Bases de Datos a Gran Escala
Master Universitario en Tecnologías de Análisis de da Datos Masivos: Big Data
Escola Técnica Superior de Enxeñaría (ETSE)
Universidade de Santiago de Compostela (USC)

Bases de datos Paralelas

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: http://citius.usc.es/equipo/persoal-adscrito/jrr.viqueira

Curso 2021/2022

Guion

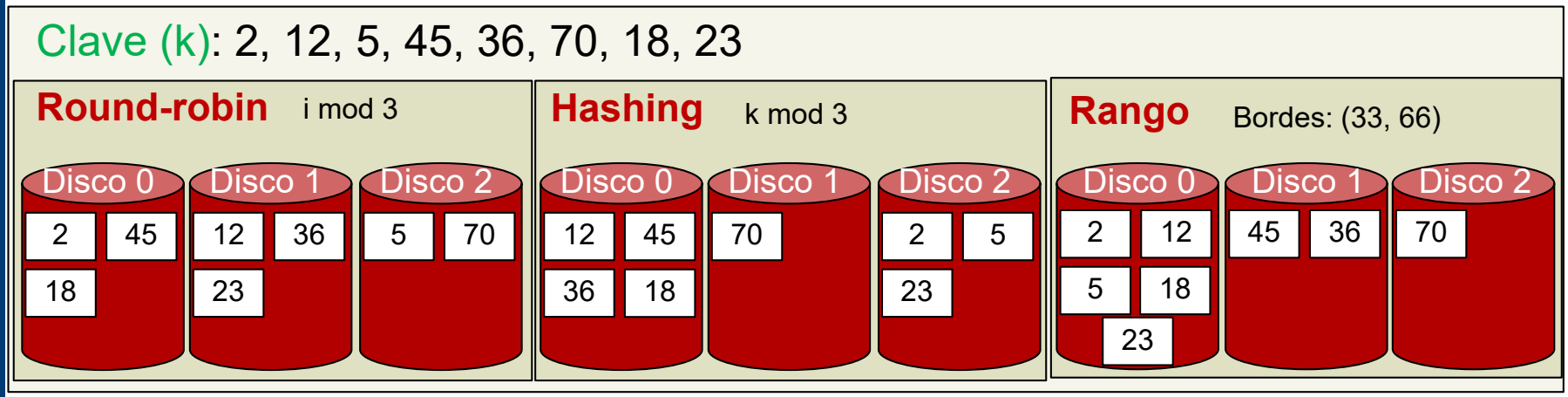
- Introducción
- Paralelismo de Entrada/Salida
- Paralelismo Interquery
- Paralelismo Intraquery
- Paralelismo Intraoperation
- Paralelismo Interoperation
- Optimización
- Diseño de sistemas paralelos
- Paralelismo en sistemas multinúcleo

Introducción

- Motivación
  - Incremento en el volumen de transacciones a procesar
    - On Line Transaction Processing (OLTP)
  - Enormes volúmenes de datos a procesar para el apoyo a la toma de decisiones
    - On Line Analytical Procesing (OLAP)
  - Facilidad para paralelizar operadores de procesamiento de conjuntos (Tablas)
  - Bajada del precio de hardware
  - Irrupción de las arquitecturas multinúcleo
- Medidas de rendimiento
  - Response Time: Tiempo de respuesta ejecución aislada de una consulta
  - Throughput: Número de consultas ejecutadas por unidad de tiempo.

Paralelismo de Entrada/Salida

- Paralelizar el acceso al disco
  - Particionamiento horizontal de las tablas en varios discos
- Estrategias (n discos numerados 0...n-1)
  - Round-robin: Tupla número i se almacena en el disco *i mod n*
  - Particionamiento hash: Aplica función de hash sobre algunos atributo para generar un valor entre 0 y n-1, que determina el disco.
    - Intenta buscar una distribución aleatoria y uniforme de los datos, basada en una clave de búsqueda.
  - Particionamiento por rango: Valores contiguos de una clave (claves) de búsqueda van al mismo disco. Usa un vector de bordes de partición.



Paralelismo de Entrada/Salida

Comparativa de las estrategias

- Tipos de acceso a los datos
  - Escanear una relación completa (scan)
  - Consultas de punto (point query): Condiciones del tipo *Atributo = valor (A = v)*
  - Consultas de rango (range query): Condiciones del tipo *v1 < A < v2*
- Eficiencia en función del tipo de consulta
  - Round-robin Solo es eficiente en scan.
  - Hash
    - Eficiente en point query por la clave de particionamiento.
      - Libera todos los discos menos uno (mejora el "throughput" aunque no mejore el "Response time")
    - Eficiente en scan si la función de hash es buena (distribuye uniformemente)
  - Rango
    - Eficiente en point y range query sobre la clave de particionamiento (mejora el throughput)
    - Sesgo (skew) de ejecución: muchas tuplas en el rango de búsqueda y pocos discos.

Comparativa de las estrategias

- La estrategia de particionamiento afecta a los algoritmos que pueden utilizarse para algunas operaciones
  - Ejemplo: Algoritmos para la operación de Join, y necesidad de mover tuplas entre nodos.
- En general, los sistemas eligen entre Hash o Rango para particionar las tablas
  - Algunos sistemas permiten elegir al usuario y otros ya solo proporcionan una opción.
- Las relaciones pequeñas en general es mejor no particionarlas
  - Tabla con M bloques de disco y sistema con N discos
  - Ideal, dividir en min(M, N) particiones. Nunca más particiones que el número de bloques.

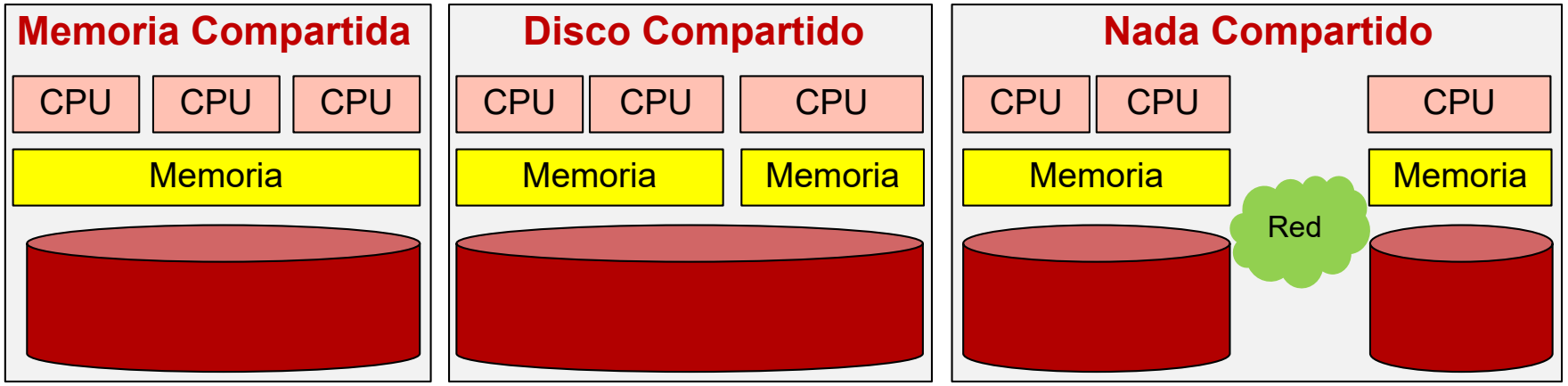
Paralelismo de Entrada/Salida

Tratamiento del sesgo (Skew)

- Muchas tuplas en algunas particiones y muy pocas en otras
  - Los discos con muchas tuplas se convierten en cuellos de botella durante la ejecución.
  - Puede ocurrir cuando no usamos round-robin
- Tipos de sesgo
  - Sesgo de valor de atributo (attribute-value skew): Muchas tuplas con mismo valor
  - Sesgo de particionamiento (partition skew)
    - Particiones mal balanceadas incluso sin attribute-value skew.
    - Más habitual en particionamiento por rango y menos en hashing
    - Impacto en rendimiento se nota mucho más cuando aumenta el paralelismo
- Vector de particionamiento por rango balanceado
  - Puede construirse ordenando los datos por la clave de particionamiento
    - Problema: Sobrecarga de E/S para realizar la ordenación
  - Puede construirse generando y manteniendo un histograma (construcción vía muestreo)

Paralelismo Interquery

- Ejecución en paralelo de varias consultas.
- Puede mejorar el Throughput pero no el Response time.
- Implementación
  - Fácil de implementar en arquitecturas de memoria compartida
  - Más complejo en disco compartido o nada compartido (shared nothing)
    - Coordinar la gestión de concurrencia y el registro de log entre varias máquinas
    - Problema de coherencia de caché
      - Asegurarse que los datos que hay en memoria principal siguen siendo consistentes, si hay otras máquinas que puedan haberlos modificado.

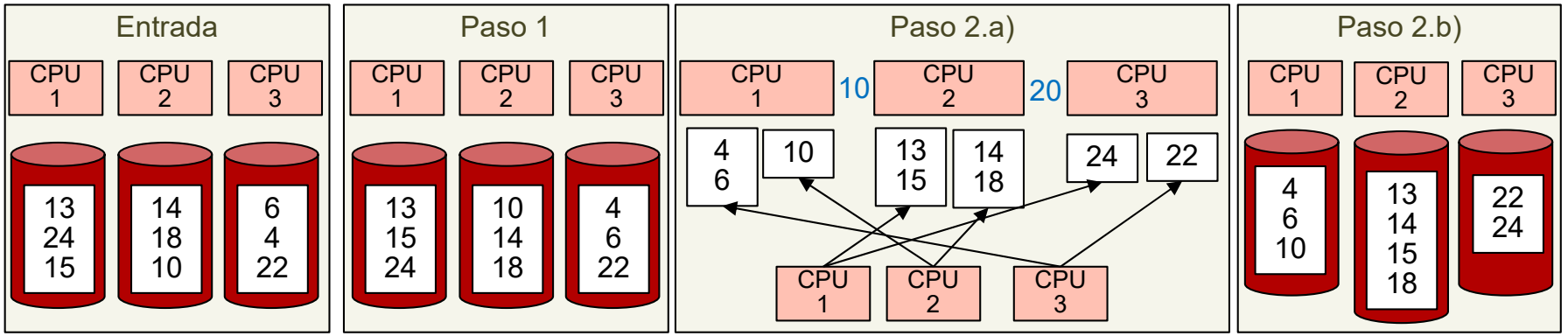


Paralelismo Intraquery

- Ejecución en paralelo de una sola consulta en varios procesadores / discos, etc.
- Importante en consultas con tiempos de ejecución largos
- Se necesita
  - ▷ Ejecutar una operación en paralelo (intraoperation)
    - _ Particionar los datos y ejecutar en paralelo sobre cada partición
    - _ Puede escalar mucho en conjuntos de datos grandes (muchas particiones)
  - ▷ Ejecutar varias operaciones independientes en paralelo (interoperation)
    - _ Uso de pipelines
    - _ Escala poco ya que una consulta no suele tener muchas operaciones

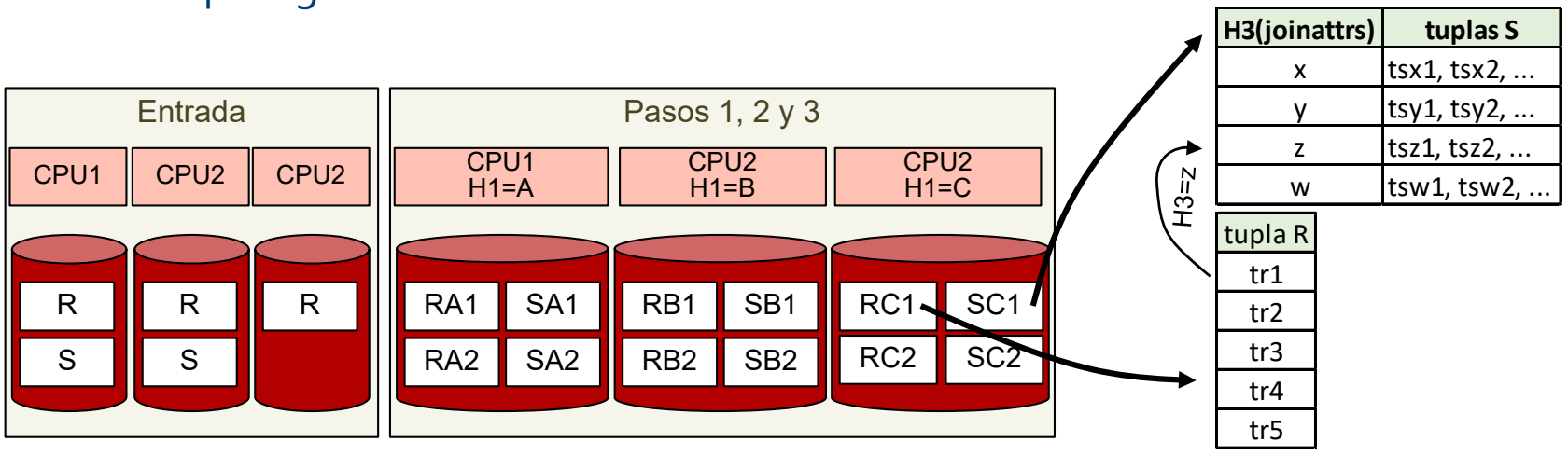
Paralelismo Intraoperation

- Veremos un par de ejemplos
- Ordenación (Parallel External Sort-Merge)
  - ▷ Relación particionada de alguna forma entre los discos
  1. Cada procesador ordena los datos de un determinado disco (partición)
  2. Se mezclan las particiones ordenadas para obtener el resultado ordenado
     a) Reparticionar por rango las particiones ordenadas entre los procesadores. Cada procesador recibe las tuplas de forma ordenada.
     b) Cada procesador mezcla los streams ordenados que recibe de entrada para generar un stream de salida ordenado
     c) El sistema concatena los streams de los procesadores para generar la salida.
  - ▷ Sesgo de ejecución en la recepción de datos en paso 2.b). Solución: enviar primer bloque de cada partición, luego segundo, etc.



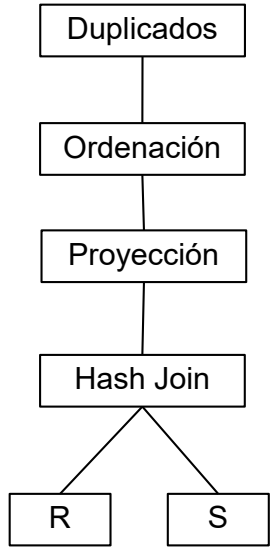
Paralelismo Intraoperation

- Join (Partitioned Parallel Hash Join)
  - ▷ Relaciones R y S (S más pequeña que R) particionadas de alguna forma entre los discos
  1. Utilizar una función hash H1 para distribuir la relación S entre los procesadores.
  2. Utilizar una segunda función hash H2 para particionar S dentro del disco de cada procesador. Cada partición de S debe de caber en memoria.
  3. Utilizar la función hash H1 para distribuir la relación R entre los procesadores, y la función H2 para particionar R dentro de cada procesador.
  4. En cada procesador, para cada número de partición de H2 se crea in índice hash (usando una tercera función hash H3) en memoria para la partición de S, y para cada tupla de R en la misma partición se buscan tuplas de S en el índice para generar tulas del resultado.



Paralelismo Interoperation

Evaluación de expresiones

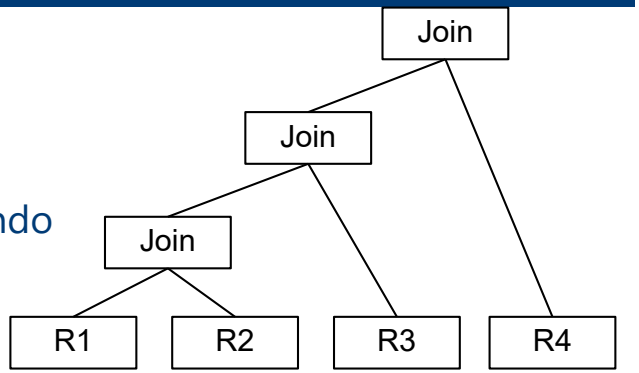


- Materialización
  - El resultado del operador se almacena en una tabla temporal
  - Hasta que termina la ejecución de una operación no se puede iniciar la siguiente
- Pipelining (tuberías)
  - Cada tupla generada por un operador se envía al siguiente operador antes de empezar a procesar la siguiente tupla
  - Si todo el árbol se evalúa con pipelining, se puden generar resultados en la salida muy pronto
  - **Implementación**
    - Guiado por demanda
      - Cada operador intenta generar la siguiente tupla solo cuando su operador cliente se la solicita con una llamada tipo next()
    - Guiado por el productor
      - Antes de que se las soliciten, cada operador pide las tuplas que necesite en su entrada y empieza a generar salidas en un buffer.
      - Permite que cada operador se ejecute en un hilo independiente de forma paralela

Paralelismo Interoperation

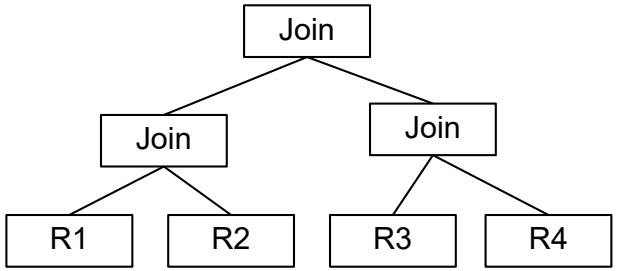
Paralelismo de pipelining

- Ejemplo: Join de 4 tablas
  - Mientras el primer join procesa una tupla, el segundo puede estar procesando un resultado anterior.
  - Útil cuando tenemos pocos procesadores, ya que una consulta no va a tener un gran número de operadores que puedan combinarse con pipelining
  - No sirve para operadores que requieren materialización (Ordenación por ejemplo)
  - Si el coste de un operador es mucho mayor que los demás, éste se convierte en un cuello de botella y el beneficio no es grande.
  - Más importante el no tener que escribir resultados intermedios que lo que aporta en paralelización



Paralelismo independiente

- Dos operaciones independientes se pueden ejecutar en paralelo
- Mismo Join de 4 tablas
- También útil cuando hay pocos procesadores



Optimización

- Optimizador: Componente fundamental en un SGBDs relacional
  - Busca el mejor plan de ejecución para cada consulta
- Con la paralelización el trabajo del optimizador se hace mucho más complejo
  - Muchas más combinaciones para hacer planes distintos.
  - Como paralelizar cada operación (Cuantos procesadores utilizar)
  - Como ejecutar cada operación (Pipelining, independientes, secuenciales)
- ¿Cómo definir los recursos de cada tipo que debería usar cada operación?
  - Algunas operaciones es mejor no ejecutarlas en paralelo
- Evitar pipelines largos (las últimas operaciones pueden tener que esperar mucho, y consumen recursos)
- Uso de heurísticas para no tener que generar todos los planes posibles
  - Ejemplo de Teradata: No usa pipelining paralelo y las operaciones se paralelizan con todos los procesadores
  - Optimizar en secuencial y después paralelizar las operaciones.

Diseño de sistemas paralelos

- Disponibilidad: Cuestiones a tener en cuenta
  - Tolerancia a fallos de algunos procesadores o discos
  - Reorganizaciones de datos en tiempo de ejecución y cambios de esquema
- Probabilidad de fallo de un sistema paralelo es mayor
  - Un procesador, un fallo cada 5 años. 100 procesadores fallarían cada 18 días
  - Replicar datos en al menos 2 procesadores
- Modificaciones del esquema pueden tardar mucho con grandes volumnes de datos (añadir atributos, crear índices, etc.)
  - Se deben de poder ejecutar on-line (sin parar la ejecución de transacciones)
- Ejemplos de sistemas:
  - Netezza (IBM)
  - DATAllegro (Microsoft)
  - GreenPlum (Open Source)
  - Aster Data
  - CITUS Data (prácticas)

Paralelismo en sistemas multinúcleo

- Todos los SGBDs actuales se ejecutan en una plataforma paralela
- Paralelismo vs Velocidad cruda
  - Progreso exponencial de la velocidad de los procesadores
  - Eficiencia energética: Alta velocidad basada en más consumo energético
    - Duración de la batería, consumo eléctrico, disipación de calor, etc.
  - Solución: colocar varios procesadores (núcleos) en cada chip
    - Procesador multinúcleo
- Memoria caché y procesos multihilo
  - Memoria principal pasa a ser un cuello de botella
    - Solución: Incluir varios niveles de memoria cache (L1, L2, etc.)
  - Necesidad de extender la jerarquía de memoria asumida por el SGBDs
    - SGBDs controla tráfico entre memoria y disco. Tráfico entre cachés asumido por hardware
  - Utilizar varios hilos para minimizar impacto de los fallos de caché
    - Si un hilo falla en el acceso a caché, otro hilo asume el control mientras se cargan los datos del primero (como en la paralelización de la entrada/salida a disco).

Paralelismo en sistemas multinúcleo

- Adaptando el diseño de los SGBDs para arquitecturas modernas
  - Uso eficiente de arquitecturas modernas (multinúcleo) es un reto
  - Cantidad de datos necesarios en cache aumenta con el número de hilos
    - Aumentan los fallos de cache
      - Incluso un núcleo con varios hilos en ejecución puede tener que esperar por el acceso a memoria principal
  - Control de concurrencia
    - Restricciones en el acceso a los datos en concurrencia
      - Esperas por datos o retrocesos de transacciones potencialmente problemáticas
    - Para disminuir los conflictos, puede ser necesario aumentar los datos en caché (más fallos de cache)
  - Componentes del SGBDs compartidos por las transacciones
    - Gestor de bloqueos, gestor del buffer, gestor del registro histórico, etc.
    - Todos son potenciales cuellos de botella
  - Muchas transacciones pueden no aprovechar bien el hardware
  - Área activa de investigación

Bases de datos Paralelas

Capítulo 18: Bases de datos paralelas. A. Silberschatz, H.F. Korth, S. Sudarshan, Database System Concepts, 6th Edition, McGraw-Hill, 2014

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: http://citius.usc.es/equipo/persoal-adscrito/jrr.viqueira

Curso 2021/2022