Concepción de Sistemas de Información

Instituto de Computación – Facultad de Ingeniería – Universidad de la República

SEMINARIO

Estudio de modelos y técnicas de workflow en vista a la definición de un proceso para la carga y mantenimiento de data warehouses

Presentación día 15 con fecha 17/08/2001

Artículo: **Aplicación de PVM a herramientas de ETL.**
Expositor: **Ignacio Larrañaga**

Índice

Introducción y Motivación

- El objetivo del trabajo es formar experiencias en el trabajo con PVM.
- Además se tratara de interceptar este trabajo de forma que pueda ser interesante para el grupo.
- La motivación es un trabajo presentado en [BBGS] (*Loading Databases Using Dataflow Parallelism*) y [MG2000] (*River Design*).

Trabajos [BBGS] y [MG2000]

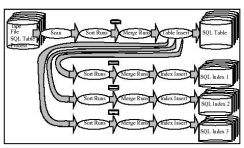
- Se presenta es un prototipo para procesos paralelos de carga basado en el encare de "*flujo de datos"*.
- Encare "*flujo de datos*":
  - La información reside en archivos o bases de datos en algunos discos, cintas u otros dispositivos de memoria.

Encare basado en "*flujo de datos*"

- 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.

Encare basado en "*flujo de datos*" (II)

- 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.



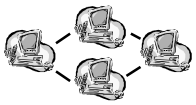
PVM (Parallel Virtual Machine)

- Se trata de una solución para atacar problemas de programación paralela.
- Es un framework de trabajo.
- Esta compuesta por:
  - Maquina Virtual (procesos ejecutando en el conjunto de maquinas involucradas). Acceso a una consola para control.
  - Bibliotecas de programacion para programar nuestras aplicaciones (C/C++, Fortran, Java).

La Herramienta

Figura 1: Ejemplo de proceso.

- Ataca 3 problemas:
  - Definición de proceso de ETL.
  - Ejecución Centralizada.
  - Ejecución Distribuida.

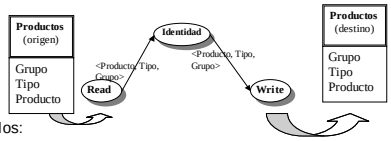


Definición del Proceso

- Que ha que definir?
  - Formatos de los registros (porque hay que saber con que datos se trabaja, o que datos se intercambian)
  - Definición del grafo (porque hay que decir como queremos hacerlo, obvio con la idea de "*flujo de datos*")
    - Nodos
    - Aristas

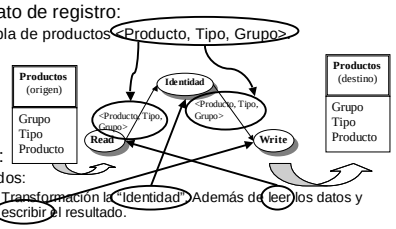
Ejemplo de definición del proceso

- Formato de registro:
  - Tabla de productos <Producto, Tipo, Grupo>.
- Grafo:
  - Nodos:
    - Transformación la "Identidad". Además de leer los datos y escribir el resultado.
  - Aristas:
    - Entre los nodos de lectura e identidad y a su vez luego hacia el nodo de escritura.



Ejemplo de definición del proceso

- Formato de registro:
  - Tabla de productos <Producto, Tipo, Grupo>.
- Grafo:
  - Nodos:
    - Transformación la "Identidad". Además de leer los datos y escribir el resultado.
  - Aritas:
    - Entre los nodos de lectura e identidad y a su vez luego hacia el nodo de escritura.



Como se define el proceso en la herramienta?

- *Dando acceso a una serie de comandos que permiten definir los distintos elementos del proceso.*
  - *define record format*: Define los formatos de registro. 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>})
  - Ejemplo:
    - *define record format Formato1 { Producto, 10, String; Tipo, 4, Integer; Grupo, 4, String};*

Comandos (define nodes)

- *define nodes*: Define los nodos del grafo. Son dados por conjuntos de duplas "*nombre"* y "*operador"* (p.e. {<"Read", Read>, <"Identidad", Identidad>, <"Write", Write>}).
  - Ejemplo:

```
define nodes {
    NodoRead, read operator (W:\data\ventas-origen.txt);
    NodoIdentity, identity operator;
    NodoWrite, write operator (W:\data\ventas-destino.txt)
};
```

Operadores

- 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).

```
define nodes {
    NodoRead, read operator (W:\data\ventas-origen.txt);
    ...
}
```

Operadores disponibles

- READ OPERATOR. Lee la información de un archivo dado y la coloca 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.

Operadores disponibles (II)

- 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 (volveremos sobre este punto posteriormente).

Comandos (define edges)

- *define edges*: Define las aristas del grafo. Están dadas por un conjunto de cuádruplas "*nombre*", "*nodo origen*", "*nodo destino*" y "*formato*" (p.e. {<"AristaRead–Identidad", Read, Identidad, "Formato 1">...}).
  - Ejemplo:

```
define edges {
    AristaRead-Identidad, NodeRead, NodeIdentity, Formato1;
    AristaIdentidad-Write, NodeIdentity, NodeWrite, Formato1
};
```

Ejecución

- Como ya mencionamos tenemos dos posibilidades:
  - Centralizada:
    - Un solo proceso del sistema operativo realizara toda la tarea y obviamente en una sola maquina.
    - Para llevar a cabo la ejecución centralizada basta que una vez definido lo anterior ejecutemos "execute alone;".

Ejecución Distribuida

- Distribuida:
  - Todo es exactamente igual salvo que se ejecuta el comando “execute;”.
  - Internamente:
    - Se levantan N procesos (1 por nodo)
    - Se transmite el sub-grafo de interés
    - Cada nodo comienza a ejecutar el operador correspondiente

Nota: Se puede configurar donde se ejecutan los nodos dentro de todas las maquinas, con opciones “on machine” u “on architecture”

Ejecución distribuida de operadores

- READ OPERATOR: Se abrirá el archivo correspondiente y como se explicara a continuación se enviaran los datos a los nodos consecutivos.
- WRITE OPERATOR: Lo que se valla recibiendo se ira colocando en el archivo que se indico en la definición del operador
- IDENTITY OPERATOR: Simplemente cuando algo llega se reenvía

Ejecución distribuida de operadores

- AGGREGATE OPERATOR: En este caso todo lo que se recibe, es colocado en un vector donde si se constata que el elemento ya existía se le aplica la operación de agregación que se dio en la definición del operador (se vera mas en detalle mas adelante) a los campos que corresponda. Luego de haber terminado de realizar toda la agregación recién se comienza entonces a propagar los datos calculados.

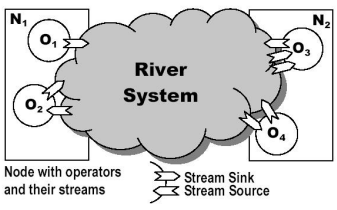
Ejecución Centralizada vs. Distribuida

- Centralizada:
  - El grafo es ejecutado por una sola maquina.
  - NO es necesario distribuir la configuración.
  - La información se transmite siempre por la memoria como parámetros de funciones.
  - Se utiliza un solo hilo de ejecución.
  - NO se realizan particiones del problema.
- Distribuida:
  - Cada nodo del grafo es lanzado en una maquina.
  - Es necesario distribuir la configuración.
  - La información se transmite como mensajes conteniendo los registros (entre maquinas).
  - Se utilizan múltiples hilos de ejecución.
  - Se realizan particiones de dominio y funcionales.
  - Overhead de transferencia.

Intercambio de registros

- Aparecen conceptos de
  - River: Ambiente de comunicación en el cual los procesos intercambian los datos.
  - Sources: Punto de contacto con el River del cual obtienen datos.
    - Implementación: Es un punto de acceso a PVM donde se enmascara la comunicación llamándole “source”. Es decir leer un dato se transforma en pmv_recv() o sea invocar una rutina para recibir los datos.
  - Sinks: Punto de contacto con el River donde se colocan datos.
    - Implementación: Idem a anterior pero llamándole “sinks”. Es decir escribir un dato se transforma en pmv_send() o sea invocar una rutina para enviar los datos.

Intercambio de registros (II)



Implementación de P9

- Especificación en [AM2000]:
  - source schema : R (A_1, ...,A_n) $\in$ Rel_M
  - 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
- Solo se cambio Y tomando los atributos que permanecen en lugar de los que se remueven

Comando para P9

- Lo que se hace es definir que un nodo tiene asociado el operador AGGREGATE OPERATOR y este se define así:
  - 'agregate operator' '(' [{Atributos} / record format '(' {Atributos de RF} ')'] ';' {Atributos a Agregar} ')'
    - '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.

Comando para P9 (II)

- '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.

Esto ultimo esta disponible en la aplicación haciendo 'help define nodes agrgegate operator'

Resumen de comandos

- **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.

Resumen de comandos (II)

- **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.

El Ejemplo

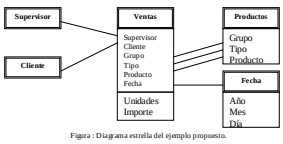


Figura : Diagrama estrella del ejemplo propuesto.

Problema: Como probarlo y que sea interesante y creíble?

Solución: Saliendo a buscar datos y viendo que aparece.

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

Y que apareció ?

- 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.
  - dimensiones: el supervisor del vendedor que realizo la venta, el cliente que la compro, el producto que compro y en que fecha lo hizo.
  - Producto y Fecha además con jerarquías conformadas por grupo, tipo y producto en el primer caso y año, mes y día en el segundo

Como plantear el problema ?

- 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).

Como plantear el problema ? (II)

- Una estrategia:
  - Dadas cuatro variables en juego si las colocamos todas significara que estamos analizando las cuatro disgregadas, es decir sin resumir ninguna de las variables.
  - Esto podría ser un primer objetivo (SCPA) en nuestro problema donde todas estas variables se encuentran sin manipular.

Como plantear el problema ? (III)

- 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.

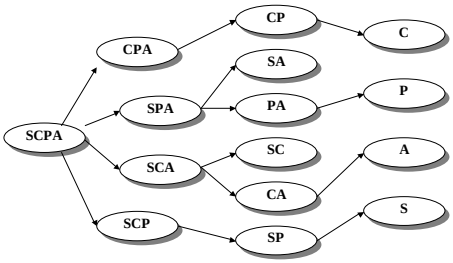
Como organizar el grafo ?

- 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?.

Como organizar el grafo ? (II)

- 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.

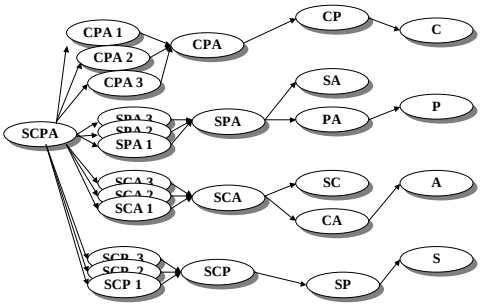
Primera propuesta



Otra posibilidad

- Otra posibilidad que también rápidamente se nos puede venir a la mente es aplicar la técnica de **partición de dominio**.
- 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.

Segunda posibilidad



Líneas de Trabajo

- Nuevos Operadores.
  - Extender la batería de operadores por ejemplo con todas las primitivas.
- Particiones de Dominio Automáticas.
  - Generar la separación y posterior integración de un nodo.
- Ejecución por maquina mono proceso.
  - Ejecutar los conjuntos de nodos en una maquina como un solo proceso.
- Mejoras en las estrategias de comunicación.
  - Analizar la posibilidad de mejora.

Conclusiones

- Se cobraron experiencias en la herramienta PVM.
- Parecería ser interesante para el grupo.
- Hay una serie de puntos con los que se podría seguir.
- Muy simple el probar nuevas estrategias distribuidas para resolver el problema.
- No se pudo generar mediciones de tiempos para esta instancia pero esta en el plan.

Referencias

- [BBGS] Tom Barclay, Robert Barnes, Jim Gray, Prakash Sundaresan, *Loading Databases Using Dataflow Parallelism*.
- [MG2000] Tobias Mayr, Jim Gray, *River Design*.
- [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*
- [AM2000] Adriana Marotta*, Data Warehouse Design and Maintenance through Schema Transformations.*