(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 776 975**

(51) Int. CI.:

G06F 16/27 (2009.01)
G06F 16/182 (2009.01)
G06F 16/2458 (2009.01)
G06F 16/20 (2009.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **23.08.2010 PCT/EP2010/062223**

(87) Fecha y número de publicación internacional: **03.03.2011 WO11023652**

(96) Fecha de presentación y número de la solicitud europea: **23.08.2010 E 10771670 (6)**

(97) Fecha y número de publicación de la concesión europea: **01.01.2020 EP 2471013**

(54) Título: **Tabla de almacenamiento de datos de escaneado completo continuo y almacén de datos distribuido con tiempo de respuesta predecible para carga de trabajo impredecible**

(30) Prioridad:

24.08.2009 EP 09305780

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
03.08.2020

(73) Titular/es:

AMADEUS S.A.S. (100.0%)
485 Route du Pin Montard, Sophia Antipolis
06410 Biot, FR

(72) Inventor/es:

FAUSER, DIETMAR;
MEYER, JEREMY;
FLORIMOND, CÉDRIC;
KOSSMANN, DONALD;
ALONSO, GUSTAVO;
GIANNIKIS, GEORGIOS y
UNTERBRUNNER, PHILIPP

(74) Agente/Representante:

SUGRAÑES MOLINÉ, Pedro

DESCRIPCIÓN

Tabla de almacenamiento de datos de escaneado completo continuo y almacén de datos distribuido con tiempo de respuesta predecible para carga de trabajo impredecible

Campo de la invención

La presente invención se refiere en general a almacenes de datos y sistemas de gestión de datos manejados en un entorno informático distribuido. De manera más específica, la presente invención se refiere a almacenes de datos que comprenden un motor de escaneado del almacenamiento funcionando en una memoria principal de nodos de almacenamiento múltiple conteniendo cada uno un segmento de una única, posiblemente muy grande, tabla relacional y capaz de manejar conjuntamente grandes números de consultas y actualizaciones concurrentes con garantía de latencia de acceso y frescura de datos.

Antecedentes de la invención

En la última década los requisitos a los que se enfrentan las aplicaciones de bases de datos relacionales tradicionales han cambiado significativamente. Más importante, las bases de datos deben funcionar con rendimiento previsible y bajo coste de administración. Además, las bases de datos deben ser capaces de manejar cargas de trabajo diversas, en evolución dado que las aplicaciones se extienden constantemente con nuevas funcionalidades y se desarrollan nuevos servicios de datos, añadiendo de ese modo nuevos tipos de consultas a la carga de trabajo de una forma impredecible. Sobre todo, estos nuevos requisitos se han expresado en el contexto de plataformas comerciales bien conocidas con una presencia mundial tales como eBay, Amazon, Salesforce, etc. Esta última, por ejemplo, permite a los usuarios personalizar su aplicación y definir sus propias consultas. Proporcionar dicha plataforma implica cargas de trabajo de consultas altamente diversificadas; aún más, los usuarios de la plataforma esperan un tiempo de respuesta constante. Desafortunadamente, es difícil proporcionar la garantía de productividad y latencia con sistemas de bases de datos tradicionales. Estos sistemas se diseñan para conseguir un mejor rendimiento para cada consulta individual. Con este fin, se basan en optimizadores de consulta sofisticados y administradores habilitados para seleccionar los índices correctos y vistas materializadas. Dichos sistemas complejos son caros de mantener y no presentan un rendimiento predecible para cargas de trabajo impredecibles, en evolución.

El documento US 5.873.075 describe la detección automática y prevención de violaciones en tablas mutantes de la integridad de la base de datos en una consulta SQL antes de la generación y selección de un plan de ejecución de consulta (QEP) óptimo que incluye la modificación del modelo de grafo de consulta (QGM).

Es por lo tanto un objeto de la invención superar las limitaciones anteriores en la divulgación de una tabla relacional distribuida, escalable y un motor de escaneado completo del almacenamiento capaz de sostener grandes números de consultas y actualizaciones diversas con latencia de acceso y frescura de datos garantizadas independientemente de los tipos de carga de trabajo y consultas con los que haya de tratar.

Objetos, características y ventajas adicionales de la presente invención serán evidentes para los expertos en la materia tras el examen de la descripción que sigue con referencia a los dibujos adjuntos. Se pretende que cualesquiera ventajas adicionales se incorporen en el presente documento.

Sumario de la invención

La invención se define mediante las reivindicaciones independientes.

Los objetos anteriores se satisfacen por la invención que describe un método para almacenar y recuperar datos en un nodo de almacenamiento de un almacén de datos que comprende la etapa de almacenar al menos un segmento de una tabla relacional en un medio de almacenamiento de datos del nodo de almacenamiento, el medio de almacenamiento de datos comprende al menos un núcleo de procesador y su memoria principal asociada. El método comprende las siguientes etapas:

almacenar en cada memoria principal asociada de un núcleo de procesador un único segmento asociado;

ejecutar desde el núcleo de procesador un hilo de escaneado dedicado al escaneado de dicho segmento asociado;

escanear continua y completamente dicho segmento asociado;

recibir y procesar lotes de consulta y operaciones de actualización;

indexar operaciones de consulta y actualización de un lote al comienzo de cada escaneado de datos;

operaciones de junta de consulta y actualización de un lote para registros de datos recuperados de dicho segmento asociado de la tabla relacional que coincidan con los predicados de las operaciones de consulta indexada y

actualización;

completar progresivamente las operaciones de consulta y actualización de un lote siempre que los datos de los registros de datos juntados se recuperen mediante el hilo de escaneado mientras se escanea en dicho segmento asociado de la tabla relacional; de ese modo, maximiza la compartición y acceso a los registros de datos en la memoria principal.

Ventajosamente, es la acción de junta de los predicados de consulta indexados con los registros de datos la que selecciona qué registros de datos se recuperan. Lo que se indexa es el conjunto de predicados de las consultas y operaciones de actualización.

El método de acuerdo con la invención puede incluir también cualquiera de las siguientes características opcionales:

- Cada hilo de escaneado ejecuta un algoritmo de Escaneado de Reloj que comprende:

  un cursor de escritura y un cursor de lectura exhaustivamente dispuestos para escanear dicho segmento de la tabla relacional para procesar, en cada ciclo de escaneado, respectivamente, todas las operaciones de consulta y actualización pendientes de un lote;

  en el que los cursores de escritura y lectura permiten que se escriban registros de datos del segmento de tabla relacional antes de que el cursor de lectura pueda leerlos;

  en el que todas las operaciones de consulta se activan al comienzo de un ciclo de escaneado;

  en el que todas las operaciones de actualización o bien se aplican totalmente o no se aplican en absoluto a los registros de datos del segmento de tabla relacional; de ese modo, proporcionar consistencia del segmento de la tabla relacional.

- Las operaciones de actualización incluyen la modificación, borrado e inserción de registros de datos desde/dentro del segmento.

Las operaciones se marcan con el tiempo y se mantienen procesadas en orden total por el hilo de escaneado.

Construcción de estadísticas internas, indexado de consultas y agregación de consultas.

Mientras escanea el segmento de tabla relacional, se recogen estadísticas internas por los hilos de escaneado en el que cada uno cuenta un número de registros, un número de valores nulos y un número de valores distintos actualmente hallados para un atributo dado del segmento. Las estadísticas internas son herramientas de Escaneado de Reloj usadas para dirigir la construcción de los índices de consulta.

Para el manejo de la agregación de consultas de usuario, los cálculos que incluyen: suma de valores de atributos, hallazgo de un valor mínimo de atributo, hallazgo de un valor máximo de atributo y recuento del número de valores no nulos, se realizan adicionalmente mediante la capa de agregación en cooperación con los nodos de almacenamiento que contienen los segmentos relevantes.

Por lo tanto, el escaneo completo permite tanto la construcción de estadísticas internas para consultas de indexado y consultas de rellenado que necesitan la agregación de registros de datos o valores de registros de datos.

Durabilidad

- El hilo de escaneado construye una instantánea de los registros de datos mientras escanea el segmento de tabla relacional y en el que la instantánea de los registros de datos se almacena cuando se completan en una memoria permanente. Si N es el número de ranuras en la memoria principal entonces se almacenan N+1 ranuras de segmento en la memoria permanente. Por lo tanto, si la máquina principal se bloquea, es solo el segmento actual, es decir, el que se está escribiendo, el que se pierde y quedan aún disponibles N segmentos válidos. El punto de comprobación se realiza segmento por segmento. Esto se denomina como punto de comprobación difuso.
- Se mantiene un archivo de registro lógico de las transacciones actuales por el hilo de escaneado mientras se escanea el segmento de tabla relacional.
- Ventajosamente, la durabilidad de la memoria principal se obtiene mediante la recarga de la última instantánea completa y la reproducción del archivo de registro de las transacciones actuales.
- La memoria principal se compone de bancos de memoria integrados no duraderos estrechamente vinculados a las unidades de procesamiento de un nodo de almacenamiento.
- El segmento de tabla relacional está compuesto totalmente y permanece residente en la memoria principal.
- Adicionalmente, el hilo de escaneado es forzado a ejecutarse en un núcleo de procesador designado y recoge datos del segmento de tabla relacional en fragmentos de tamaño compatible con uno o más niveles de almacenes de memoria caché dedicados al núcleo de procesador designado para impedir pérdidas de almacén intermedio.

Otra materia objeto de la invención es un nodo de almacenamiento de un almacén de datos y sistema de gestión de datos, que comprende un medio de almacenamiento de datos que tiene una memoria principal y dispuesto para almacenar en la memoria principal una pluralidad de segmentos de una tabla relacional, caracterizado por que el nodo de almacenamiento comprende al menos un núcleo de procesador asociado a cada segmento y configurado para ejecutar al menos un hilo de escaneado dedicado al escaneado de dicho segmento asociado, estando dispuesto el nodo de almacenamiento de modo que:

cada hilo de escaneado escanea únicamente, continua y totalmente el segmento dedicado;

para cada segmento, el nodo de almacenamiento recibe y procesa lotes de operaciones de consulta y actualización;

las operaciones de consulta y actualización de un lote se reindexan al comienzo de cada escaneado por el hilo de escaneado;

las operaciones de consulta indexada y actualización de un lote se juntan independientemente a registros de datos del segmento de tabla relacional que coinciden con los predicados de las operaciones de consulta indexada y actualización y se recuperan al menos algunos datos de los registros de datos para los que se juntan las operaciones de consulta y actualización,

las operaciones de consulta indexada y actualización del lote se completan progresivamente siempre que los datos de los registros de datos juntados se recuperen por el hilo de escaneado mientras escanea el segmento de tabla relacional;

de ese modo, maximiza la compartición y acceso a los registros de datos en la memoria principal.

Otra materia objeto de la invención es un almacén de datos distribuidos que comprende:

una capa de almacenamiento compuesta por una pluralidad de nodos de almacenamiento de acuerdo con cualquiera de las características previas;

una capa de agregación compuesta de nodos de agregador indicadas para encaminar lotes de operaciones de consulta y actualización a los nodos de almacenamiento y mezclar los resultados devueltos por los nodos de almacenamiento.

Opcionalmente, el almacén de datos de acuerdo con la invención puede incluir también una cualquiera de las siguientes características:

- La capa de almacenamiento se organiza en grupos de nodos de almacenamiento. Cada nodo de almacenamiento se almacena como tantos segmentos como su número de núcleos computacionales. El conjunto de todos los segmentos mantenidos por un nodo de almacenamiento constituye un conjunto de segmento. Todos los nodos de almacenamiento de un grupo almacenan conjuntos de segmento idénticos. - cada nodo de almacenamiento de un grupo es una réplica de los otros nodos de almacenamiento del mismo grupo, esto es un almacén de datos replicado distribuido.
- De acuerdo con una realización específica, un nodo de almacenamiento de un grupo es un nodo maestro y los otros nodos de almacenamiento son nodos de réplica, disponiéndose el nodo maestro de un grupo de réplica para agrupar y dirigir las consultas entrantes a diferentes nodos de réplica.
- Ventajosamente, el almacén de datos distribuido se dispone para extraer valores de los registros mientras escanea cada segmento; agregar estos valores; realizar al menos uno cualquiera de los siguientes cálculos: sumar valores de atributos, hallar un valor mínimo de atributo y hallar un valor máximo de atributo, contar el número de valores no nulos; completar una consulta proporcionado los resultados del cálculo realizado en dichos valores agregados.
- Los diferentes grupos de replicación almacenan diferentes conjuntos de segmentos de la tabla relacional.
- Los grupos de replicación se especializan para manejar operaciones de consulta específicas.
- La capa de agregación se dispone para encaminar solamente lotes específicos de operaciones de consulta y actualización a nodos de almacenamiento especializados correspondientes.
- El almacén de datos distribuido se configura de modo que los nodos de almacenamiento especializados se organizan para acelerar el procesamiento de predicados de tipo igualdad y rango.

De acuerdo con otro aspecto, la invención se refiere a un sistema de escaneado en un entorno distribuido informatizado para escanear exclusivamente al menos un segmento de memoria de una tabla relacional de base de datos durante ciclos de escaneado sucesivos continuos, comprendiendo el sistema de escaneado:

un primer y segundo cursor adaptados para escanear exhaustivamente el al menos un segmento de memoria en cada ciclo de escaneado y para procesar las operaciones de base de datos de actualización y consulta pendientes,

en el que el primer cursor es un cursor de escritura configurado para escribir en el al menos un segmento de memoria y procesar operaciones de base de datos dirigidas a actualizar registros de datos en la tabla relacional,

en el que el segundo cursor es un cursor de lectura configurado para leer desde el al menos un segmento de memoria registros de datos de tabla relacional en respuesta a la consulta,

caracterizado el sistema de escaneado porque el primer cursor se configura además para escribir registros de datos en la tabla relacional dentro de al menos un segmento de memoria antes de que el segundo cursor pueda leerlos y porque el retardo de tiempo entre la escritura del primer cursor y la lectura del segundo cursor es menor que un ciclo de escaneado.

En realizaciones funcionales que pueden implementarse en asociación o alternativamente, dicho sistema de escaneado es tal que:

- las operaciones de consulta de base de datos se reorganizan para realizar juntas de consulta-datos;
- las operaciones de actualización de base de datos incluyen la actualización, borrado e inserción de registros de datos de tabla relacional dentro del al menos un segmento de memoria;
- las operaciones de actualización de la base de datos se procesan en su orden de llegada;
- las operaciones de base de datos de consulta y actualización se reorganizan para realizar juntas de consulta/actualización-datos;
- se configura para procesar al menos una consulta y una actualización durante un mismo ciclo.
- se configura para indexar las operaciones al comienzo de un ciclo;
- dichos cursores son cursores lógicos integrados en un único cursor físico;
- el tiempo requerido para procesar una operación de base de datos es predecible e independiente de los tipos de operación y de la carga de trabajo.

La invención se refiere también a un sistema para procesamiento de datos que comprende al menos una CPU, una base de datos y el sistema de escaneado como se ha presentado anteriormente. Este sistema puede comprender al menos una de las siguientes características opcionales:

- comprende funciones de reparto y mezcla y en el que una operación inicial se configura para ser de reparto en un número de suboperaciones;
- el número de suboperaciones depende del rendimiento y del número de las CPU y memorias principales;
- el sistema de escaneado se configura para escanear dichas suboperaciones independientemente y para producir la salida de un sub-resultado para cada uno de dichos escaneados realizados;
- los sub-resultados producidos para cada uno de dichos escaneados realizados se mezclan conjuntamente para proporcionar una salida de resultado para las operaciones iniciales.

Finalmente, la invención divulga un método de escaneado en un entorno distribuido informatizado para escanear exclusivamente al menos un segmento de memoria de una tabla relacional de base de datos durante ciclos de escaneado sucesivos continuos, comprendiendo el sistema de escaneado las etapas de:

escanear exhaustivamente, con un primer y un segundo cursores adaptados, al menos un segmento de memoria en cada ciclo de escaneado y para procesar las operaciones de base de datos de actualización y consulta pendientes,
en el que el primer cursor es un cursor de escritura configurado para escribir en el al menos un segmento de memoria y procesar operaciones de base de datos dirigidas a actualizar registros de datos en la tabla relacional,
en el que el segundo cursor es un cursor de lectura configurado para leer desde el al menos un segmento de memoria registros de datos de tabla relacional en respuesta a la consulta,
caracterizado el método de escaneado porque el primer cursor se configura además para escribir registros de datos en la tabla relacional dentro de al menos un segmento de memoria antes de que el segundo cursor pueda leerlos y porque el retardo de tiempo entre la escritura del primer cursor y la lectura del segundo cursor es menor que un ciclo de escaneado.

Breve descripción de los dibujos

La FIGURA 1 ilustra y explica adicionalmente el objetivo de la invención.
La FIGURA 2 muestra una visión general del entorno distribuido informatizado en el que opera un almacén de datos de la invención.
La FIGURA 3 visualiza la arquitectura de un nodo de almacenamiento de acuerdo con la invención.
La FIGURA 4 analiza el algoritmo que escanea continuamente los datos en un hilo de control separado y compara el algoritmo de escaneado de la invención (Figura 4b) con los algoritmos de la técnica anterior (Figura 4a).
La FIGURA 5 analiza dos tipos de algoritmos de junta, concretamente: "Índice Unión Junta" e "Índice Unión Actualización Junta" y muestra un ejemplo de cómo se indexan las consultas.
La FIGURA 6 analiza el optimizador multiconsulta dirigido a la planificación de juntas descritas en la figura previa y muestra adicionalmente cómo se construyen los índices de consulta.
La FIGURA 7 analiza la durabilidad de un almacén de datos de acuerdo con la invención.
La FIGURA 8 muestra resultados experimentales.

Descripción detallada

La siguiente descripción detallada de la invención se refiere a los dibujos adjuntos. Aunque la descripción incluye realizaciones de ejemplo, son posibles otras realizaciones y pueden realizarse cambios a las realizaciones descritas sin apartarse del alcance de la invención.

Se proporciona a continuación un glosario de los términos y referencias usados para describir la invención.

- ACID: En la ciencia informática, ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) es un conjunto de propiedades que garantizan que las transacciones de base de datos se procesan fiablemente. En el contexto particular de la invención una transacción solo se compone de una operación y no muchas como sería posible con otros sistemas de almacenamiento de datos. Esto incluye todas las operaciones de consulta y actualización que se envían por los usuarios finales. La Atomicidad se refiere a la capacidad del DBMS (Sistema de gestión de base de datos) para garantizar que o bien se realizan todas las tareas de la transacción o bien ninguna de ellas. La atomicidad establece que las modificaciones de la base de datos deben seguir una regla de "todo o nada". Cada transacción se dice que es "atómica" si cuando falla una parte de la transacción, falla la transacción completa. Es crítico que el sistema de gestión de la base de datos mantenga la naturaleza atómica de las transacciones a pesar de cualquier DBMS, sistema operativo o fallo de hardware. La propiedad de Consistencia asegura que la base de datos permanece en un estado consistente antes del comienzo de la transacción y después de que finalice la transacción, tanto si tiene éxito como si no. En un sistema de almacenamiento de datos de acuerdo con la invención la consistencia solo se garantiza a nivel de segmento. En general, la consistencia establece que solo se escribirá en la base de datos un dato válido. Si, por algún motivo, se ejecuta una transacción que viola las reglas de consistencia de la base de datos, se deshará toda la transacción y se restaurará la base de datos al estado consistente con esas reglas. Por otra parte, si se ejecuta una transacción con éxito, llevará a la base de datos desde un estado que es consistente con las reglas a otro estado que es también consistente con las reglas. El Aislamiento se refiere al requisito de que otras operaciones no pueden acceder o ver los datos en un estado intermedio durante una transacción. Esta restricción se requiere para mantener el rendimiento así como la consistencia entre transacciones en un DBMS. Por lo tanto, cada transacción desconoce otras transacciones en ejecución simultáneamente en el sistema. En un almacén de datos de acuerdo con la invención esto se garantiza automáticamente dado que hay, como se ha mencionado anteriormente, solo una operación ejecutada por transacción. La Durabilidad se refiere a la garantía de que una vez que el usuario ha sido notificado del éxito, la transacción permanecerá y no se deshará. Esto significa que sobrevivirá a un fallo del sistema y que el sistema de base de datos ha comprobado las restricciones de integridad y no necesitará abortar la transacción. Muchas bases de datos implementan la durabilidad mediante la escritura de todas las transacciones en un registro de transacción que puede reproducirse para recrear el estado del sistema justamente antes de un fallo. Una transacción solo puede considerarse realizada después de que esté con seguridad en el registro. La durabilidad no implica un estado permanente de la base de datos. Otra transacción puede sobrescribir cualquier cambio realizado por la transacción actual sin perjudicar a la durabilidad.
- Agregador: Nodo de enrutado del sistema de acuerdo con la invención que no aloja datos pero es capaz de despachar consultas y operaciones sobre nodos de almacenamiento y de realizar procesamiento de orden más elevado sobre los datos antes de devolverlos al cliente.
- Localidad del almacén intermedio: La localidad de referencia, también llamada principio de localidad, es el fenómeno por el que se accede frecuentemente al mismo valor o a localizaciones de almacenamiento relacionadas. Hay dos tipos básicos de localidad de referencia. La localidad temporal se refiere a la reutilización de datos específicos y/o recursos dentro de duraciones de tiempo relativamente pequeñas. La localidad espacial se refiere al uso de elementos de datos dentro de localizaciones de almacenamiento relativamente próximas. Localidad temporal: si en un instante se hace referencia a una localización de memoria particular, entonces es probable que se haga referencia de nuevo a la misma localización en un futuro próximo. Hay una proximidad temporal entre las referencias adyacentes a la misma localización de memoria. En este caso es ventajoso realizar esfuerzos para almacenar una copia de los datos referenciados en un almacenamiento de memoria especial, al que pueda accederse más rápidamente. La localidad temporal es un caso muy especial de la localidad espacial, concretamente cuando la localización prospectiva es idéntica a la localización actual. Localidad espacial: si se hace referencia a una localización de memoria particular en un instante particular, entonces es probable que se haga

referencia a localizaciones de memoria cercanas en el próximo futuro. Hay una proximidad espacial entre las localizaciones de la memoria, referenciadas casi al mismo tiempo. En este caso es ventajoso realizar esfuerzos para estimar, cómo de grande es el tamaño de la vecindad alrededor de la referencia actual que merece la pena preparar para un acceso más rápido. La memoria jerárquica es una optimización de hardware que adquiere los beneficios de la localidad espacial y temporal y puede usarse en varios niveles de la jerarquía de memoria. Una memoria caché es un ejemplo simple de aprovechamiento de la localidad temporal, debido a que es un área de memoria especialmente diseñada más rápida pero más pequeña, usada en general para mantener datos recientemente referenciados y datos cercanos a los datos recientemente referenciados, lo que puede conducir a incrementos potenciales del rendimiento.

- Escaneado de Reloj: El nombre del algoritmo principal que se comporta por el escaneado cíclico de los datos, como si los datos fueran distribuidos en un reloj. Hay dos cursores de escaneado, un cursor de lectura y uno de escritura, que representan las manecillas del reloj.
- Registro de datos: Una fila de una tabla relacional, o una fila de una relación, que contiene posiblemente un valor definido para cada columna de la tabla. El valor es NULO si no se ha realizado una entrada en una columna de esa fila. Cualquier comparación que implique un valor NULO devuelve un estado DESCONOCIDO. Véase la definición de "Predicado" posteriormente. Se hace referencia también a un registro de datos, en la descripción que sigue, como una tupla, es decir, es decir un conjunto de n valores en el que n es el número de columnas definidas de la tabla relacional correspondiente. Algunos de estos valores de tupla son posiblemente valores nulos. También, en la descripción de la invención que sigue se dice que un apartado de columna es un atributo de la fila de tabla correspondiente o registro de datos.
- Algoritmo ambicioso: Un algoritmo ambicioso es cualquier algoritmo que siga una técnica heurística de resolución de problemas para realizar la elección localmente óptima en cada etapa con la esperanza de hallar el óptimo global.
- Afinidad de procesador fuerte: Lleva a cabo la idea de, por ejemplo: "ejecutar siempre este proceso en un procesador" o "ejecutar estos procesos en todos los procesadores excepto el procesador cero". El planificador obedece entonces a la orden y ejecuta procesos solamente en los procesadores permitidos. Hay dos tipos de afinidad de CPU (unidad de procesamiento central). La primera, afinidad suave, también llamada afinidad natural, es la tendencia de un planificador para tratar de mantener los procesos en la misma CPU siempre que sea posible. Este es meramente un intento; si es finalmente no factible, los procesos migrarán ciertamente a otro procesador. Este comportamiento da como resultado un efecto de ping-pong. El planificador hace saltar los procesos entre múltiples procesadores cada vez que se planifican y replanifican. Afinidad fuerte, por otra parte, es lo que una llamada al sistema de afinidad de CPU proporciona. Es un requisito y los procesos deben adherirse a una afinidad fuerte especificada. Si un procesador está unido a la CPU cero, por ejemplo, entonces puede ejecutarse solamente en la CPU cero.
- Particionado horizontal, vertical: Una partición es una división de una base de datos lógica o sus elementos constituyentes en distintas partes independientes. El particionado de una base de datos se realiza normalmente por razones de manejabilidad, rendimiento o disponibilidad. También, permite el escalado del almacén de datos. El particionado horizontal implica poner diferentes filas en diferentes tablas. El particionado vertical implica la creación de tablas con menores columnas y el uso de tablas adicionales para almacenar las columnas restantes. Este tipo de particionado se llama "división de fila", es decir, la fila es dividida por sus columnas.
- Vista materializada: En el sistema de gestión de base de datos que sigue el modelo relacional, una vista es una tabla virtual que representa el resultado de una consulta de base de datos. Siempre que una tabla de vista ordinaria es consultada o actualizada, el sistema de DBMS convierte estas en consultas o actualizaciones contra las tablas de la base subyacente. Una vista materializada lleva a un planteamiento diferente en el que el resultado de la consulta es capturado como una tabla concreta que puede actualizarse a partir de las tablas de la base original de vez en cuando. Esto permite un acceso mucho más eficiente, con el coste de que algunos datos pueden estar potencialmente desactualizados. Es más útil en escenarios de almacenista de datos, en donde consultas frecuentes de las tablas de base de reales pueden ser extremadamente caras. Además, debido a que la vista se manifiesta como una tabla real, cualquier cosa que pueda hacerse con una tabla real puede hacerse con ella, muy importantemente construir índices sobre cualquier columna, permitiendo aceleraciones drásticas en el tiempo de consulta. En una vista normal, es posible solo típicamente aprovechar índices sobre columnas que proceden directamente desde, o están mapeadas a, columnas indexadas en las tablas de la base; frecuentemente esta funcionalidad no se ofrece en absoluto.
- NUMA: (Acceso a Memoria No Uniforme) Una arquitectura multiprocesamiento en el que la memoria se separa en bancos próximos y distantes. En máquinas con arquitectura NUMA y SMP (Procesamiento de Memoria Compartido), todas las múltiples CPU tienen acceso al contenido completo de la memoria que equipa la máquina. Sin embargo, en SMP, todas las CPU acceden a una memoria común a la misma velocidad. En NUMA, la memoria en la misma tarjeta de procesador que la CPU, es decir, la memoria local, es accedida más rápidamente que la memoria en otras tarjetas de procesador, es decir, la memoria compartida, de ahí la nomenclatura "no uniforme". Como resultado, la arquitectura NUMA se escala mucho mejor a números más altos de CPU que la SMP. "NUMA coherente con caché" significa que el almacenamiento caché está soportado en el sistema local.
- Predicado: Por lo general, en una consulta de base de datos, un predicado es una restricción que filtra los registros de datos a ser recuperados desde una tabla relacional. Siguiendo la norma SQL, la cláusula WHERE de una operación SELECT define predicados que son expresiones lógicas que pueden evaluar a: verdadero, falso o desconocido. El predicado más básico es una comparación de igualdad. Por ejemplo, si hay una columna 'Color' en una tabla relacional, una consulta en esta tabla puede devolver solo aquellas filas de la tabla en donde; por ejemplo, está realmente presente un valor 'Rojo' en la columna 'Color'. Esto se lleva a cabo teniendo una cláusula

tal como sigue: WHERE Color = 'Rojo'. Toda clase de otras comparaciones pueden aplicarse incluyendo: diferente, mayor, menor, etc. siempre que los datos sobre los que esto se aplica sean consistentes para la comparación. Por ejemplo, si los valores en una columna de tabla relacional son cantidades de un apartado específico (cantidad) una cláusula válida puede ser: WHERE cantidad >= 200. Entonces, en este caso, la consulta devuelve solo las filas en donde las cantidades son mayores que o iguales a 200. Una consulta puede contener uno o más predicados de modo que puedan especificarse rangos.

- Sondeo: Exploración.
- Índice de sondeo: Considerando una tupla, tomar un valor dado de la tupla y explorar el índice de atributo asociado con ella para comprobar si hay consultas en el índice de consultas que obtendrían esta tupla como tupla de resultado.
- Datos de consulta junta: La forma en que la invención realiza las consultas se basa en tomar los datos a partir de las consultas, predicados y realizar a continuación la junta con estos datos y los registros reales de la tabla usando los índices de consulta.
- Índice de consulta, Indexación: Indexa todas las estructuras de datos de valores claves indicadas para dar un acceso aleatorio más rápido a un conjunto de datos dado. Por lo general, pueden usarse árboles de datos como índices. Un índice de consulta es un índice que contiene, como claves, valores que proceden de los predicados de la consulta para un atributo dado, es decir: para una columna; y como valores, referencias a las consultas. Las consultas de indexación consisten en tomar el conjunto de consultas activas, para recoger todos sus predicados y crear índices con estos datos. El número de índices de consulta construidos y la clase de estructura de datos usada, son impulsados por un optimizador de algoritmo ambicioso, del que se describe un ejemplo en la Figura 6.
- Replicación: Para asegurar la durabilidad, puede configurarse un conjunto de nodos para alojar exactamente los mismos datos, de modo que si uno falla, haya aún copias activas disponibles de los datos. Los nodos forman un grupo de replicación.
- Segmento: La memoria asignada a cada núcleo de CPU que es escaneada continuamente desde el comienzo al final usando el algoritmo de Escaneado de Reloj.
- Nodo de almacenamiento: Un subsistema, típicamente una máquina que comprende un total de N núcleos de CPU, que alojan un proceso controlador en uno de los núcleos de CPU y, por lo tanto, N-1 procesos de Escaneado de Reloj en los núcleos restantes.
- Tupla resultado: Considerando una consulta, la tupla que coincide con los predicados de la consulta y que será parte del conjunto de resultado enviado al remitente de la consulta.
- Escalabilidad: En ingeniería de telecomunicaciones y software, la escalabilidad es una propiedad deseable de un sistema, una red o un proceso, que indica su capacidad para o bien manejar cantidades crecientes de trabajo de manera elegante o para ser ampliada fácilmente. Por ejemplo, puede referirse a la capacidad del sistema para incrementar la productividad total bajo una carga creciente cuando se añaden recursos, típicamente recursos de hardware. Escalar verticalmente, o escalar hacia arriba, significa añadir recursos a un único nodo en un sistema, implicando típicamente la adición de unas CPU o memorias a un único ordenador. Escalar horizontalmente, o escalar hacia afuera, significa añadir más nodos a un sistema, tales como añadir un nuevo ordenador a una aplicación de software distribuido.

La Figura 1 ilustra adicionalmente el objetivo de la invención y compara el comportamiento que puede obtenerse con un almacén de datos como se describe en la siguiente especificación de la invención con uno de bases de datos tradicionales.

Como ya se ha mencionado en la sección de antecedentes la garantía de productividad y latencia es difícil de conseguir con sistemas de base de datos tradicionales cuando el nivel de actualizaciones a manejar es significativamente creciente y/o cuando han de procesarse cargas de trabajo de consultas altamente diversas. Los usuarios de sistemas de bases de datos relacionales tradicionales aún esperarán un tiempo de respuesta constante incluso aunque estén enviando consultas complejas. Los sistemas tradicionales se diseñan para conseguir el mejor rendimiento para cada consulta individual. Con este fin, se basan en optimizadores de consulta sofisticados y administradores habilitados para seleccionar los índices correctos y vistas materializadas. Dichos sistemas complejos son caros de mantener y no presentan un rendimiento predecible para cargas de trabajo impredecibles, en evolución incumpliendo frecuentemente por ello las expectativas de los usuarios.

La Figura 1 esboza dos gráficos que comparan el comportamiento real de un almacén de datos de acuerdo con la invención con el de una base de datos tradicional. Como se muestra por el gráfico 110, si la carga de actualización es ligera, una base de datos tradicional bien ajustada puede soportar una productividad de consultas elevada siempre que se hayan implementado los índices correctos y las vistas materializadas necesarias para soportarlas. Sin embargo, la productividad de consultas 112 siempre cae rápidamente con una carga de actualización creciente. Análogamente, como se muestra en el gráfico 120, la productividad de consultas 122 también disminuye rápidamente con el número de diferentes tipos de consulta, dado que más y más consultas requieren entonces escaneado completo de tablas. Obviamente, los efectos mostrados en la Figura 1 producen incluso una productividad más baja para cargas de trabajo que combinan elevada diversidad de consultas y actualizaciones simultáneas.

El almacén de datos y la tabla relacional distribuida de la invención se diseña específicamente para superar las limitaciones anteriores. Aunque un almacén de datos de acuerdo con la invención pueda ser inferior a soluciones tradicionales que se han ajustado para un conjunto reducido de tipos de consultas, presenta sin embargo un

rendimiento bueno y, más importante, predecible para todos los tipos de cargas de trabajo como se muestra por las curvas 114 y 124.

Como se describe en la siguiente especificación, el rendimiento predecible se consigue implementando una arquitectura de solo escaneado completo que no requiera ningún indexado de los datos almacenados. Un almacén de datos de acuerdo con la invención usa almacenamiento en memoria principal y particionado de datos y opcionalmente paralelizado fuerte el día-1 y diseños de no compartición para escalar hacia arriba linealmente en máquinas multi-núcleo. También se emplean escaneados colaborativos, es decir compartidos, para superar el cuello de botella de ancho de banda de memoria que se observa más frecuentemente en los sistemas tradicionales modernos en donde, aunque realmente la densidad de memoria se ha incrementado dramáticamente, el ancho de banda de la memoria no se ha mejorado, de lejos, al mismo ritmo que los rendimientos del procesador.

Por lo tanto, la invención divulga un novedoso algoritmo de escaneado colaborativo, llamado Escaneado de Reloj, para conseguir tanto elevada productividad de consultas como actualizaciones con latencia predecible. La idea detrás del algoritmo de Escaneado de Reloj es procesar consultas entrantes en lotes y modelizar el procesamiento de consulta/actualización como una junta entre sentencias de consultas y actualizaciones en un lado y la tabla de almacenamiento de datos por otro lado. Las juntas de bucle de índice enlazado pueden ser particularmente efectivas debido al acceso aleatorio es barato en una memoria principal de ordenador. Por lo tanto, en lugar de indexar la tabla, como se hace en bases de datos tradicionales, un almacén de datos de acuerdo con la invención indexa las consultas.

La invención se describe en detalle a continuación y se ilustra través de un caso de uso particular, aunque representativo de cualquier gran plataforma informatizada, es decir, a través de la descripción de un Sistema de Distribución Global (GDS), un comercio electrónico que forma la columna vertebral de la industria de viajes. Dicho GDS es por ejemplo AMADEUS, un proveedor de servicios europeo del liderazgo mundial para la gestión de reservas relacionadas con viajes, por ejemplo vuelos, hoteles, alquiler de coches, etc. Unos pocos GDS a través de todo el mundo comparten las mayores líneas aéreas del mundo y muchos miles de agencias de viaje usan los GDS para integrar sus datos. La base de datos del núcleo en cualquier GDS contiene típicamente decenas de millones de reservas de vuelo. Por razones históricas y de rendimiento, la copia autorizada de cada reserva se almacena en un BLOB (objeto grande binario) de unos pocos kilobytes, directamente accesible a través de una clave única. Para las reservas que necesitan mantenerse en línea, da como resultado una única tabla de hechos plana de varios centenares de gigabytes de tamaño. Dicha tabla BLOB debe sostener típicamente una carga de trabajo de varios centenares de actualizaciones y varios miles de búsquedas por valor clave por segundo.

El acceso por valor clave es suficiente para todas las cargas de trabajo transaccionales enfrentadas por el sistema. Sin embargo, el sistema no está adaptado para responder a la cantidad creciente de consultas de soporte de decisión, en tiempo real que seleccionan atributos no clave, por ejemplo: *"dar el número de pasajeros de 1ª clase que requieren una silla de ruedas y salida desde Tokio con destino en los Estados Unidos mañana".* Consultas como esta son cada vez más comunes y presentan restricciones de latencia exigentes, dado que se realizan decisiones operativas en base a sus resultados. Para dar soporte a dichas consultas, Un GDS mantiene típicamente un número creciente de vistas relacionadas materializadas en la tabla BLOB, algunas de las cuales se actualizan en tiempo real a través de una arquitectura de flujo de eventos. La existencia de estas vistas materializadas implica que hay unas pocas juntas en la carga de trabajo. La gran mayoría de las consultas son realmente de la forma SELECT <Atr1>, <Atr2>... FROM <Vista> WHERE..., con agregación ocasional.

La mayor vista existente es generalmente una vista de reservas de vuelo: un registro para cada persona en un avión, es decir la "vista materializada del Tique" o justamente el "Tique" en el contexto de la industria de viaje y de un GDS. Un registro de Tiques, es decir una única fila de la vista materializada de Tiques, es típicamente de unos pocos centenares de bytes de tamaño fijo y consiste en unas pocas decenas de atributos, muchos de los cuales son marcadores de elevada selectividad, por ejemplo, clase de asiento, silla de ruedas, vegetariano. Dado que una reserva de vuelo puede relacionarse con múltiples personas y vuelos, el Tique puede contener centenares de millones de dichos registros.

La vista materializada de Tique se actualiza unos pocos centenares de veces por segundo, en tiempo real. Las tasas de actualización pueden ser muchas veces más altas en breves períodos, dado que incidentes de meteorología o seguridad pueden producir grandes avalanchas de solicitudes de realojamiento de pasajeros. La carga de actualización es creciente con una tasa más baja que la carga de consultas, pero ya está provocando graves problemas con relación al mantenimiento del índice en la configuración actual.

La vista se usa en un gran número de servicios de datos: desde la generación de la lista de pasajeros de un único vuelo al análisis de perfil de cliente de diferentes líneas aéreas y mercados, es decir, pares de aeropuertos <origen, destino>. Dado que el sistema ha alcanzado un nivel de complejidad en el que la adición de vistas e índices ya no es factible ni económica, un número creciente de consulta sobre Tiques no se adapta al índice primario de <número de vuelo, fecha de salida>.

Como resultado, más y más consultas han de responderse en lotes, es decir fuera de línea, usando escaneados de tabla completos, con un impacto dramático en el rendimiento durante este periodo. Otras consultas que no se adaptan

al índice y no permiten procesamiento por lotes simplemente no están permitidas. Como solución a todas estas consultas que no justifican una vista propia, la invención propone, como se explica adicionalmente en lo que sigue, una única instancia de Tique.

Aunque la invención se describe a través del ejemplo particular de un GDS debe estar claro para un experto en la materia que muchos sistemas de inteligencia comercial y soporte de decisiones en tiempo real se enfrentan a requisitos similares a los analizados anteriormente y en la descripción que sigue.

La Figura 2 muestra una visión general del entorno distribuido informatizado en el que opera un almacén de datos de la invención.

Debido a que la memoria principal de cualquier ordenador está necesariamente limitada, una única máquina es generalmente incapaz de almacenar toda la tabla para las aplicaciones de almacenamiento de datos grandes y muy grandes consideradas por la invención. Por lo que, para escalabilidad y disponibilidad, la invención se basa en una arquitectura distribuida 200 basada en partición y replicación de datos en múltiples niveles horizontales.

La invención supone que la tabla de almacenamiento de datos se particiona horizontalmente entre grupos de replicación 210 que consisten en nodos de almacenamiento individuales, por ejemplo: 240. En un grupo de replicación, cada nodo contiene la misma partición de la tabla de almacenamiento de datos. El nodo maestro de un grupo de replicación actúa como un divisor de la carga de trabajo también llamado planificador de consultas, que agrupa y envía las consultas entrantes a diferentes réplicas esclavas basado en los predicados de selección de consultas. Esto homogeniza la carga de trabajo en réplicas individuales, incrementando la efectividad de los índices de consulta, con un resultado de escalado superlineal potencialmente para consultas. Una operación en los nodos de almacenamiento es o bien una consulta, es decir, una simple sentencia SELECT de estilo SQL (lenguaje de consultas estructurado) con agregación escalar opcional o bien una actualización, es decir una sentencia INSERT, UPDATE o DELETE. Las actualizaciones no están afectadas, dado que un grupo de replicación funciona en la forma leer-uno-escribir-todo (ROWA). Excepto para equilibrado de cargas y decisiones de adscripción al grupo realizadas por el maestro, las réplicas son completamente simétricas, lo que es altamente beneficioso respecto a la tolerancia a fallos. En una arquitectura tradicional, la pérdida de una vista o índice tiene un impacto dramático sobre al menos parte de la carga de trabajo. A diferencia de esto, la pérdida de una réplica en una arquitectura simétrica hace que disminuya la productividad en aproximadamente el mismo grado predecible para todas las consultas.

Conjuntamente los grupos de replicación forman la capa de almacenamiento 220. La capa de almacenamiento es también responsable de proporcionar durabilidad. La durabilidad se soporta por medio de puntos de comprobación y registro lógico de las actualizaciones. Para puntos de comprobación, un hilo en segundo plano es el responsable de enviar periódicamente las consultas sin predicado, es decir consultas que se adaptan a todas las tuplas. Los resultados de estas consultas se serializan y almacenan en un archivo plano, el punto de comprobación, en un almacenamiento persistente, por ejemplo, un disco duro. Considerando el registro, cada actualización debe añadirse al registro previamente a ser confirmada, es decir un registro de escritura adelantada. Un hilo en segundo plano envía periódicamente, por ejemplo cada segundo, el registro al disco y lo trunca a la marcación de tiempos del último punto de comprobación. Esto se analiza adicionalmente en la Figura 7.

Cada nodo de almacenamiento individual, por ejemplo 240, se implementa preferentemente en una máquina de núcleo múltiple. En dicha máquina cada núcleo 241 es una unidad de procesamiento independiente que tiene su propio caché de nivel 1 (L1) 242 y caché de nivel 2 (L2) 243; y a veces un caché de nivel 3 compartido (no mostrado), antes de interrelacionarse en un bus frontal externo 244 con bancos de memoria principal 245. Un almacenamiento persistente está siempre disponible en general en la forma de un disco duro 247 accesible a través de un controlador 246 de entrada/salida (E/S). El tiempo necesario para acceder a los datos almacenados desde un núcleo se incrementa significativamente cuando se pasa del caché L1 al caché L2, memoria principal y disco duro. Este último es el almacenamiento persistente usado para mantener los puntos de comprobación y archivos de registro anteriores que proporciona la durabilidad del almacén de datos. El particionado de la tabla relacional del almacén de datos se consigue de modo que un segmento de memoria se asigne a un único núcleo de procesador. La arquitectura y funcionamiento de un nodo de almacenamiento se analizan adicionalmente en la Figura 3.

Una o más capas de nodos de agregador 230 son responsables de las operaciones de enrutado a grupos de replicación y mezcla, es decir, agregación, de los resultados. En arquitecturas tradicionales, los administradores ajustan el rendimiento para proporcionar vistas o índices especiales en un nodo de almacenamiento pero no en otros o mediante el uso de tecnología totalmente diferente para ciertas réplicas, por ejemplo, replicación heterogénea. Este no es el caso con la invención en donde las réplicas son completamente homogéneas aunque pueden especializarse réplicas. Por ejemplo, algunas pueden ajustarse para consulta de rango mientras que otras pueden ajustarse para consultas de igualdad para mejorar adicionalmente los rendimientos.

Aun así, el procesamiento de consultas en lotes, es decir: agrupación de consultas basándose en sus predicados de selección; es beneficioso para el rendimiento. Por ejemplo, suponiendo que todas las consultas con un predicado de selección sobre el número de vuelo van a la réplica A, mientras que todas las consultas con un predicado de selección sobre el aeropuerto van a la réplica B; entonces, pueden indexarse y procesarse conjuntamente consultas similares

muy eficientemente. Las decisiones de agrupación se realizan autónomamente en tiempo de ejecución. Los índices de consulta son de vida extremadamente corta, es decir, solo viven el tiempo de escaneado, típicamente, menos de 2 segundos, de modo que la agrupación de consultas puede cambiar en cualquier momento. Por lo tanto, como ya se ha analizado, en una arquitectura tradicional, la pérdida de una vista o índice especial tiene un impacto dramático sobre al menos parte de la carga de trabajo. A diferencia de esto, la pérdida de una réplica en el sistema de la invención hace que la productividad disminuya aproximadamente en el mismo grado predecible para todas las consultas. Esto permite un rendimiento predecible en configuraciones de alta disponibilidad sin redundancia de datos adicional.

La capa de agregación media 230 es por ello responsable de la instrumentación y orquestación de la capa de almacenamiento. Define la forma en que se particionan los datos a través de los nodos de almacenamiento. El sistema de la invención no impone restricciones en cómo se realiza el particionado. La implementación del sistema decide si el mapeado de tuplas a nodos de almacenamiento es aleatorio, es decir particionado de todos contra todos o único es decir particionado aleatorio o partición por criterios específicos, por ejemplo todos los vegetarianos se almacenan en un nodo de almacenamiento específico. La capa de agregación mezcla, es decir, agrega los resultados generados por los nodos de almacenamiento permitiendo así el procesamiento de consultas de agregación, es decir, consultas que implican rangos de valores que requieren cálculos que incluyen, por ejemplo: suma de valores de atributos, hallar un valor mínimo y un valor máximo de atributo. Por lo tanto, cuando un usuario final pide que se calcule una suma/mínimo/máximo/cuenta, entonces todos los nodos de almacenamiento procesan la suma/mínimo/máximo/cuenta para su conjunto de segmento y la capa de agregación solo agrega los resultados de la subagregación. Este esquema de agregación multinivel que puede comprender también un controlador de hilos de escaneado que agrega los datos agregados.

Durante esta operación la capa de agregación esperaría confirmaciones o rechazos desde los nodos de almacenamiento que se implicaran en la evaluación de una operación antes del envío de la confirmación al cliente. Para actualizaciones, la confirmación es un mensaje de escritura confirmada o escritura rechazada. Para consultas la confirmación es un mensaje de fin de resultado especial, que declara que no se van a enviar más tuplas para la operación dada. La capa de agregación puede contener cualquier número de nodos de agregador, en uno o múltiples niveles. En una configuración de agregación multinivel, se usa un nodo agregador para particionado de la tabla de almacén de datos a través de un conjunto de nodos de agregador y mezcla sus resultados. Para ser recursivo, es decir multinivel, la agregación ha de ser factible, la interfaz de una capa de agregador debería coincidir con la interfaz de un nodo de almacenamiento. La invención requiere que todos los nodos de agregación sean totalmente sin estado. La carencia de estado permite al administrador del sistema añadir tantos nodos de agregador como se requiera para soportar la carga de trabajo. También elimina la complejidad de manejo de fallos de nodos de agregación. La capa superior, es decir, la capa de clientes 250 se espera que conozca un conjunto de nodos de agregador, al que es capaz de enviar operaciones. Un nodo agregador que no consiga confirmar o rechazar operaciones consecutivas debe considerarse como fuera de línea.

La Figura 3 visualiza la arquitectura del nodo de almacenamiento de acuerdo con la invención. Los nodos de almacenamiento exponen dos funciones principales: poner en cola una operación y sacar de la cola una tupla resultado. En lugar de poner en cola una operación y esperar al resultado, los usuarios, es decir, los nodos de agregador, se espera que pongan en cola simultáneamente un gran número de operaciones y saquen de la cola de modo asíncrono resultados. Cada nodo agregador a su vez puede servir de ese modo miles de clientes externos.

Una vez dentro de un nodo de almacenamiento, una operación 310 se divide 320 y se pone en la cola de entrada de uno o más hilos de escaneado 330. Cada hilo de escaneado es un hilo de espacio de usuario con afinidad de procesador fuerte que escanea continuamente una partición horizontal de la tabla del almacén de datos almacenada en una partición dedicada de un núcleo de memoria, es decir, un segmento como ya se ha definido. Los hilos de escaneado eliminan periódicamente operaciones de su cola de entrada 325 y las activan, es decir las consultas están listas para ser procesadas y completadas. En cualquier momento dado, un hilo de escaneado puede tener múltiples operaciones activas. Cuando un hilo de escaneado ejecuta su conjunto de operaciones activas contra los registros bajo el cursor de escaneado, genera un flujo de tuplas de resultado 335 que se mezclan adicionalmente 340. Una vez que una operación ha completado un escaneado completo de una partición de tabla, el hilo de escaneado pone una tupla de final-de-flujo especial en la cola de salida 350 y desactiva la operación.

Aunque esta arquitectura hace surgir cuestiones con relación a su imparcialidad: consultas "baratas" respecto a "caras" y utilización de recursos: hilos ocupados respecto a inactivos, el hecho de que cada operación lleve aproximadamente el mismo tiempo es una característica clave y un tipo fuerte de imparcialidad. Asimismo, la invención se basa en la ley de los grandes números: cuantas más operaciones compartan un cursor de escaneado, más representativas serán de la carga de trabajo como conjunto, equilibrando de ese modo la carga a través de los hilos de escaneado 330. Esto permite que miles de operaciones compartan un cursor de escaneado.

La memoria física se particiona en segmentos disjuntos, es decir: no solapados que se asignan a núcleos de predicador dedicados. Esta técnica se denomina como segmentación. Cada núcleo ejecuta un único hilo de escaneado con afinidad fuerte, es decir, los hilos no migran entre procesadores. Esta arquitectura de nada compartido permite un escalado lineal debido a las siguientes propiedades clave:

- No hay bloqueo debido a que se garantiza que un hilo de escaneado sea el único que actualiza registros en su segmento, la ejecución puede proseguir sin ningún bloqueo o cerrojo.
- Se consigue la máxima coherencia del caché dado que distintos cachés de procesadores nunca contienen una copia del mismo registro, de modo que son implícitamente coherentes con respecto a su rendimiento crítico. No se necesita volver a escribir registros en la memoria principal hasta que el cursor de escaneado se mueve a ellos, incluso si se modifican.
- La distancia NUMA mínima se obtiene debido a que los hilos de escaneado tienen afinidad de procesador fuerte y debido a que el código fuente se escribe de modo que la asignación de memoria se realiza al nodo NUMA local y no a un nodo NUMA remoto. Sus segmentos de memoria respectivos se asocian de modo único con un procesador. Esto es crítico en arquitecturas NUMA. Usando segmentación, las CPU nunca acceden a cada memoria local de otros excepto para pasar operaciones y resultados de operación, dando el ancho de banda de memoria máximo y latencia de acceso mínimo donde importa: la evaluación del predicado.

Bajo estas condiciones el ancho de banda de la memoria ya no es un cuello de botella como en bases de datos tradicionales. Los algoritmos de escaneado de la invención están ligados a la CPU, es decir, al procesador bajo carga.

La **Figura 4** analiza el algoritmo que escanea continuamente los datos en un hilo separado de control y compara el algoritmo de escaneado de la invención, mostrado en la **Figura 4b**, con algoritmos de la técnica anterior mostrados en la **Figura 4a**.

La invención divulga un algoritmo de escaneado colaborativo novedoso, llamado Escaneado de Reloj. El algoritmo de Escaneado de Reloj ejecuta lotes de consultas entrantes y modeliza el procesamiento de consulta/actualización como una junta entre las sentencias de consulta y actualización por un lado y la tabla por otro lado. En lugar de indexar los datos de la tabla, el Escaneado de Reloj crea índices temporales de las consultas y actualizaciones que se están evaluando actualmente. Para la memoria principal, esta configuración es particularmente efectiva, dado que el acceso aleatorio es barato. Se compara con el estado de los algoritmos de la técnica denominados posteriormente como Escaneado Clásico y Escaneado Elevador.

Como se muestra en la Figura 3 todos los algoritmos de escaneado 330 escanean continuamente los datos en un hilo de control separado. También, los algoritmos operan en ranuras de registros de tamaño fijo. La extensión de la invención a registros de tamaño variable o diferentes disposiciones de registro afectaría a los algoritmos en algún grado, pero no plantea problemas conceptuales, dado que no hay estructuras de datos auxiliares y los registros pueden disponerse con libertad.

Un algoritmo directo denominado en el presente documento como Escaneado Clásico se muestra en la **Figura 4a** bajo el nombre de "Algoritmo 1" 410. Esta es una implementación directa "clásica" de la línea de división/escaneado-mezcla mostrada en la Figura 3. En este caso cada hilo de escaneado solo procesa una operación entrante a la vez. La función de ejecutar una operación comprueba primero si está ocupada la ranura. Si no es así y la operación es un INSERT, se inserta un registro en la ranura siguiendo una política de "primer encaje". Si la ranura está ocupada y la operación no es un INSERT, se evalúan los predicados de selección de la operación. Si se satisfacen todos los predicados, la función o bien pone una tupla resultado en la cola de salida como un resultado de una operación SELECT o bien modifica la ranura si se realiza un UPDATE o DELETE. Después del procesamiento de todos los registros, el escaneado clásico pone una tupla de fin-de-flujo especial en la cola de salida y procede con la siguiente operación desde la cola de entrada. El tiempo de ejecución asintótico del Escaneado Clásico es $O(n * m)$ para n operaciones sobre m ranuras. Obviamente, es una pequeña ventaja de los recursos computacionales de los procesadores modernos, dado que no hace uso esencialmente de los cachés.

El Escaneado Elevador también se muestra en la Figura 4a bajo el nombre de "Algoritmo 2" 420. Esta es una primera mejora sobre el escaneado clásico. Zukowski et al. (referencia: M. Zukowski et al.; Cooperative scans: Dynamic bandwidth sharing in a DBMS; In Proc. VLDB'07, 2007) y Raman et al. (referencia: V. Raman et al. Constant-time query processing; in Proc. ICDE'08, 2008) han investigado previamente variantes del Escaneado Elevador para cargas de trabajo solo de lectura en bases de datos basadas en disco y memoria principal respectivamente. El Algoritmo 2 muestra una generalización de ejemplo del Escaneado Elevador para cargas de trabajo mezcladas.

El Escaneado Elevador mantiene una cola de operaciones activas que se ejecutan, en orden de llegada, contra la ranura bajo el cursor de escaneado antes de moverse a la siguiente ranura.

El Algoritmo 2 actualiza la cola activa en cada ranura. Todas las operaciones que han finalizado un escaneado completo se desactivan y se despeja la cola de entradas. Esta implementación de ejemplo del algoritmo hace esto solamente en límites de fragmentos que son equivalentes a bloques en una base de datos basada en disco. Como los bloques, los fragmentos se almacenan próximos entre sí. También, divide la cola activa en múltiples colas de tipo diferente, para evitar la ejecución de DELETE en una ranura vacía, por ejemplo.

El Escaneado Elevador es un así llamado escaneado cooperativo, en que deja que múltiples operaciones compartan el cursor de escaneado para mejorar la localidad del caché y evitar la creación de un cuello de botella de acceso a memoria (referencia: W. A. Wulf y S. A. McKee; Hitting the memory wall: implications of the obvious. ACM SIGARCH

Comput. Archit. News, 23(1):20-24, 1995; y referencia: P. A. Boncz et al. Database architecture optimized for the new bottleneck: Memory access. In Proc. VLDB '99, 1999.). Sin embargo, el tiempo de ejecución asintótico del Escaneado Elevador es aún *O(n * m)* para *n* operaciones sobre *m* ranuras.

El algoritmo de escaneado de la invención, es decir: el Escaneado de Reloj se muestra en la **Figura 4b** bajo el nombre de "Algoritmo 3" 430.

Incluso aunque el Escaneado Elevador mejora sobre el comportamiento de caché del Escaneado Clásico, esta mejora es como mucho un factor constante en el tiempo de ejecución. Por el contrario, el Escaneado de Reloj junta consulta/actualización de datos sobre conjuntos de consultas/actualizaciones para permitir mejoras asintóticas del tiempo de ejecución. La Figura 4b analiza el algoritmo de escaneado en sí mismo. Las juntas de consulta/actualización de datos se cubren en detalle en la siguiente figura.

Es conocido que intercalar la evaluación de múltiples predicados de selección es útil para optimizaciones de bajo nivel. De hecho, intercalar la evaluación de múltiples operaciones puede crear oportunidades adicionales para maximizar la localidad-caché. Sin embargo, en presencia de escrituras, la reordenación e intercalado sin restricción de operaciones activas puede comprometer la consistencia. El Escaneado de Reloj hace así la distinción entre operaciones de lectura y escritura y procesa en lotes independientemente la ejecución de cada clase de operación de lectura o escritura. Esto crea una gran oportunidad de intercalado para lecturas incluso bajo una carga de trabajo de escritura pesada.

Una ilustración de alto nivel de la idea del algoritmo 440 se muestra en la Figura 4b. El algoritmo ejecuta continuamente dos escaneados circulares sobre el segmento: un escaneado de lectura 442, un escaneado de escritura 444. El cursor de lectura no puede sobrepasar al cursor de escritura y viceversa, es decir, el cursor de lectura está siempre con alguna delta menor de un ciclo por detrás del cursor de escritura. El cursor de escritura ejecuta operaciones estrictamente en el orden de llegada. Puede probarse fácilmente que el cursor de lectura verá siempre una instantánea consistente si el algoritmo solo activa operaciones en el registro 0, independientemente del orden en el que se ejecuten las consultas. El Escaneado de Reloj, dado en el Algoritmo 3, mezcla realmente los dos cursores lógicos en un único físico para una localidad de caché más elevada. En cada iteración del bucle infinito, primero despejan las colas de entrada y crea planes de junta para las consultas y actualizaciones activas. Entonces, realiza el escaneado en fragmentos reales del segmento, juntando cada fragmento de registro con el conjunto de consultas y el conjunto de actualizaciones.

El algoritmo de Escaneado de Reloj no solo reordena operaciones, sino que realmente intercala su ejecución mediante la realización de juntas de consulta/actualización de datos. En este contexto, una junta es cualquier evaluación de predicados contra registros que conduzca al mismo conjunto de tuplas resultado como si se ejecutara cada consulta/actualización en cada registro. La tarea de hallar la mejor estrategia de junta posible, es decir el mejor plan multiconsulta posible, se delega a un optimizador multiconsulta. Una implementación simple pero efectiva de dicho optimizador multiconsulta se describe en la Figura 6. La complejidad del tiempo de ejecución del Escaneado de Reloj se determina por las juntas.

Las Figuras 5a, 5b y 5c analizan dos tipos de algoritmos de junta, concretamente: "Unión Índice Junta" e "Índice Unión Actualización Junta". El Escaneado de Reloj permite asintóticamente un mejor rendimiento que el Escaneado Elevador debido a que, como se ha mencionado anteriormente, reordena e intercala consultas para realizar juntas de consulta para actualización de datos. La expresión junta de consulta de datos fue acuñada por Chandrasekaran et al. (referencia S. Chandrasekaran y M. J Franklin: Streaming queries over streaming data. In Proc. VLDB '02, 2002) y se basa en la idea de interpretar un conjunto de consultas pendientes como una relación de predicados. Los dos algoritmos de junta se analizan posteriormente.

Una solución flexible y general de la invención para implementar junta de unión es indexar predicados. Por lo tanto, una junta de datos de consulta consciente de la caché basada en predicados de corta vida indexa: Índice Unión Junta 510 se implementa como el "Algoritmo 4" mostrado en la Figura 5a. La conciencia de caché se consigue forzando el hilo de escaneado a ejecutarse en un núcleo computacional designado creando así afinidad fuerte; y mediante la captura de datos desde el segmento de tabla relacional en fragmentos de tamaño compatible con uno o más niveles de caché de memoria dedicados al núcleo computacional designado para impedir pérdidas en el caché. Por lo tanto, la conciencia de caché es una forma de programación en donde cada código fuente ha de considerar cómo impacta en el uso del caché. Es por ello no solamente una materia del S.O./hardware sino principalmente el aspecto que determina cómo se escribe el código fuente.

Un índice de predicado mapea un único valor de atributo a un conjunto de consultas. Por ejemplo, las tres consultas q_1: edad = 12, q_2: edad = 27, y q_3: edad = 27 podrían indexarse en un mapa multifunción que devuelve $\{q_2; q_3\}$ cuando se prueba con un registro con edad = 27. Los predicados de rango tales como 30 < edad <50 pueden indexarse en cualquier estructura de índice espacial que soporte punzadas o consultas de punzada, es decir, consultas enviadas en geometría computacional, coincidencia de patrón y otras aplicaciones en donde hay necesidad de determinar rápidamente cuál de una colección de intervalos se solapa en un punto. Dicha estructura es, por ejemplo, la "Lista de salto de intervalos", una estructura de datos dirigida a hallar todos los intervalos que se solapan en un punto.

La **Figure 5a** visualiza la estructura de datos 520 del Índice Unión Junta. Existe exactamente una ruta de acceso para cada consulta. Cualquiera de los predicados de consulta es parte de un índice de predicado o la consulta es parte de un conjunto de consultas sin indexar. Dado que esto constituye un particionado del conjunto de consultas, siguiendo cada ruta de acceso, es decir, cada índice más el conjunto de consultas sin indexar, con cada registro, y uniendo las tuplas resultado, se conduce a la misma relación de resultado que la ejecución de cada consulta contra cada registro. El Algoritmo 4 se deduce directamente de esta observación. Obsérvese que poner todos los resultados de las consultas de ejecución en una cola de salida común da semántica de unión.

La complejidad en tiempo de ejecución del peor caso del Índice Unión Junta, es decir, cada registro coincide con cada índice, no es mejor que la ejecución de cada consulta contra cada registro, como se realiza por el Escaneado Clásico y el Escaneado Elevador. Sin embargo, el Índice Unión Junta es más rápido para cualquier conjunto razonablemente selectivo de predicados, debido a que la prueba de índice lleva inmediatamente a todas las consultas no coincidentes a fuera de consideración. El tiempo de ejecución es dominado por el coste de prueba del índice, que es constante o logarítmico. Esto es análogo a la diferencia entre una junta en bucle anidado y un índice de junta en bucle anidado en procesamiento de consultas tradicional. Para grandes conjuntos de consultas, la aceleración resultante es significativa.

A diferencia del pseudocódigo dado que describe el Algoritmo 4, la implementación optimizada real vectoriza la prueba de índice, es decir, pasa el fragmento completo de una función de prueba de índice en lugar de un único registro. Esto da una localidad de caché de datos e instrucción significativamente más elevada.

En términos de índices, la invención implementa actualmente una matriz mellada para atributos con dominio pequeño (por ejemplo: género), un índice de función encadenada con prueba lineal para predicados de igualdad, así como un árbol en R monodimensional empaquetado (referencia: A. Guttman; R-trees: a dynamic index structure for spatial searching; in Proc. SIG- MOD'84, 1984) para predicados de rango. Estas estructuras de índice siempre terminan teniendo un mejor rendimiento que algunas otras estructuras más complejas debido a su elevada localidad de caché de datos e instrucción.

Un ejemplo simple 530 de consultas de indexado se muestra en la **Figura 5b**. El optimizador de consulta puede decidir, por ejemplo, sobre la base de estadísticas internas recogidas por un hilo de escaneado y mediante el análisis del conjunto de consultas particular, que se necesitan tres índices de consulta. En primer lugar, en este caso, todas las consultas que tienen un predicado sobre el atributo "Aeropuerto de salida" se indexan conjuntamente con un primer índice 531. Entonces, todas las consultas restantes que tienen un predicado sobre el atributo "Aeropuerto de llegada" se indexan conjuntamente en un segundo índice 532. Finalmente, un último índice con consultas con predicado sobre el atributo "Línea aérea" se indexa conjuntamente en el índice 533. La última consulta Q6 534 que se estima por el optimizador no merece ser indexada. Por lo tanto, se recuperan posiblemente múltiples registros 535 rápidamente gracias al indexado.

El Índice Unión Actualización Junta 540 se implementa como el "Algoritmo 5" mostrado en la **Figura 5c**. Bajo una carga de actualización pesada, es conveniente usar índices de predicado también para actualizaciones. El problema es que las actualizaciones han de ejecutarse en orden de serialización y se expresan como marcas de tiempo. Lo que lo hace difícil de realizar eficientemente es el hecho de que un estado de ranura puede cambiar después de cada actualización, cambiando de ese modo el conjunto de actualizaciones coincidentes.

El Índice Unión Actualización Junta dado en el Algoritmo 5 resuelve el problema. Mantiene una cola *iq* de los INSERT indexables y un conjunto de índices de predicado *is*, mientras que *us* contiene los UPDATE y los DELETE sin indexar.

La función RealizActualiz 550 es una extensión del Índice Unión Junta mostrado en la sección previa. Recoge un conjunto *M* de todas las actualizaciones de registros coincidentes. Entonces, busca la actualización *u* que pertenece a *M* con la marca de tiempos más baja mayor o igual a *t*, si existe y la ejecuta. La variable *t* es inicialmente 0 y asegura que las actualizaciones se ejecutan en el orden de la marca de tiempos como sigue. Si *u* era un DELETE, la repetición finaliza debido a que la ranura está vacía. De lo contrario, la función se repite para *t = u.marcatiempo+1.* Esto segura que ninguna actualización *v* en la que *v.marcatiempo* sea igual o mayor que *u.marcatiempo* se ejecutará sobre el registro actualizado, incluso aunque *v* permanezca en los índices y pueda aparecer repetidamente en *M* cuando se repite la función.

El Índice Unión Actualización Junta prosigue a la siguiente ranura cuando *t* se hace mayor que cualquier marca de tiempos legal. Esto sucede si y solo si la ranura está ocupada pero no hay actualización coincidente con una marca de tiempos más pequeña o igual a t o la ranura está vacía pero no restan operaciones de inserción en *iq*.

Como para el rendimiento, puede indicarse que en el peor caso, solo hay UPDATE a ser juntadas y cada uno de estos *n* UPDATE coincide con cada registro, cada vez que se llama a RealizActualiz. Dado que |M| es más pequeño o igual a *n*, la profundidad de la repetición es hasta *n* y la complejidad del tiempo de ejecución del peor caso para m registros es $O(n^2 * m)$. En realidad, *M* contendrá típicamente 0 o 1 actualización, de modo que el tiempo de ejecución está dominado por el coste de prueba de los índices, que es constante o logarítmico en *n*. En nuestra implementación optimizada, la "repetición" es justamente una asignación a *t* y una sentencia goto.

La **Figura 6** analiza el optimizador multiconsulta dirigido a la planificación de las juntas descritas en la figura previa. La expresión de optimización multiconsulta se refiere tradicionalmente a la práctica de hallar subexpresiones comunes entre el conjunto de consultas, con el objetivo de compartir y canalizar los resultados intermedios, es decir, selecciones y juntas parciales, en un plan de (multi) consulta global (referencia: T. K. Sellis; Multiple-query optimization; ACM TODS, 13(1):23-52, 1988; y referencia: S. Harizopoulos et al.; Qpipe: A simultaneously pipelined relational query engine; in Proc. SIG- MOD'05, 2005). Dicho planteamiento es útil para pequeños conjuntos de consultas de ejecución larga, compleja.

A diferencia de esto, la invención se diseña para grandes conjuntos de consultas de ejecución corta, simple. El problema de optimización consiste en este caso en hallar un conjunto de índices de predicado que minimicen el coste, es decir el tiempo de ejecución, de los algoritmos de junta dados previamente. También, la optimización multiconsulta significa realmente en este caso consultas y actualizaciones, dado que se indexan exactamente en la misma forma. En cualquier caso, el problema es NP (no determinista y polinomial en el tiempo) fuerte, como el caso especial de "hallar el conjunto mínimo de índices de predicado que cubran todas las consultas/actualizaciones" ya es una instancia de cobertura del conjunto mínimo.

Dada la corta vida útil de un plan de consulta/actualización, típicamente de 1 segundo, no es posible hallar una solución óptima. De modo que en su lugar, el optimizador usa un algoritmo ambicioso, es decir: el "Algoritmo 7" 610. En cada iteración, el algoritmo construye un índice sobre el atributo que da la ganancia más elevada y a continuación lleva a fuera de consideración a todas las consultas/actualizaciones que están cubiertas por el índice. La función de Ganancia 620 se define como se muestra en la Figura 6.

Se basa en la siguiente idea. El toque de una consulta/actualización *q* se asocia con una cierta sobrecarga es decir ramificación, acceso a valores de atributo, etc., que debe evitarse. La probabilidad de tocar q después de probar un índice-predicado en a es la selectividad de q con respecto a a. Si *q* no tienen predicado en a esa probabilidad es 1. La ganancia es entonces el número esperado de operaciones en *Q* que no han de ser tocadas dado un índice en a. Obviamente, maximizar este número minimiza el número de operaciones que se espera sean tocadas. El valor empíricamente obtenido *umbr* (2.5) impide al optimizador construir más índices de los beneficiosos.

Para calcular la métrica de la ganancia y recoger un conjunto adecuado de índices de predicado, el optimizador requiere una estimación de la selectividad de cada predicado. La invención mantiene un conjunto pequeño de estadísticas internas que pueden mantenerse de modo eficiente justamente escaneando los datos: número de registros, número de valores nulos para cada atributo y número de valores distintos para cada atributo. Para el cálculo de estas estadísticas internas, la invención emplea una técnica simple pero efectiva conocida como recuento lineal o probabilístico (referencia P. Flajolet y G. N. Martin; Probabilistic counting algorithms for data base applications; J. Comput.Syst. Sci., 31(2):182-209, 1985. y referencia: K.-Y. Whang et al.; A linear-time probabilistic counting algorithm for database applications; ACM TODS, 15(2):208{229, 1990.). El algoritmo de recuento probabilístico se implementa como un efecto colateral de una consulta de estadística periódica, que para el algoritmo de escaneado es solamente una consulta no condicional.

La **Figura 7** analiza la durabilidad.

Para la durabilidad, la invención usa una combinación de registro de escritura adelantada y punto de comprobación. en cualquier momento, se mantiene un punto de comparación en el disco en la forma de una instantánea difusa 710, es decir, un conjunto de instantáneas con marcación de tiempo para cada segmento de memoria 720, que se obtienen a través de consultas de instantánea no condicionales 735. Cuando se ejecutan contra un registro, las consultas de instantánea copian el registro en una memoria intermedia basculante 730 que escribe de modo síncrono los registros de vuelta al disco.

Mediante la planificación de las consultas instantáneas sobre segmentos en una forma de todos contra todos, $n + 1$ segmentos de espacio de disco son suficientes para tener siempre un punto de comprobación de n segmentos de memoria. La memoria intermedia basculante de tamaño constante asegura la predictibilidad. Los datos se escriben al disco por el hilo controlado.

Después de un bloqueo, la recuperación prosigue simultáneamente para cada segmento. En primer lugar, el segmento de instantánea es cargado desde el disco y a continuación se reproduce el registro. Se implementa un registro de repetición lógico que es extremadamente eficiente, pero también implica un modo de autocuidado es decir, es decir no hay cuidado y no hay vuelta atrás gestionada por el sistema. El sistema realiza lo que el usuario pide sin la capacidad de confirmación/cancelación. Una extensión a la completa atomicidad se obtiene añadiendo un registro físico y una fase de deshacer a la reproducción del registro.

La **Figura 8** muestra resultados experimentales que están en línea con el objetivo de la invención analizado en la sección de antecedentes y en la Figura 1. Los resultados obtenidos con un almacén de datos de la invención se comparan con los de una base de datos relacional disponible comercialmente que se ha ajustado de modo que se aplique una comparación limpia.

El experimento se ha llevado a cabo sobre una máquina de 16 núcleos del tipo que se espera sea usado para implementar los nodos de almacenamiento. Se ha construido a partir de cuatro procesadores de núcleo cuádruple disponibles comercialmente con 32 GB (gigabytes, es decir: 2^{30} bytes) de memoria principal compuesta de módulos RAM (memoria de acceso aleatorio) que tienen un bus de salida DDR2 (doble tasa de datos) ejecutándose a 667 MHz (megahercios, es decir 10^6 hercios). Había disponible entonces un ancho de banda de memoria acumulado de más de 42 GB/s. Cada núcleo tenía una frecuencia de reloj de 2,2 GHz (10^9 hercios), un caché L1 con 64 KB (2^{10} bytes) dobles de memoria para instrucciones y datos y un caché de 512 KB L2. La máquina ejecutaba un núcleo Linux de 64 bits SMP (multiproceso simétrico) como sistema operativo (S.O.). El experimento se ejecutó a partir de una tabla de almacenamiento de datos de 15 GB de la clase analizada previamente, es decir, la vista de "Tiques" usada en la industria de viajes por un GDS como AMADEUS.

Para verificar que los objetivos de la invención se cumplen con la máxima productividad de consultas, en consultas/segundo, se midió para una carga de trabajo mezclada de la clase usada por el GDS anterior con cargas de actualización variables. Esto corresponde al gráfico 810 en el que la curva 812 se refiere a la invención y muestra que la productividad de consultas está difícilmente afectada por el número de actualizaciones simultáneas, en un rango 0-100, mientras que la curva 814, para la base de datos comercial, cae bruscamente tan pronto como se realizan más de unas pocas actualizaciones por segundo.

También, se midió la productividad para una carga de trabajo solo de lectura variable sintética como se muestra en el gráfico 820 en el que la productividad se trata contra un parámetro de la carga de trabajo sintética que es representativo de la diversidad de consultas. Mientras que la productividad de consultas 824 de la base de datos comercial está cercana a la de la invención para una carga de trabajo estándar cae a continuación significativamente más rápido que la de la invención 822. Más interesante que los números absolutos son las formas de las curvas. Realmente coinciden con las de la Figura 1.

La invención cumple así el objetivo asignado y garantiza significativamente números de productividad más altos y de latencia más baja con mínimo esfuerzo de administración, incluso si la carga de trabajo está fluctuando o evolucionando rápidamente. Si fuera necesario, los objetivos de productividad y latencia pueden cumplirse simplemente añadiendo solamente hardware, debido a la escalabilidad del almacén de datos de acuerdo con la invención.

En resumen, como un único nodo de almacenamiento, la invención es un sistema de gestión de datos que proporciona rendimiento predecible para carga de trabajo impredecible permitiendo así recogida de datos en tiempo real. Con este fin, es una arquitectura libre de bloqueos y libre de índices capaz de ejecutar consultas analíticas no SQL masivas con rendimiento predecible en una única tabla relacional.

Es también una herramienta de gestión de datos de la memoria principal basándose en escaneados completos cooperativos. Los datos manejados por la herramienta no están indexados, pero los predicados de las consultas/operaciones lo están, tanto cargas de trabajo de lectura como de escritura y a continuación se procesan en lotes.

Un nodo de almacenamiento de acuerdo con la invención es consistente a nivel de segmento. Siendo un segmento un conjunto de datos dedicado a un algoritmo de Escaneado de Reloj ejecutado por un núcleo de CPU. Consistencia significa que cualquier solicitud de lectura verá o bien una única solicitud de escritura totalmente aplicada al segmento o ninguna en absoluto. Para conseguir esto se usan dos cursores: siendo el primero responsable de las escrituras y el segundo de las lecturas. La configuración en donde solo se activan consultas/operaciones al comienzo de un escaneado, es decir, en el registro 0, garantiza que se cumple la consistencia anterior.

La eficiencia se obtiene primero a partir del indexado de consulta/operación. Con este fin, se crean diferentes clases de índices basándose en las operaciones de consulta soportadas. Por lo tanto, para igualdades o para rangos, se construyen diferentes índices. También se obtiene eficiencia a través del uso de afinidad fuerte, minimizado de las distancias NUMA e indexado de consulta consciente del caché y procesamiento en el contexto de escaneados completos de datos continuos.

Los escaneados completos permiten más aún la recogida de estadísticas internas de pequeño tamaño con recuento probabilístico sobre el número de registros, número de valores nulos y número de valores distintos que pueden usarse eficientemente por la gestión del caché de la CPU y la construcción de índices de consulta/operación. El planteamiento basado en el escaneado completo es también beneficioso para la durabilidad dado que permite que se realicen unos puntos de comprobación difusa con baja sobrecarga sobre el sistema. La agregación de consultas tales como: suma, recuento, máximo o mínimo, puede ejecutarse también eficientemente.

Como un sistema distribuido la invención se extiende al modelo de almacenamiento único para permitir la escalabilidad, disponibilidad, particionado a gran escala y especialización de nodos. Para lograr esto, se implementa una capa de agregación con la finalidad de encaminar las consultas a los nodos de almacenamiento apropiados y la mezcla de resultados procedente de ellos. Por lo tanto, se consigue la escalabilidad dado que la capa de agregación permite que se construyan granjas con cientos de nodos de almacenamiento que pueden almacenar conjuntos de

datos mucho mayores que un único nodo.

La disponibilidad se obtiene debido a que los nodos de almacenamiento se organizan en grupos de replicación en el que cada máquina del grupo aloja los mismos datos. Entonces, si una máquina pasa a estar defectuosa, los datos estarían aún disponibles desde las otras réplicas del grupo.

El particionado horizontal a gran escala puede conseguirse a través de múltiples grupos de replicación que alojan datos diferentes. Entonces, los nodos de enrutado de la capa de agregación pueden, si la consulta es suficientemente expresiva, saltarse algunos grupos de replicación; por lo tanto, no enviándoles una consulta que se sabe que no devolvería resultados para esta consulta. Son posibles entonces diferentes estrategias de particionado.

Puede llevarse a cabo también especialización del nodo. Los nodos dentro de un grupo de replicación pueden configurarse de modo diferente incluso si alojan los mismos datos. Por ejemplo, algunos nodos pueden tener estructuras apropiadas para procesar mejor las consultas de rango y otros nodos pueden construir estructuras para procesar mejor predicados de tipo igualdad.

REIVINDICACIONES

1. Un método para almacenar y recuperar datos en un nodo de almacenamiento (240) de un almacén de datos, el nodo de almacenamiento (240) comprende al menos un núcleo de procesador y su memoria principal asociada; comprendiendo el método la etapa de almacenar al menos un segmento de una tabla relacional en el medio de almacenamiento de datos del nodo de almacenamiento (240), **caracterizado por que** el método comprende las siguientes etapas:

almacenar en cada memoria principal asociada de un núcleo de procesador un único segmento;
ejecutar desde el núcleo de procesador un hilo de escaneado dedicado al escaneado de dicho segmento;
escanear continua y completamente dicho segmento;
recibir y procesar lotes de consulta y operaciones de actualización;
indexar un conjunto de predicados de operaciones de consulta y actualización de un lote en un comienzo de cada escaneado de datos;
operaciones de consulta de junta y actualización de un lote para registros de datos recuperados de dicho segmento que coincidan con los predicados de las operaciones de consulta indexada y actualización;
completar progresivamente las operaciones de consulta y actualización de un lote siempre que los datos de los registros de datos juntados se recuperen mediante el hilo de escaneado mientras se escanea en dicho segmento.

2. Método de acuerdo con la reivindicación 1 en el que el hilo de escaneado ejecuta un algoritmo de escaneado de reloj (440) que comprende:

un cursor de escritura (444) y un cursor de lectura (442) que escanean exhaustivamente dicho segmento para procesar, en cada ciclo de escaneado, respectivamente, todas las operaciones de consulta y actualización pendientes de un lote;
en el que los cursores de escritura y lectura permiten que se escriban registros de datos de dicho segmento antes de que el cursor de lectura pueda leerlos;
en el que todas las operaciones de consulta se activan al comienzo de un ciclo de escaneado;
en el que todas las operaciones de actualización o bien se aplican totalmente o no se aplican en absoluto a los registros de datos de dicho segmento.

3. El método de acuerdo con una cualquiera de las reivindicaciones anteriores en el que las operaciones de actualización incluyen modificación, borrado e inserción de registros de datos desde/dentro del segmento.

4. Método de acuerdo con una cualquiera de las reivindicaciones anteriores en el que mientras se escanea el segmento de tabla relacional, se recogen estadísticas internas por el hilo de escaneado que cuenta un número de registros, un número de valores nulos y un número de valores distintos hallados simultáneamente para un atributo dado de dicho segmento y en el que los índices de consulta se construyen a partir de estas estadísticas internas.

5. Método de acuerdo con la reivindicación anterior en el que para la recogida de estadísticas el hilo de escaneado cuenta un número de registros, un número de valores nulos y un número de valores distintos presentes en unos registros de dicho segmento.

6. Método de acuerdo con una cualquiera de las cuatro reivindicaciones anteriores en el que las estadísticas internas construidas sobre los contenidos de los registros de datos se usan para construir diferentes tipos de índices de consulta.

7. Método de acuerdo con una cualquiera de las reivindicaciones anteriores que comprende la etapa de extraer valores de los registros mientras se escanea dicho segmento; realizar al menos uno cualquiera de los siguientes cálculos: sumar valores de atributos, hallar un valor mínimo de atributo y hallar un valor máximo de atributo, contar el número de valores no nulos; hallar una consulta proporcionando el resultado de al menos dicho cálculo.

8. Método de acuerdo con cualquier reivindicación anterior en el que el hilo de escaneado construye una instantánea de los registros de datos mientras escanea dicho segmento y en el que la instantánea de los registros de datos se almacena cuando se completa en una memoria duradera.

9. Método de acuerdo con la reivindicación anterior en el que se mantiene un archivo de registro lógico de las transacciones actuales en una memoria duradera por parte del hilo de escaneado mientras escanea dicho segmento.

10. Método de acuerdo con la reivindicación anterior en el que la durabilidad de la memoria principal se obtiene mediante la recarga de la última instantánea completa y la reproducción del archivo de registro de las transacciones actuales.

11. Método de acuerdo con la reivindicación anterior en el que dicho segmento está totalmente comprendido y permanece residente en la memoria principal.

12. Método de acuerdo con la reivindicación anterior en el que el hilo de escaneado es forzado a ejecutarse en un núcleo de procesador designado y recoge datos de dicho segmento en fragmentos de tamaño compatible con uno o más niveles de almacenes de memoria caché dedicados al núcleo de procesador designado para impedir pérdidas de almacén intermedio.

13. Un nodo de almacenamiento (240) de un almacén de datos, que comprende al menos un núcleo de procesador y su memoria principal asociada, una tabla relacional que comprende al menos un segmento **caracterizado por que** el nodo de almacenamiento (240) se dispone de modo que;
la memoria principal almacena un segmento simple;
el núcleo de procesador ejecuta un hilo de escaneado dedicado al escaneado de dicho segmento;
el hilo de escaneado escanea continua y totalmente dicho segmento;
se reciben y se procesan lotes de operaciones de consulta y actualización;
se indexa un conjunto de predicados de operaciones de consulta y actualización de un lote en un comienzo de cada escaneado de datos;
se juntan operaciones de consulta y actualización de un lote para registros de datos recuperados de dicho segmento que coincidan con los predicados de las operaciones de consulta indexada y actualización; las operaciones de consulta y actualización de un lote se completan progresivamente siempre que los datos de los registros de datos juntados se recuperen mediante el hilo de escaneado mientras se escanea en dicho segmento,

14. Un almacén de datos distribuido que comprende:

una capa de almacenamiento (220) compuesta por una pluralidad de nodos de almacenamiento (240) de acuerdo con la reivindicación anterior;
una capa de agregación (230) compuesta por nodos de agregador (230) indicada para encaminar lotes de operaciones de consulta y actualización a los nodos de almacenamiento (240) y mezclar los resultados devueltos por los nodos de almacenamiento (240).

15. El almacén de datos distribuidos de acuerdo con la reivindicación anterior en el que la capa de almacenamiento (220) se organiza en grupos de nodos de almacenamiento (240), almacenando todos los nodos de almacenamiento (240) de un grupo un segmento idéntico de una tabla relacional y en el que los grupos de nodos de almacenamiento (240) almacenan diferentes conjuntos de segmento.

16. El almacén de datos distribuidos de acuerdo con una cualquiera de las dos reivindicaciones anteriores dispuesto para extraer valores de los registros mientras escanea cada segmento; agregar estos valores; realizar al menos uno cualquiera de los siguientes cálculos: sumar valores de atributos, hallar un valor mínimo de atributo y hallar un valor máximo de atributo, contar el número de valores no nulos; completar una consulta proporcionado los resultados del cálculo realizado en dichos valores agregados.

17. El almacén de datos distribuidos de acuerdo con una cualquiera de las tres reivindicaciones anteriores en el que los grupos de replicación (210) se especializan para manejar operaciones de consulta específicas.

18. El almacén de datos distribuidos de acuerdo con una cualquiera de las cuatro reivindicaciones anteriores en el que la capa de agregación (230) se dispone para encaminar solamente lotes específicos de operaciones de consulta y actualización a los nodos de almacenamiento especializados (240) correspondientes.

19. El almacén de datos distribuidos de acuerdo con una cualquiera de las cinco reivindicaciones anteriores configurado de modo que los nodos de almacenamiento especializados (240) se organizan para acelerar el procesamiento de predicados de tipo igualdad y de tipo rango.

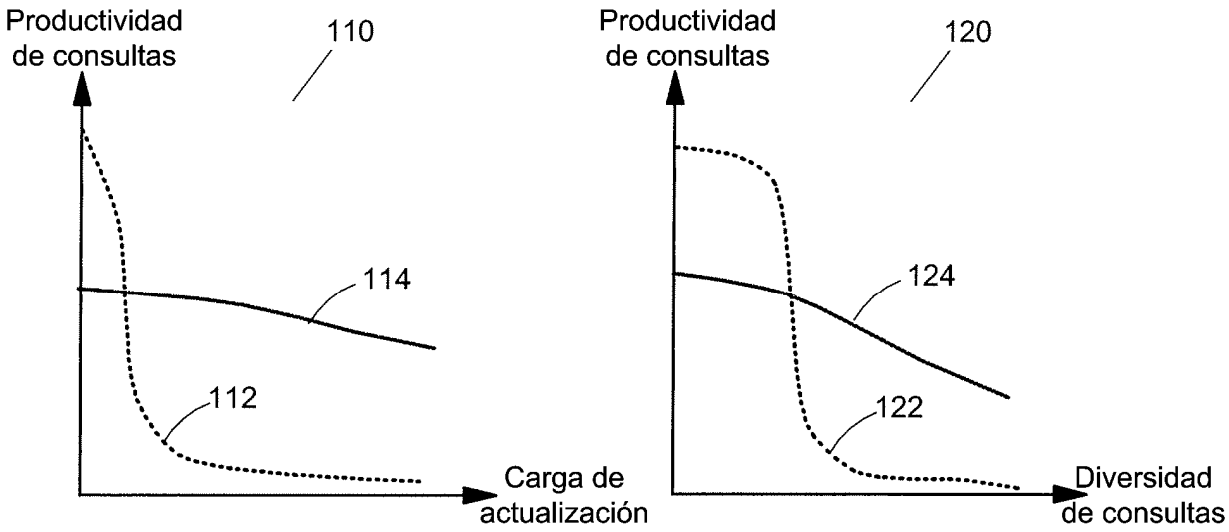

Figura 1

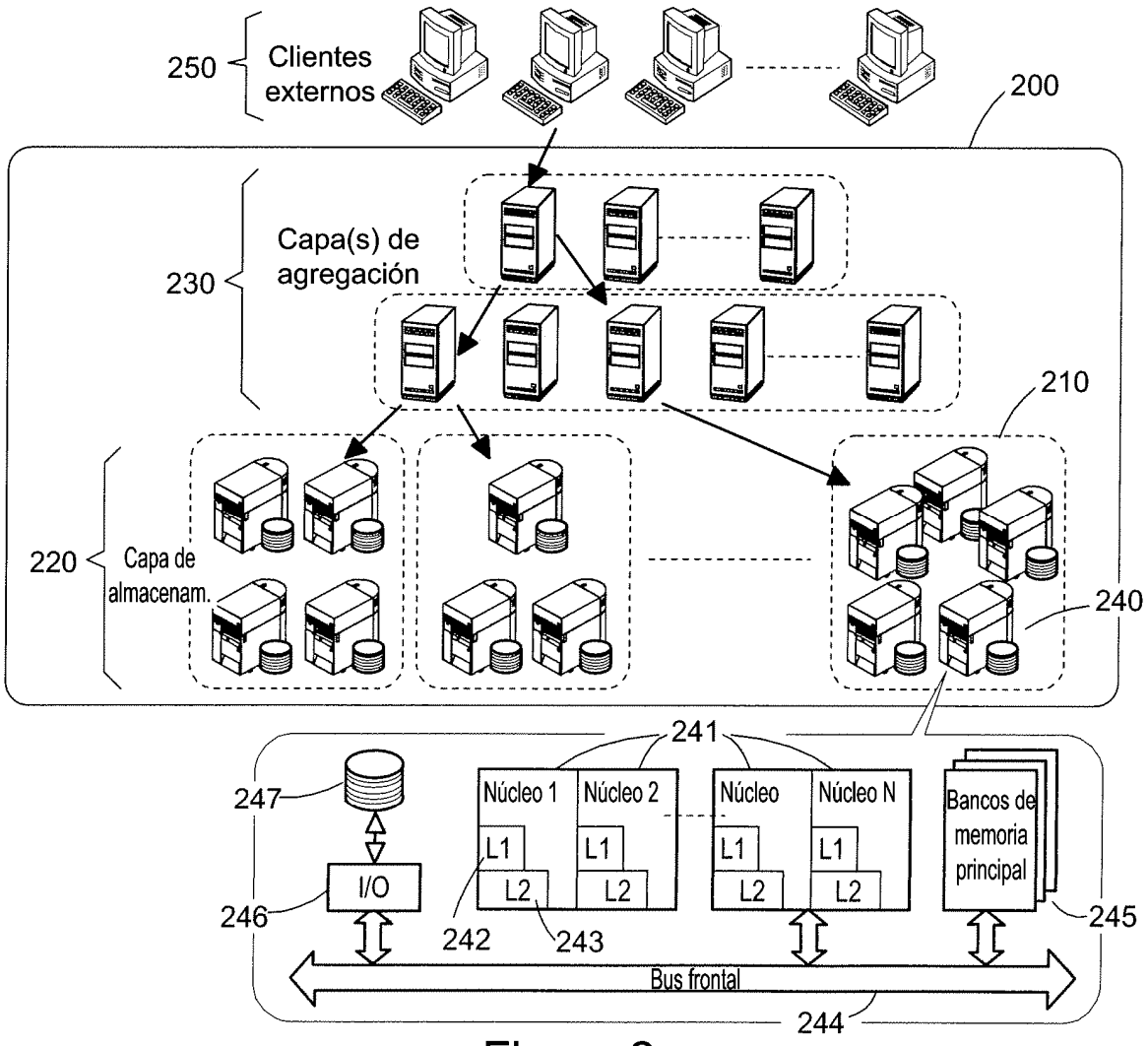

Figura 2

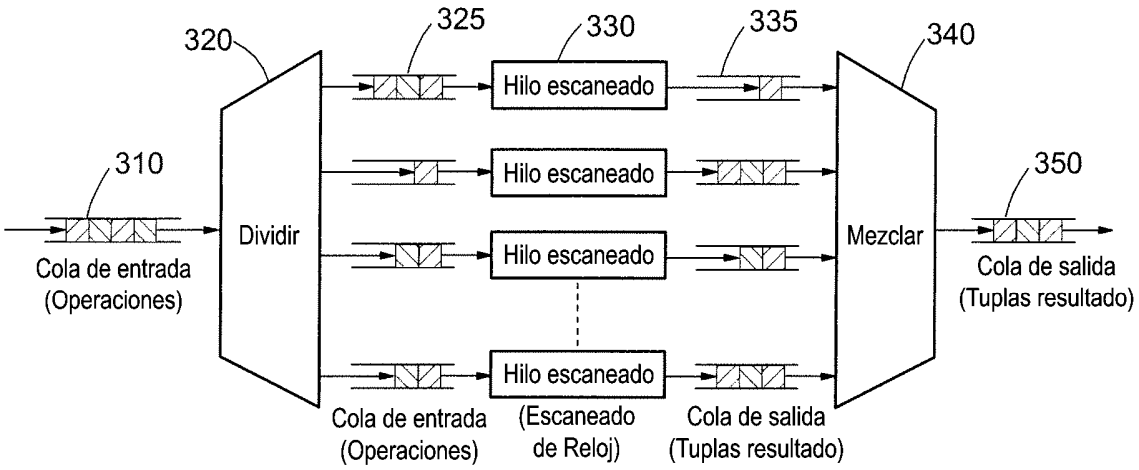

Figura 3

410 Técnica anterior

Algoritmo 1: Hilo de Escaneado Clásico

```
Datos: Segmento seg
Datos: ColaOp iq;                    // consulta de entrada y de actualización
Datos: ColaResultado oq;                                  // cola de salida
while verdadero do
    Op op ← iq.get();                          // activar una operación simple
    // escanea todo el segmento, por ranuras
    foreach Ranura s ∈ seg do Ejecutar(op, s, oq)
    Poner(oq, FinDeFlujo(op));                 // desactivar la operación
```

420 Técnica anterior

Algoritmo 2: Hilo de Escaneado Elevador

```
Datos: Segmento seg
Datos: ColaOp iq;                    // consulta de entrada y de actualización
Datos: ColaOp aq;                      // consulta activa y de actualización
Datos: ColaResultado oq;                                  // cola de salida
while verdadero do
    // escanea todo el segmento, por ranuras
    foreach Ranura s ∈ seg do
        //ejecutar todas las operaciones activas contra la ranura
        foreach Op op ∈ aq do Ejecutar(op, s, oq)
        //desactivar todas operaciones que acaben un escaneo total
        while Acabado(Recoger(aq)) do
            Poner(oq, FinDeFlujo(Obtener(aq)))
        //activar todas las operaciones en la cola de entrada
        while EstaVacio(iq) do Poner(aq, Obtener(iq))
```

Figura 4a

430

Algoritmo 3: Hilo de Escaneado de Reloj

```
Datos: OptimizadorMultiConsulta opt
Datos: Segmento seg
Datos: ColaOp iqq, iuq;        // consulta de entrada y de actualización
Datos: ColaResultado oq;                                  // cola de salida
while verdadero do
    // activa todas las actualizaciones en cola de actualiz. de entrada
    ConjuntoActualiz us ← 0
    while ¬ EstaVacio(iuq) do Poner(us, Obtener(iuq))
    //activar todas las consultas en cola de consultas de entrada
    ConjuntoConsultas qs ← 0
    while ¬ EstaVacio(iqq) do Poner(qs, Obtener(iqq))
    //realizar optimización multiconsulta
    PlanActuliz up ← ActulizPlan(opt, us)
    PlanConsulta qp ← ConsultaPlan(opt, qs)
    //escanear todo el segmento, por fragmentos
    foreach Fragmento c ∈ seg do
        Junta(up, c, oq);                    // junta de actulización-datos
        Junta(qp, c, oq);                        // junta de consulta-datos
    //desactiva todas las operaciones activas
    foreach Op op ∈ qs ∪ us do Poner(oq, FinDeFlujo(op))
```

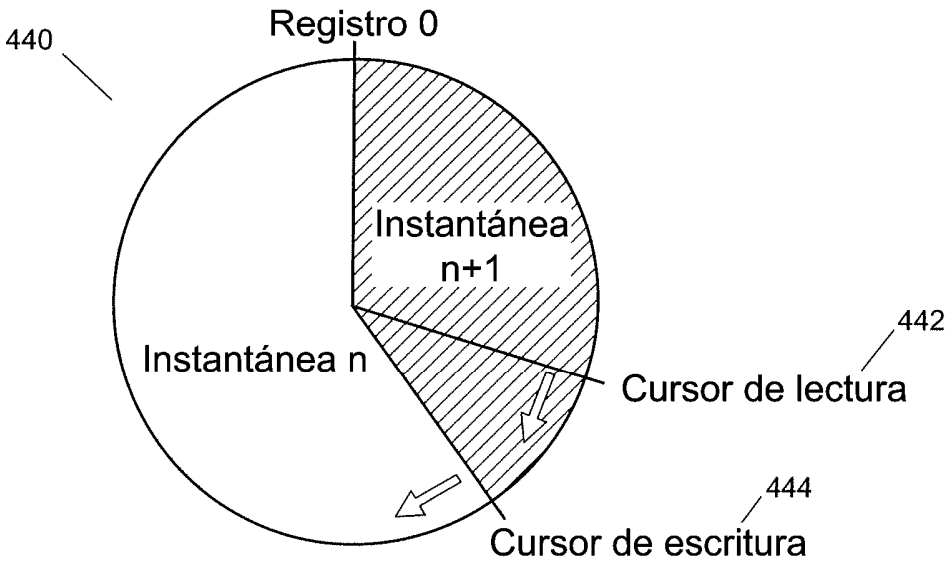

Figura 4b

510

Algoritmo 4: Índice Unión Junta

Entrada: Fragmento c
Entrada: ConjuntoIndice is; // *índices de predicado*
Entrada: ConjuntoConsulta qs; // *consultas sin indexar*
Entrada: ColaResultado oq; // *cola de salida*
foreach Registro $r \in c$ **do**
  // *sondea los índices para los candidatos*
  foreach Índice $i \in is$ **do**
    ConjuntoConsulta $C \leftarrow$ `Sondea(`i, r`)`; // *colas de candidatos*
    foreach Consulta $q \in C$ **do** `Ejecutar(`q, r, oq`)`
  //*ejecutar consultas sin indexar*
  foreach Consulta $q \in qs$ **do** `Ejecutar(`q, r, oq`)`

520

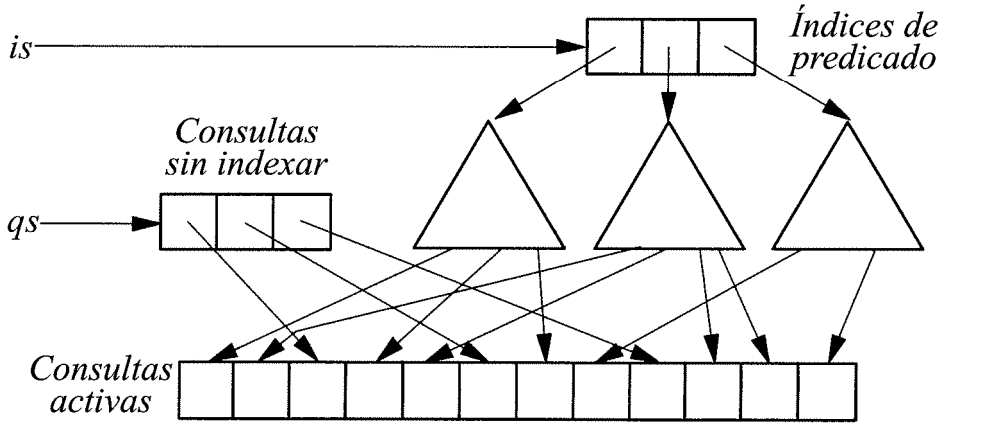

Figura 5a

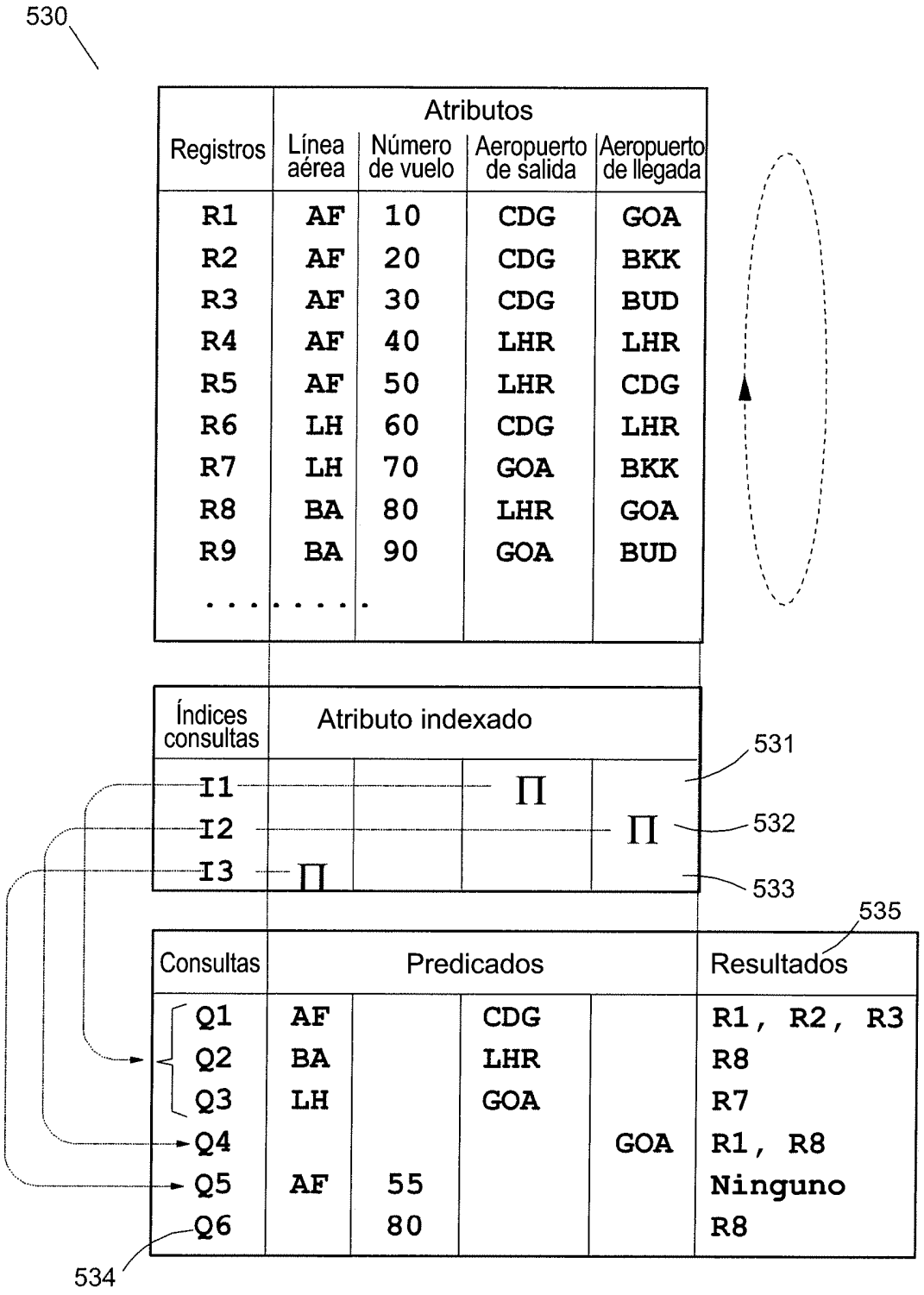
530
Registros
Atributos
Línea aérea
Número de vuelo
Aeropuerto de salida
Aeropuerto de llegada
R1 AF 10 CDG GOA
R2 AF 20 CDG BKK
R3 AF 30 CDG BUD
R4 AF 40 LHR LHR
R5 AF 50 LHR CDG
R6 LH 60 CDG LHR
R7 LH 70 GOA BKK
R8 BA 80 LHR GOA
R9 BA 90 GOA BUD
Índices consultas
Atributo indexado
I1
I2
I3
Π
531
532
533
535
Consultas
Predicados
Resultados
Q1 AF CDG R1, R2, R3
Q2 BA LHR R8
Q3 LH GOA R7
Q4 GOA R1, R8
Q5 AF 55 Ninguno
Q6 80 R8
534

Figura 5b

Algoritmo 5: Índice Unión Actualización Junta — 540

```
Entrada: Fragmento c
Entrada: ConjuntoIndice is;                    // índices de predicado
Entrada: ConjuntoActualiz us;   // UPDATE, DELETE sin indexar
Entrada: ColaInserción iq;                     // INSERT
Entrada: ColaResultado oq;                     // cola de salida
foreach Ranura s ∈ c do
    MarcaTiempo t ← 0
    while t < ∞ do
        if EstaOcupado(s) then
            //la ranura está ocupada; realizar actulizaciones
            t ← RealizActualiz(s, t, is, us)
        else if ¬ EstaVacio(iq) then
            //ranura no ocupada, tiene inserto; ejecutarlo
            Inserto i ← Obtener(iq)
            Ejecutar(i, s)
            t ← i.marcatiempo + 1
        else t ← ∞;          // no ocupado, no inserción; sig. ranura
```

Function RealizActualiz (Ranura s, Marcatiempo t, ConjÍndice is, ConjuntoActualiz us) : Marcatiempo — 550

```
ConjuntoActualiz C ← us;                       // conjunto candidatos
ConjuntoActualiz M ← 0;                        // conjunto coincidente
//índices de sondeo para candidatos adicionales
foreach Índice i ∈ is do C ← C ∪ Sondeo(i, s.registro)
//hallar coincidencias entre candidatos
foreach Actualiz u ∈ C do
    if Coincide(u, s.registro) then M ← M ∪ {u}
if M ≠ 0 then
    //ejecutar coincidencia con marcatiempos más baja
    Actualiz u ← min_{u.marcatiempo}{u ∈ M}
    if u.marcatiempo ≥ t then
        Ejecutar(u, s)
        if EstaBorrado(u) then
            return u.marcatiempo + 1;          //ranura vacía; volver
        else
            //ranura actualizada; repetición
            return RealizActualiz(s, u.marcatiempo + 1, is, us)
return ∞
```

Figura 5c

610

Algoritmo 7: Optimizador multi-consulta

Datos: Ganancia *umbr* // *umbral de ganancia mínimo*
Entrada: ConjuntoOp *os*; // *activar consultas/actualizaciones*
Entrada: ConjuntoAtributos A; // *atributos indexables en esquema*
Salida: ConjuntoIndices $is \leftarrow 0$; // *índices de predicados*
ConjuntoConsultas $uos \leftarrow os$; // *consultas/actualiz. no indexadas*
repeat
  Atributo $a \leftarrow \max_{\texttt{Ganancia}(a)} \{a \in A\}$
  Ganancia $g \leftarrow \texttt{Ganancia}(a)$
  if $g > umbr$ **then**
    Índice $idx \leftarrow \texttt{ConstrIndice}(a, uos)$
    $is \leftarrow is \cup idx$
    $uos \leftarrow uos \setminus \{q \in idx\}$
    $A \leftarrow A \setminus \{a\}$
until $g < umbr$

620

$$ganancia(Q,\ a) := \sum_{q \in Q} 1 - selectividad(a,\ q)$$

Figura 6

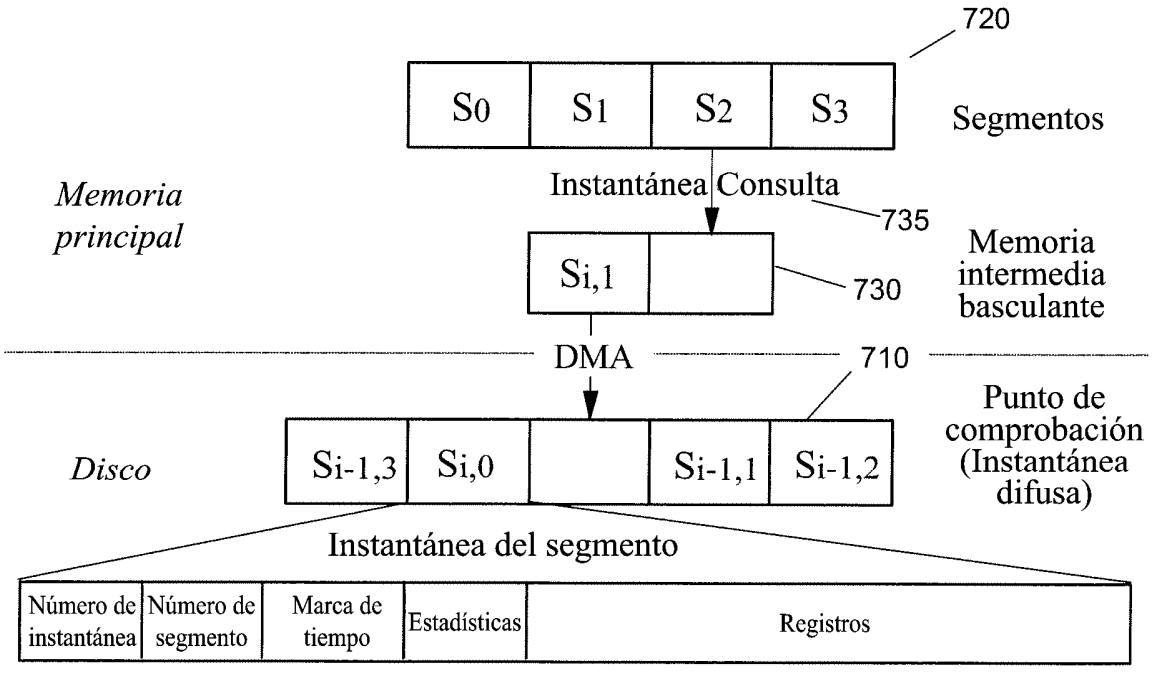

Figura 7

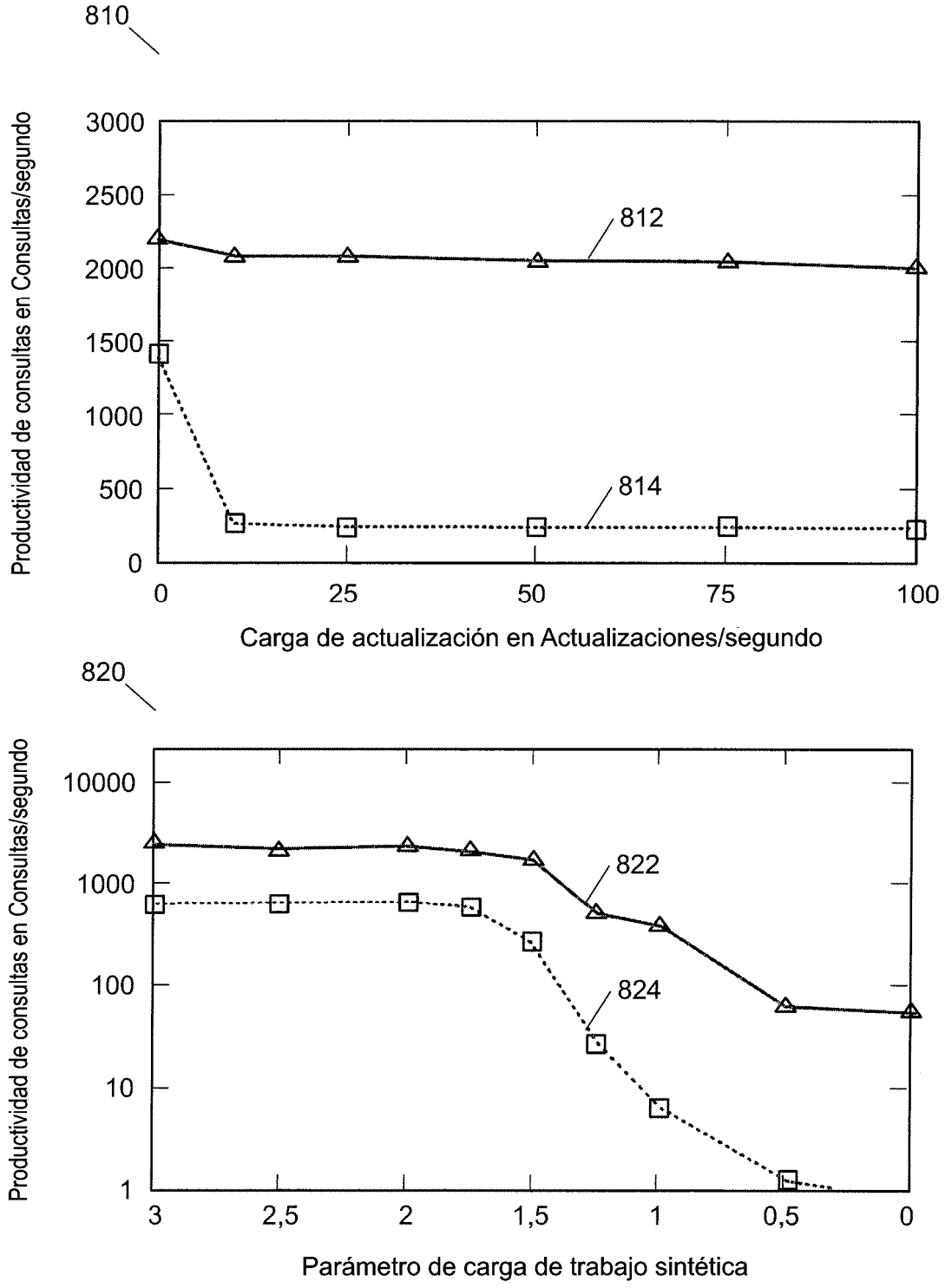
810
Productividad de consultas en Consultas/segundo
3000
2500
2000
1500
1000
500
0
812
814
0
25
50
75
100
Carga de actualización en Actualizaciones/segundo
820
Productividad de consultas en Consultas/segundo
10000
1000
100
10
1
822
824
3
2,5
2
1,5
1
0,5
0
Parámetro de carga de trabajo sintética

Figura 8