación presentamos la tabla de valores de Speed-Up calculados para las pruebas, esta tabla se puede ver gráficamente en la Ilustración 2.

```
Dimensiones: Equipos / Tamaños
Medida: SpeedUp
Filtrado Por:
        Problemas = Paralelo
```

Tamaño	***Equipos***			
	Conjunto 1	Conjunto 2	Conjunto 3	Conjunto 4
Rango 1	0,019260000000000	0,023190000000000	0,0297	0,0573
Rango 2	0,605333333333333	1,183333333333333	1,351555555555556	1,795111111111111
Rango 3	0,845666666666667	1,809333333333334	2,511333333333333	3,209333333333333

Tabla 8: Valores de Speed-Up por rangos de valores (promediados) .

Referencias

- [BBGS] Tom Barclay, Robert Barnes, Jim Gray, Prakash Sundaresan, *Loading Databases Using Dataflow Parallelism*.
- [MG2000] Tobias Mayr, Jim Gray, *River Design*.
- [IL2001] Ignacio Larrañaga, *Análisis de Articulo: "Loading Databases Using Dataflow Parallelism"*.
- [AM2000] Adriana Marotta, *Data Warehouse Design and Maintenance through Schema Transformations*.
- [WPVM] http://www.epm.ornl.gov/pvm/, septiembre 2001.
- [MPER] *Measuring Performance*, Material de referencia publicado en la pagina del curso (http://www.fing.edu.uy/cecal/hpc, septiembre 2001).
- [O3] http://www.isoft.com.uy,septiembre 2001.
- [PVM94] Al Geist, ,Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Mancheck, Vaidy Sunderam, *PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for networked Parallel Computing*.