(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 626 026**

(51) Int. CI.:

G06F 12/08 (2006.01)
G06F 17/30 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **23.12.2011 PCT/US2011/067293**

(87) Fecha y número de publicación internacional: **05.07.2012 WO12092213**

(96) Fecha de presentación y número de la solicitud europea: **23.12.2011 E 11854263 (8)**

(97) Fecha y número de publicación de la concesión europea: **08.03.2017 EP 2659378**

(54) Título: **Indexación de superficie ram rápida y lenta para deduplicación de datos**

(30) Prioridad:

28.12.2010 US 979669

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
21.07.2017

(73) Titular/es:

MICROSOFT TECHNOLOGY LICENSING, LLC (100.0%)
One Microsoft Way
Redmond, Washington 98052-6399, US

(72) Inventor/es:

SENGUPTA, SUDIPTA;
DEBNATH, BIPLOB;
LI, JIN;
DESAI, RONAKKUMAR N. y
OLTEAN, PAUL ADRIAN

(74) Agente/Representante:

CARPINTERO LÓPEZ, Mario

Aviso: En el plazo de nueve meses a contar desde la fecha de publicación en el Boletín Europeo de Patentes, de la mención de concesión de la patente europea, cualquier persona podrá oponerse ante la Oficina Europea de Patentes a la patente concedida. La oposición deberá formularse por escrito y estar motivada; sólo se considerará como formulada una vez que se haya realizado el pago de la tasa de oposición (art. 99.1 del Convenio sobre Concesión de Patentes Europeas).

DESCRIPCIÓN

Indexación de superficie ram rápida y lenta para deduplicación de datos

Antecedentes

La deduplicación de datos (a veces denominada como optimización de datos) es una tendencia reciente en los sistemas de almacenamiento y, en general, se refiere a reducir la cantidad física de bytes de datos que deben almacenarse en disco o transmitirse a través de una red sin comprometer la fidelidad o la integridad de los datos originales, es decir, la reducción en bytes es sin pérdida y los datos originales pueden recuperarse completamente. Reduciendo los recursos necesarios para almacenar y/o transmitir los datos, la deduplicación de datos conduce de este modo a ahorros en costes de hardware (para almacenamiento y transmisión de red) y en costes de gestión de datos (por ejemplo, copia de seguridad). A medida que crece la cantidad de datos almacenados digitalmente, estos ahorros de costes se vuelven significativos.

La deduplicación de datos usa normalmente una combinación de técnicas para eliminar la redundancia dentro y entre los archivos almacenados de manera persistente. Una técnica opera para identificar regiones idénticas de datos en uno o múltiples archivos, y almacenar físicamente solo una única región (fragmento), mientras que se mantiene un puntero a ese fragmento en asociación con el archivo. Otra técnica consiste en mezclar la deduplicación de datos con la compresión, por ejemplo, almacenando fragmentos comprimidos para cada fragmento único.

Con el fin de identificar los fragmentos, el servidor que almacena los fragmentos mantiene un servicio de índice de hash para las hashes de los fragmentos en el sistema. La hash identifica únicamente el fragmento y sirve como la clave de un par clave - valor. El valor corresponde a la localización del fragmento en un almacén de fragmentos.

Debido a que los sistemas de deduplicación contemporáneos pueden necesitar escalar de decenas de terabytes a petabytes de volumen de datos, el índice de hash del fragmento es demasiado grande para caber en un dispositivo de almacenamiento primario (es decir, una RAM). Por lo tanto, necesita usarse un dispositivo de almacenamiento secundario, tal como una unidad de disco duro o unidad de estado sólido. Por lo tanto, las operaciones de índice están limitadas en rendimiento por las operaciones de E/S relativamente lentas ejecutadas en el dispositivo de almacenamiento secundario. Lo que se necesita es una forma de reducir los tiempos de acceso de E/S tanto como sea posible dado los recursos de almacenamiento primario limitados.

El documento US 6.928.526 B1 se refiere a un sistema de almacenamiento de datos que elimina eficazmente la redundancia. Un motor de verificación de redundancia de segmentos accede a una caché que almacena la información de segmentos para verificaciones preliminares rápidas. La información de segmento incluye el ID del segmento, los metadatos de segmento, los datos de segmento o combinaciones de los mismos. La caché se implementa normalmente usando memoria que es rápidamente accesible, tal como diversos tipos de memoria dinámica de acceso aleatorio. El motor de comprobación de redundancia de segmentos también accede a un resumen que se implementa en memoria, usado para determinar si un segmento se ha almacenado anteriormente. Si las verificaciones preliminares no determinan de manera concluyente si el segmento ya se ha almacenado, entonces se realiza una búsqueda en una base de datos de segmentos para confirmar si el segmento se ha almacenado anteriormente. La base de datos de segmentos se almacena normalmente en una memoria de latencia relativamente alta.

Sumario

El objeto de la presente invención es mejorar la velocidad de las técnicas de la técnica anterior.

Este objeto se resuelve con el objeto de las reivindicaciones independientes.

Las realizaciones preferidas se definen en las reivindicaciones dependientes.

Este resumen se proporciona para introducir una selección de los conceptos representativos en una forma simplificada que se describen más adelante en la descripción detallada. El presente resumen no pretende identificar las características clave o las características esenciales del objeto reclamado, ni está destinado a usarse de ninguna manera que limite el ámbito del objeto reclamado.

Brevemente, diversos aspectos del objeto descrito en el presente documento se dirigen hacia una tecnología de deduplicación mediante la que un índice del servicio de índice de hash está configurado para mantener un índice basado en un registro en el dispositivo de almacenamiento secundario y mantener una tabla de índices compactos basada en un almacenamiento primario (por ejemplo, una RAM) y/o una caché de búsqueda anticipada basada en una RAM que se accede para reducir el acceso de E/S del dispositivo de almacenamiento secundario.

La tabla de índices compactos basada en una RAM contiene una firma compacta y un puntero que se usa para mapear un valor de hash proporcionado en una solicitud a una localización del índice en el dispositivo de almacenamiento secundario en el que se mantiene (posiblemente) ese índice de hash. Si no se encuentra una hash

en la tabla de índices compactos, entonces la hash corresponde a un nuevo fragmento que todavía no se ha indexado y se devuelve un resultado de no encontrado en respuesta a la solicitud. De lo contrario, se sigue el puntero para localizar el valor de hash, para devolver los metadatos de fragmento en respuesta a la solicitud; obsérvese que el valor de hash solicitado se compara a través del valor de hash encontrado en el índice a través del puntero en una alta probabilidad del tiempo, pero puede no encontrarse, en cuyo caso se devuelve un resultado de no encontrado en respuesta a la solicitud.

Además, puede usarse una caché de búsqueda anticipada basada en una RAM. Siempre que se encuentre un valor de hash coincidente en el dispositivo de almacenamiento secundario, algún número k del valor de hash, se cargan las entradas de metadatos en la caché de búsqueda anticipada. El número obtenido para cargar en la caché puede corresponder a un límite de archivo, por ejemplo, detectado en el índice de hash puesto allí al final de cada uno de los fragmentos del archivo.

En un aspecto, las entradas del índice de hash estructurado en registros comprenden unos valores de hash y unos metadatos asociados con cada valor de hash. El índice de hash se actualiza añadiendo nuevas entradas al índice de hash estructurado en registros. Durante la anexión, las entradas pueden estar dispuestas en una o más páginas de entradas, tal como por razones de optimización de E/S. El índice de hash puede actualizarse con datos de una caché de sesión, que representa las entradas correspondientes a los archivos procesados en la última sesión de optimización. También puede accederse a la caché de sesión para buscar metadatos de fragmentos asociados con un valor de hash solicitado. Al final de cada sesión de deduplicación periódica, los contenidos de la caché de sesión se transfieren transaccionalmente al índice de hash principal. De esta manera, el índice de hash permanece "consistente al fallo" durante la duración de la sesión de deduplicación, es decir, si hay un fallo o una interrupción repentina del servicio que se produce durante la sesión de deduplicación, el almacenamiento secundario permanece intacto y se recuperará al inicio de la próxima sesión de deduplicación.

En otro aspecto, la tabla de índices compactos puede corresponder a un subespacio que comprende un subconjunto más pequeño de la tabla de índices de registro. Si el subespacio está lleno, se puede sellar (solo lectura) y la tabla de índices compactos corresponde a un modo de solo lectura. Un codificador puede acceder al subespacio para codificar el índice compacto en una tabla de índices compactos de tamaño de memoria reducido.

En un aspecto, las firmas compactas se insertan en la tabla de índices compactos usando un algoritmo basado en una hash de cuco para evitar colisiones de hash. Si el algoritmo basado en una hash de cuco no encuentra un intervalo disponible para una firma compacta, que puede ser porque no existe, o porque el número de intentos permitidos está limitado, la firma compacta puede escribirse en una tabla de desbordamiento.

Otras ventajas pueden resultar evidentes a partir de la siguiente descripción detallada cuando se toman junto con los dibujos.

Breve descripción de los dibujos

La presente invención se ilustra a modo de ejemplo y no está limitada por las figuras adjuntas, en las que los números de referencia similares indican elementos similares y en los que:

La figura 1 es un diagrama de bloques que representa unos componentes de ejemplo de un servicio de almacenamiento de datos configurado para una deduplicación usando un servicio de índice de hash configurado con una tabla de índices compactos y una caché de búsqueda anticipada.
La figura 2 es un diagrama de bloques que representa cómo puede usarse una tabla de desbordamiento eficientemente o cuando la tabla de índices compactos está llena.
La figura 3 es un diagrama que representa unas operaciones de ejemplo realizadas por un servicio de índice de hash para manejar una operación de búsqueda de un valor de hash solicitado, incluso cuando se encuentra una firma compacta para el valor de hash en la tabla de índices compactos.
La figura 4 es un diagrama que representa unas operaciones de ejemplo realizadas por un servicio de índice de hash para manejar una operación de búsqueda para un valor de hash solicitado, incluso cuando no se encuentra una firma compacta para el valor de hash en la tabla de índices compactos.
La figura 5 es un diagrama de flujo que representa las etapas de ejemplo para manejar una solicitud para devolver los metadatos o una respuesta de no encontrado dado un valor de hash.
La figura 6 es un diagrama de bloques que representa unos entornos de red no limitativos a modo de ejemplo en los que pueden implementarse diversas realizaciones descritas en el presente documento.
La figura 7 es un diagrama de bloques que representa un sistema informático no limitante o un entorno operativo en el que pueden implementarse uno o más aspectos de las diversas realizaciones descritas en el presente documento.

Descripción detallada

Diversos aspectos de la tecnología descrita en el presente documento se dirigen, en general, hacia un servicio de índice de hash que usa un índice de hash completo (por ejemplo, para un sistema de deduplicación de almacenamiento) en un dispositivo de almacenamiento secundario (por ejemplo, una unidad de disco duro o una unidad de estado sólido) como un índice estructurado en registros, manteniendo al mismo tiempo un índice auxiliar

relativamente pequeño denominado como una “tabla de índices compactos” en el dispositivo de almacenamiento primario (por ejemplo, una RAM). También puede usarse una caché, denominada caché de búsqueda anticipada, en el almacenamiento primario. En general, la tabla de índices compactos y la caché de búsqueda anticipada funcionan para reducir los accesos de E/S al dispositivo de almacenamiento secundario durante las operaciones de deduplicación tanto como sea posible para lograr un alto rendimiento de deduplicación, al tiempo que tienen una superficie de almacenamiento primario relativamente bajo. Al mismo tiempo, el diseño del índice adapta el uso de diversos tipos de medios de almacenamiento secundario más rápidos, como la memoria flash.

En un aspecto, un índice de hash global para un servicio de deduplicación de datos (que puede corresponder a todo el índice de hash o a un conjunto de índices de hash para múltiples subespacios como se describe en la solicitud de patente de Estados Unidos US 2012/0166401 A1 titulada “Using Index Partitioning and Reconciliation for Data Deduplication”) se almacena en el dispositivo de almacenamiento secundario. Los índices se almacenan en un orden basado en un registro, basado en el orden en el que se depositan y/o se hacen referencia. La caché de búsqueda anticipada está asociada con el índice de hash y cuando se identifica un acierto de hash, las entradas de hash cercanas en el archivo de registro se cargan en la caché de búsqueda anticipada. Como se comprenderá, esto reduce la cantidad de E/S necesaria para realizar las operaciones de búsqueda de hash posteriores.

Además, se crea una tabla de índices de memoria compactos para ayudar en la búsqueda de las hashes en el dispositivo de memoria secundaria. Como se describe a continuación, la tabla de índices compactos comprende una firma de hash truncada (por ejemplo, una suma de comprobación) a la que puede accederse para determinar cuándo no existe una hash en el índice estructurado en registros o para determinar que una hash puede existir posiblemente en el índice estructurado en registros, en cuyo caso la entrada compacta identifica además una localización en el índice estructurado en registros por la que puede encontrarse esa hash si realmente existe. Normalmente, hay suficiente información en la firma de índices truncados de tal manera que el porcentaje de falsos positivos (es decir, la hash puede existir pero en realidad no lo hace) es relativamente bajo.

Debería entenderse que cualquiera de los ejemplos del presente documento no son limitativos. De hecho, la tecnología descrita en el presente documento se aplica a cualquier tipo de almacenamiento volátil o no volátil, incluyendo local y remoto. Además, los mecanismos de indexación pueden usarse con cualquier clave, almacenamiento de tipo valor, y no se limitan a las aplicaciones de deduplicación de datos. Como tal, la presente invención no se limita a ninguna realización, aspectos, conceptos, estructuras, funcionalidades o ejemplos específicos descritos en el presente documento. Más bien, cualquiera de las realizaciones, aspectos, conceptos, estructuras, funcionalidades o ejemplos descritos en el presente documento son no limitantes y la presente invención puede usarse de diversas maneras que proporcionen beneficios y ventajas en la informatización y la indexación y recuperación de datos en general.

La figura 1 muestra unos componentes de ejemplo de un sistema de almacenamiento de datos de deduplicación de datos sensible al contenido, tal como se implementa en un servicio 102 de almacenamiento de datos/archivos. El servicio 102 recibe los datos 104 (un archivo, paquete discreto de datos binarios o similares) y la lógica 106 de deduplicación procesa los datos para la deduplicación; obsérvese que los datos y/o los servicios pueden residir esencialmente en cualquier lugar, por ejemplo, localmente, remotamente y/o en un escenario de “deduplicación en la nube”. Para procesar los datos, la lógica 106 de deduplicación proporciona los datos 104 a un módulo 108 de fragmentación, que procesa el contenido en fragmentos, tal como de acuerdo con la estructura del archivo (por ejemplo, una partición de un archivo de medios en un encabezado de medios y un cuerpo de medios), o usando una hash débil calculable rápidamente (por ejemplo, una hash de Rabin) que se calcula en cada intervalo pequeño (generalmente para cada byte). En general, se determina un límite de fragmentos de una manera dependiente del contenido en posiciones para las cuales la hash satisface una cierta condición. La siguiente descripción es con respecto a un fragmento 110, aunque se entiende que los datos se dividen normalmente en múltiples fragmentos. Obsérvese que “dependiente del contenido” se refiere, en general, al concepto de que las modificaciones o los desplazamientos añadidos a un archivo solo hacen cambios locales en los fragmentos generados, por ejemplo, debido a que la huella digital de Rabin se genera a partir del propio contenido (y no a que el fragmento es necesariamente sensible al formato del archivo con el fin de fragmentarse de una manera sensible al formato).

La lógica 106 de deduplicación pasa el fragmento 110 a un mecanismo 112 de hash, que calcula una hash del fragmento, denominado como función 114 de troceo de fragmento. Una hash fuerte, por ejemplo, una hash SHA-256 criptográficamente segura o similares puede usarse como la función 114 de troceo de fragmento que identifica de manera única el fragmento 110. Obsérvese que con una hash seguro de este tipo, la probabilidad de una colisión de hash es despreciable, por ejemplo, una colisión de hash es de alrededor de treinta órdenes de magnitud menos probable que un error de hardware dado el hardware más confiable actualmente disponible.

La función 114 de troceo de fragmento se proporciona a un servicio 116 de índice de hash, que busca la hash de fragmento a través de una o más varias estructuras como se describe en el presente documento. Si se encuentra la función 114 de troceo de fragmento (es decir, ya existe) en el servicio 116 de índice de hash, se considera que ya se ha depositado una copia duplicada del fragmento 110 en el almacén 118 de fragmentos y no es necesario que el fragmento actual se almacene adicionalmente. En su lugar, cualquier referencia a este fragmento puede referirse simplemente al fragmento anterior existente.

Si no se encuentra la función 114 de troceo de fragmento en el servicio 116 de índice de hash, el fragmento 110 se deposita en el almacén 118 de fragmentos y la función 114 de troceo de fragmento se deposita en el servicio 116 de índice de hash. Como puede apreciarse fácilmente, dados suficientes datos a lo largo del tiempo, puede guardarse una gran cantidad de almacenamiento haciendo referencia a un fragmento en lugar de mantener muchas instancias separadas del mismo fragmento de datos. Los fragmentos a menudo también se comprimen, ahorrando aún más almacenamiento.

La tecnología descrita en el presente documento está dirigida hacia una arquitectura y unos algoritmos del servicio 116 de índice de hash, y más específicamente hacia el mantenimiento de una memoria 120 caché de búsqueda anticipada y una tabla 122 de índices compactos en un almacenamiento 124 primario y en un índice 126 estructurado en registros en un almacenamiento 128 secundario. Obsérvese que el índice 126 estructurado en registros puede ser un único índice global, o puede dividirse en múltiples índices, tal como en un servicio de índice de hash basado en un subespacio, donde un subespacio es una parte más pequeña del índice global del sistema general, como se describe en la solicitud de patente de Estados Unidos mencionada anteriormente titulada “Using Index Partitioning and Reconciliation for Data Deduplication”.

En general, el uso de un índice estructurado en registros permite que se agrupe un número de entradas de índice (por ejemplo, en una memoria intermedia o en una caché de sesión como se describe a continuación) para añadir al índice de registro, por ejemplo, basándose en alcanzar un cierto tamaño, periódicamente o según se necesite, con el fin de reducir el número de operaciones de E/S frente a insertar cada entrada individualmente en el índice de registro. Esto también facilita el uso de dispositivos de almacenamiento de estado sólido tales como, por ejemplo, los dispositivos de memoria basados en flash donde los datos se escriben en unidades de páginas y donde la escritura de datos en forma de estructura de registro es más eficiente.

La tabla 122 de índices compactos asigna cada hash a una localización del índice en el dispositivo 128 de almacenamiento secundario en el que se mantiene (posiblemente) ese valor de hash. Además, se usa la memoria 120 caché de búsqueda anticipada, de manera que cuando se confirma un acierto en el dispositivo 128 de almacenamiento secundario, un número de k índices de hash, el vecino del acertado se carga en la memoria 120 caché de búsqueda anticipada. El número de vecinos puede determinarse por algún número fijo, o puede corresponder a un límite de archivo. Un límite de archivo para seleccionar vecinos es útil en muchos escenarios debido a que los fragmentos de un archivo se indexan inicialmente en orden y, por lo tanto, cualquier archivo similar probablemente tendrá algunos o todos los fragmentos y, en consecuencia, las hashes de ese vecino de una hash de fragmento dado. Obsérvese que el índice estructurado en registros puede configurarse para incluir un marcador de fin de archivo para delinear las hashes correspondientes a los fragmentos de cada archivo distinto, por ejemplo, una entrada cero total; (a pesar de que una hash de fragmento puede ser todos ceros, el identificador de fragmento (id de localización) puede definirse como distinto de cero de tal manera que una entrada cero total es inválida y puede servir como un marcador de fin de archivo en el índice de registro).

La memoria 120 caché de búsqueda anticipada se basa en el concepto de localidad en el que los fragmentos deduplicados suelen ser vecinos entre sí en el archivo de origen y en el lugar de destino (por ejemplo, el almacén de fragmentos y, en consecuencia, el índice). Cuando esto ocurre y se alcanza un valor de hash en el dispositivo 128 de almacenamiento secundario, existe una probabilidad razonable de que el siguiente fragmento de hash o de hashes sea vecino del fragmento anterior, por lo que dichos vecinos se cargan en la caché donde pueden alcanzarse en una búsqueda de caché rápida, en lugar de en una búsqueda de índice secundario más lenta.

Obsérvese que la memoria 120 caché de búsqueda anticipada no necesita ser totalmente reactiva en su uso, sino que puede cargarse de manera proactiva con valores de hash del índice estructurado en registros, tal como basado en estadísticas u otros datos tales como unas cuentas mantenidas por un sistema. Por ejemplo, la memoria 120 caché de búsqueda anticipada puede estar pre-poblada con las hashes más populares (en general y/o calculados en relación con alguna ventana de tiempo, tal como los últimos treinta días) de acuerdo con lo rastreado por el sistema de deduplicación. Como alternativa, o además de las hashes de fragmentos más populares, la memoria 120 caché de búsqueda anticipada puede estar pre-poblada con las hashes usadas más recientemente, sobre la base de que los mismos fragmentos a menudo tienden a concentrarse juntos en el tiempo. De este modo, la memoria 120 caché de búsqueda anticipada pre-busca las hashes contando con la localidad de fragmento y también puede almacenar hashes de los fragmentos más populares y/o más frecuentemente usados para reducir aún más la probabilidad de acceso y búsqueda en disco durante las operaciones de búsqueda.

Pueden usarse técnicas de gestión de caché bien conocidas (por ejemplo, LRU o SLRU) para desalojar las hashes cuando se agregan unas nuevas por la técnica de carga de hash de búsqueda anticipada/de vecindad descrita anteriormente. Obsérvese que por eficiencia, la carga y el desalojo de hash con respecto a la memoria 120 caché no necesitan estar dispuestos para manejar una sola hash a la vez, y en su lugar pueden cargarse/desalojarse en unidades mayores, tal como una página de entradas de hash a la vez.

Para resumir hasta ahora, en un ejemplo de implementación, el servicio 116 de índice de hash se usa para soportar un sistema de deduplicación de datos, en el que los elementos de datos comprenden pares de hashes de metadatos y en el que cada una de las hashes de un fragmento de datos se usa para determinar si ese fragmento es un duplicado de otro fragmento de datos. Los metadatos son un identificador de fragmentos que localizan el fragmento

en el almacén de fragmentos. El par de hashes de metadatos se almacena principalmente en un índice 126 estructurado en registros en el dispositivo 128 de almacenamiento secundario.

Con respecto a la tabla 122 de índices compactos, se mantiene una versión truncada de la hash en el dispositivo de almacenamiento primario como una tabla de índices compactos. Como se ha descrito anteriormente, en general, la tabla 122 de índices compactos almacena firmas de clave compactas (hash truncada) en lugar de hashes de fragmentos completos, con el fin de establecer compensaciones entre el uso de RAM y las lecturas falsas.

Para aproximarse a maximizar la capacidad de índice de tabla de hash (el número de entradas) mientras se minimizan las lecturas de flash falsas, una implementación almacena una firma compacta de clave de longitud de M bit/byte (del orden de unos pocos bytes, por ejemplo, dos bytes) en cada entrada de la tabla 122 de índices compactos. Esta firma puede obtenerse tanto de la clave como del número de posición candidata en el que se almacena la clave. Cuando una clave x se almacena en su número de posición candidata *i*, la firma en el intervalo de índice de tabla de hash respectiva se obtiene a partir de los bits de orden superior del valor de hash $_{hi}$(x). Durante una operación de búsqueda, cuando se busca una clave *y* en su número de intervalo candidato *j*, la firma respectiva se calcula a partir de $_{hj}$(y) y se compara con la firma almacenada en ese intervalo. Solo si se produce una coincidencia, el puntero sigue al índice 126 estructurado en registros para comprobar si la clave completa (hash completa) coincide. El porcentaje de lecturas falsas es relativamente bajo, por ejemplo, el número de intervalos candidatos dividido por el espacio de firma (que para un espacio de firma de dos bytes es 65536).

El tamaño del almacenamiento primario (por ejemplo, una RAM) para la tabla 122 de índices compactos puede determinarse teniendo en cuenta los requisitos de la aplicación. Con una firma de clave compacta de dos bytes y un puntero de índice de registro de N-bits / byte de longitud (por ejemplo, igual a cuatro bytes), lo que es un total de seis bytes por entrada, un uso típico de RAM de 4 GB por máquina para el índice de tabla de hash aloja un máximo de aproximadamente 715 millones de entradas fragmento-hash. Con un promedio de 8 KB de tamaño por fragmento de datos, esto aloja aproximadamente 6 TB de datos deduplicados. Como alternativa, con 64 bytes asignados para un fragmento-hash y sus metadatos, esto corresponde a aproximadamente 45 GB de metadatos de fragmento. Para la deduplicación, el sistema de deduplicación está diseñado para usar un número relativamente pequeño de bytes en RAM por entrada (por ejemplo, los seis bytes por entrada compacta) con el fin de maximizar la capacidad de índice de tabla de hash de RAM para un tamaño de uso de RAM determinado.

La tabla 122 de índices compactos puede corresponder a un modo de solo lectura y puede haber tablas de índices compactos de solo lectura si se usan subespacios, por ejemplo, un subespacio se convierte en un subespacio de solo lectura cuando ese subespacio está completo y puede convertirse en “sellado”. Como se describe a continuación, para una tabla de índices compactos de solo lectura, puede usarse un mecanismo de codificación (algoritmo) para calcular una tabla de índices compactos codificada (por ejemplo, una tabla basada en una hash truncada de cuco como se describe a continuación) a partir de las entradas existentes en el subespacio que usan aún menos espacio de memoria con una mayor tasa de llenado de tabla de hash. A continuación, puede consultarse la tabla 122 de índices compactos para localizar una hash en el subespacio, usando solo seis bytes por entrada de hash para la tabla de índices compactos en una implementación.

La tabla 122 de índices compactos puede corresponder a un modo de escritura al que pueden añadirse nuevas entradas de hash. También puede usarse una tabla de hash truncada de cuco en el modo de escritura. Para reducir la complejidad de las colisiones cuando se añaden nuevas entradas de hash a la tabla de índices compactos, puede usarse una tabla de hash de desbordamiento relativamente pequeña, como se describe a continuación.

Más específicamente, la tabla de índices compactos puede resolver las colisiones de hash mediante una hash de cuco. Como se representa en la figura 2, el índice 122 de tabla de hash compacto está estructurado como una serie de intervalos. En una implementación, las colisiones de hash, en las que múltiples firmas compactas se mapean al mismo intervalo de índice de tabla de hash, se resuelven mediante la lógica 230 de inserción de tabla usando la hash de cuco o una variante de la hash de cuco. Para este fin, la hash de cuco proporciona flexibilidad para que cada firma compacta esté en una de $n \geq 2$ posiciones; una variante de hash de cuco mantiene la secuencia de cadena de sondeo lineal superior limitada en *n* y puede dar como resultado una escritura en una tabla de desbordamiento como se describe a continuación. Obsérvese que la hash de cuco aumenta los factores de carga de tabla de hash mientras mantiene el tiempo de búsqueda limitado a una constante.

En una variante de la hash de cuco usada en una implementación, se usan *n* hashes aleatorias *h*1, *h*2;... *h*n para obtener *n* posiciones candidatas para una firma x compacta dada. Estos índices de posición candidata para la firma compacta x se obtienen a partir de los valores de bit de orden inferior de *h*1(x), *h*2(x);... *h*n(x) correspondientes a una operación de módulo.

Durante la inserción, la firma compacta y su entrada de metadatos se insertan en el primer intervalo candidato disponible. Cuando se ocupan todos los intervalos de una firma x compacta dada durante la inserción (por ejemplo, mediante las firmas compactas *y*1, *y*2,.... *y*n), puede hacerse espacio para la entrada de *x* firmas compactas recolocando las firmas compactas *yi* en estos intervalos ocupados, debido a que cada firma compacta *yi* puede colocarse en una elección de otras (*n* - 1) localizaciones. Obsérvese que en el esquema de hash de cuco original, se usa una estrategia recursiva para recolocar una de las firmas compactas *yi*; sin embargo, en el peor de los casos,

esta estrategia puede tomar muchas relocalizaciones, aunque la probabilidad de que pueda mostrarse es muy pequeña y disminuye exponencialmente en *n*.

De este modo, el procedimiento de la lógica 230 de inserción de tabla puede intentar cierto número de relocalizaciones de entrada, después de lo cual, si fracasa, puede operar de acuerdo con una serie de opciones. Una de las opciones es hacer que el servicio de deduplicación detenga la deduplicación y reconstruya la tabla de índices compactos como una tabla de índices compactos más nueva y más grande antes de reanudar la deduplicación. Otra opción es el encadenamiento lineal, que inserta / busca intervalos linealmente posteriores después de cierto número de intentos. Otra opción más, que puede evitar la sobrecarga de una reconstrucción de una tabla de índices compactos en modo de escritura, y el aumento del coste de acceso de E/S en el encadenamiento lineal, es escribir la firma compacta y la entrada de metadatos en una tabla 232 de desbordamiento. En la variante de cuco, la escritura puede ser en la tabla 232 de desbordamiento tan pronto como un pequeño número de intentos no hayan tenido éxito, para evitar los cálculos necesarios para continuar con otros intentos de la hash de cuco. Si se usa la tabla 232 de desbordamiento, la lógica 234 (o un codificador) de lectura de tabla sabe que tiene acceso a la tabla 232 de desbordamiento si no logra buscar una firma compacta en la tabla 122 de índices compactos o si la tabla de índices se sella y/o se codifica como se describe a continuación.

Volviendo a otro aspecto, una tabla de índices compactos de solo lectura puede codificarse para ahorrar espacio de memoria adicional, así como para reducir falsos positivos, tal como en un procedimiento de codificación sin conexión. Más específicamente, cuando en un modo de escritura, una tabla de índices compactos tiene que reservar un espacio de memoria extra para ayudar a la hash de cuco a resolver eficientemente las colisiones (para evitar demasiado cálculo / búsqueda de un intervalo abierto y/o el uso excesivo de una tabla de desbordamiento o similares, que agregan ineficiencia). La tasa de ocupación es, en general, del orden del ochenta al noventa por ciento. Sin embargo, una vez que se leen, se conocen todas las claves, y tal espacio extra no es necesario ya que no hay nuevas claves para insertar, y la mayor parte del espacio extra puede recuperarse (algo puede permanecer después de la codificación); la tasa de ocupación después de la codificación es, en general, del orden del noventa y cinco al noventa y ocho por ciento.

Para este fin, puede procesarse un subespacio sellado y, para poner todas las entradas en la tabla de índices compactos que indexa ese subespacio, puede generarse una gráfica bipartita para vincular cada clave a sus posiciones candidatas. Después de eliminar los conflictos, puede calcularse la codificación mediante la búsqueda de una coincidencia, como se describe en la solicitud de patente de Estados Unidos número de serie 12/725840 “Cuckoo hashing to store beacon reference data”. Obsérvese que hacer coincidir sin eliminar primero los conflictos aumentará los falsos positivos y, por lo tanto, es en general más óptimo eliminar los conflictos, lo que añade un poco de espacio de memoria adicional pero resulta en una tabla de índices compactos codificada que elimina falsos positivos.

Las figuras 3 y 4 muestran algunas de las operaciones de flujo de datos entre los diversos componentes descritos anteriormente, con las operaciones representadas por flechas etiquetadas con números circulares. En la figura 3, tal como se representa mediante la flecha uno (1), el servicio 116 de índice de hash comprueba primero una memoria 330 caché de sesión (si existe alguna), para determinar si el fragmento ya estaba determinado para ser un nuevo fragmento en esta sesión. Más concretamente, por razones de eficiencia, el servicio de deduplicación puede procesar un número de archivos como un lote (por ejemplo, en una sesión de deduplicación) y, a continuación, confirmar los fragmentos al almacén de fragmentos y el índice (hash, fragmento de metadatos) se actualiza conjuntamente. El servicio de deduplicación usa una memoria 330 caché de sesión para este fin (si dicho procedimiento por lotes se realiza en una implementación dada). Por lo tanto, existe una posibilidad de que una hash ya se haya detectado como nueva y, por lo tanto, resida en la memoria 330 caché de sesión, por lo que puede buscarse eficazmente ya que reside en el almacenamiento primario para posteriormente confirmarse con el índice 126 estructurado en registros en el almacenamiento secundario. Si se encuentra, el servicio 116 de índice de hash devuelve los metadatos para esa hash y las operaciones de búsqueda terminan.

Al final de la sesión de deduplicación, la caché de la sesión se “limpia” periódicamente en el índice principal. Esto se hace iterando a través de las hashes almacenadas en la caché de sesión y moviéndolos al índice principal, en un enfoque transaccional de todo o nada. La razón para mantener separados el índice de hash y la caché de sesión es garantizar una actualización transaccional de las estructuras de datos de índice de hash al final de cada sesión de deduplicación.

Si no se encuentra en la memoria 330 caché de sesión o no hay caché de sesión en uso, el servicio de índice de hash accede a la memoria 120 caché de búsqueda anticipada para la hash, tal como se representa mediante la flecha dos (2) y como se ha descrito anteriormente. Si se encuentra, el servicio 116 de índice de hash devuelve los metadatos para esa hash y las operaciones de búsqueda terminan.

Si no se encuentra en la memoria 120 caché de búsqueda anticipada, el servicio de índice de hash busca en la tabla de índices compactos una firma compacta del valor de hash representada por la flecha tres (3). Si no se encuentra, entonces se trata de una nueva hash, y el servicio de índice de hash devuelve un estado de “no encontrado” al servicio de deduplicación, después de lo cual el fragmento y la entrada de índice pueden manejarse en consecuencia a través de una operación de inserción, o directamente en el índice estructurado en registros (o en

una memoria intermedia para ese índice estructurado en registros, tal como un memoria intermedia basada en una página, que puede considerarse parte del índice incluso si se mantiene en el almacenamiento primario). La tabla de índices compactos se actualiza a medida que se agrega el índice de hash.

En el ejemplo de la figura 3, considérese que se encuentran una o más firmas compactas en la tabla 122 de índices compactos, y que se devuelven los punteros correspondientes al índice estructurado en registros (flecha (4a)). El servicio de índice de hash usa cada puntero (flecha cinco (5)) para buscar en el índice 126 estructurado en registros para el valor de hash completo. Si no se encuentra (todas las firmas compactas eran realmente falsos positivos), la información “no encontrada” se devuelve al servicio de deduplicación. Si se encuentra, esto es un acierto, y se devuelven los metadatos de fragmento de la hash, junto con las hashes vecinas de la hash encontrada (flecha seis (6)). Las hashes vecinas pueden añadirse a la memoria 120 caché de búsqueda anticipada (flecha siete (7)), desalojando otras entradas según sea apropiado.

En el ejemplo de la figura 4, considérese que después de acceder a la tabla de índices compactos, se encuentra que la hash no existe (flecha (4b)). En este caso, la hash es nueva, se realizará un resultado de “no encontrado” devuelto al servicio de deduplicación y una inserción posterior. Esto puede incluir una escritura en la caché de sesión (flecha ocho (8)), o una escritura en el índice estructurado en registros (o a una memoria intermedia a escribir, que puede considerarse parte del índice) en la flecha de nueve (9). Si una caché de sesión no está en uso, la tabla de índices compactos puede actualizarse (flecha diez (10)).

La figura 5 es un diagrama de flujo que muestra algo de la lógica del servicio de indexación de hash en la forma de etapas de ejemplo para determinar si un valor de hash se corresponde con un fragmento nuevo o existente. La etapa 502 comprueba la caché de sesión para la hash, si es que existe. Si se encuentra (etapa 504), el procedimiento devuelve los metadatos de fragmento de acompañamiento en la etapa 520 y termina.

Si no se encuentra en la etapa 502 no existe ninguna sesión de caché, la etapa 506 representa el acceso a la caché de búsqueda anticipada. Si se encuentra (etapa 506), el procedimiento devuelve los metadatos de fragmento de acompañamiento en la etapa 520 y termina.

La etapa 510 representa la búsqueda de la firma compacta de la hash en la tabla de firmas compactas. Si no se encuentra (etapa 512), entonces se sabe que esta hash corresponde a un nuevo fragmento, y una respuesta de no encontrado se devuelve a la etapa 522. La etapa 522 también representa otros aspectos de la manipulación de un fragmento nuevo, tal como en respuesta a una llamada de inserción al servicio de índice de hash.

Como se ha descrito anteriormente, si se encuentran una o más firmas compactas en la etapa 512, entonces se usa el puntero asociado con cada una de las firmas compactas para buscar el valor de hash completo en el índice estructurado en registros en la etapa 514 (que puede incluir un acceso a la tabla de desbordamiento o un procesamiento de cadena lineal). Si no se encuentra (etapa 516), esta entrada de firma compacta es un falso positivo. Si todas las entradas de firmas compactas (o si solo una entrada) son falsos positivos, el procedimiento devuelve un no encontrado a través de la etapa 522 para que se maneje como un nuevo fragmento.

Si la hash coincide con el índice en la etapa 516, entonces se devuelve el fragmento de metadatos a través de la etapa 520. Además, se obtienen las hashes vecinas (etapa 518) y se agregan a la caché de búsqueda anticipada.

Como puede verse, el uso de un índice estructurado en registros, una caché de búsqueda anticipada, y una tabla de índices compactos proporciona un uso eficiente de la memoria al tiempo que se reducen los accesos de E/S en el dispositivo de almacenamiento secundario en un sistema de deduplicación. Otros aspectos, tales como una caché de sesión, unas tablas de índices compactos de solo lectura para los subespacios, unas tablas de índices compactos de codificación de solo lectura, proporcionan beneficios adicionales en un sistema de deduplicación.

ENTORNOS DE RED Y DISTRIBUIDOS A MODO DE EJEMPLO

Un experto en la materia puede apreciar que las diversas realizaciones y procedimientos descritos en el presente documento pueden implementarse en relación con cualquier ordenador u otro cliente o dispositivo servidor, que puede desplegarse como parte de una red de ordenadores o en un entorno de informatización distribuida, y puede conectarse a cualquier tipo de almacén o almacenes de datos. En este sentido, las diversas realizaciones descritas en el presente documento pueden implementarse en cualquier sistema o entorno de ordenadores que tenga cualquier número de unidades de memoria o de almacenamiento, y cualquier número de aplicaciones y procedimientos produciéndose a través de cualquier número de unidades de almacenamiento. Esto incluye, pero no se limita a, un entorno con ordenadores servidores y ordenadores clientes desplegados en un entorno de red o en un entorno informático distribuido, que tiene un almacenamiento remoto o local.

La informatización distribuida proporciona compartición de los recursos y servicios informáticos mediante el intercambio comunicativo entre los dispositivos y sistemas informáticos. Estos recursos y servicios incluyen el intercambio de información, almacenamiento en caché y almacenamiento en disco para objetos, tales como archivos. Estos recursos y servicios también incluyen la compartición de la potencia de procesamiento a través de múltiples unidades de procesamiento para equilibrar la carga, la expansión de los recursos, la especialización del procesamiento, y similares. La informatización distribuida toma ventaja de la conectividad de red, permitiendo a los

clientes aprovechar su potencia colectiva para beneficio de toda la empresa. A este respecto, una variedad de dispositivos pueden tener aplicaciones, objetos o recursos que pueden participar en los mecanismos de gestión de recursos como se describe para las diversas realizaciones de la presente descripción.

La figura 6 proporciona un diagrama esquemático de un entorno informático en red o distribuido a modo de ejemplo. El entorno informático distribuido comprende unos objetos 610, 612, etc. informáticos y unos objetos o dispositivos 620, 622, 624, 626, 628, etc. de cálculo que pueden incluir programas, procedimientos, almacenes de datos, lógica programable, etc., como se representa por las aplicaciones 630, 632, 634, 636, 638 de ejemplo. Puede apreciarse que los objetos 610, 612, etc. informáticos y los objetos o dispositivos 620, 622, 624, 626, 628, etc. de cálculo, puede comprender diferentes dispositivos, tales como asistentes digitales personales (PDA), dispositivos de vídeo / audio, teléfonos móviles, reproductores de MP3, ordenadores personales, ordenadores portátiles, etc.

Cada objeto 610, 612, etc. informático y los objetos o dispositivos 620, 622, 624, 626, 628, etc. de cálculo, pueden comunicarse con uno o más objetos 610, 612, etc. informáticos y con los objetos o dispositivos 620, 622, 624, 626, 628, etc. de cálculo por medio de la red 640 de comunicaciones, o directa o indirectamente. A pesar de que se ilustra como un único elemento en la figura 6, la red 640 de comunicaciones puede comprender otros objetos informáticos y dispositivos informáticos que proporcionan servicios al sistema de la figura 6, y/o pueden representar múltiples redes interconectadas entre sí, que no se muestran. Cada objeto 610, 612, etc. informático o un objeto o dispositivo 620, 622, 624, 626, 628, etc. de cálculo, también puede contener una aplicación, tal como las aplicaciones 630, 632, 634, 636, 638, que podría hacer uso de una API, u otro objeto, software, firmware y/o hardware, adecuado para la comunicación con o la implementación de la aplicación proporcionada de acuerdo con las diversas realizaciones de la presente descripción.

Hay una gran variedad de sistemas, componentes y configuraciones de red que soportan los entornos informáticos distribuidos. Por ejemplo, los sistemas informáticos pueden conectarse entre sí por sistemas de cable o inalámbricos, por redes locales o redes ampliamente distribuidas. En la actualidad, muchas redes están acopladas a Internet, lo que proporciona una infraestructura para la informática ampliamente distribuida y abarca muchas redes diferentes, aunque cualquier infraestructura de red puede usarse para las comunicaciones a modo de ejemplo se realizan incidentes en los sistemas como se describe en diversas realizaciones.

Por lo tanto, pueden utilizarse una serie de topologías de red e infraestructuras de red, tales como cliente/servidor, par a par, o arquitecturas híbridas. El “cliente” es un miembro de una clase o grupo que usa los servicios de otra clase o grupo con el que no está relacionado. Un cliente puede ser un procedimiento, por ejemplo, aproximadamente un conjunto de instrucciones o tareas, que solicita un servicio proporcionado por otro programa o procedimiento. El procedimiento cliente utiliza el servicio solicitado sin tener que “conocer” ningún detalle de trabajo sobre el otro programa o del propio servicio.

En una arquitectura cliente/servidor, específicamente un sistema en red, un cliente es normalmente un ordenador que accede a unos recursos de red compartidos proporcionados por otro ordenador, por ejemplo, un servidor. En la ilustración de la figura 6, como un ejemplo no limitativo, unos objetos o dispositivos 620, 622, 624, 626, 628, etc. informáticos pueden considerarse como clientes y los objetos 610, 612, etc. informáticos pueden considerarse como servidores, donde los objetos 610, 612, etc. informáticos que actúan como servidores proporcionan servicios de datos, tales como la recepción de datos desde los objetos o dispositivos 620, 622, 624, 626, 628, etc. informáticos cliente, el almacenamiento de datos, el procesamiento de datos, la transmisión de datos a los objetos o dispositivos 620, 622, 624, 626, 628, etc. informáticos cliente, aunque cualquier ordenador puede considerarse como un cliente, un servidor, o ambos, en función de las circunstancias.

Un servidor es normalmente un sistema informático remoto accesible a través de una red remota o local, tal como Internet o las infraestructuras de red inalámbricas. El procedimiento cliente puede estar activo en un primer sistema informático, y el procedimiento servidor puede estar activo en un segundo sistema informático, comunicándose entre sí a través de un medio de comunicaciones, proporcionando de este modo una funcionalidad distribuida y permitiendo que múltiples clientes tomen ventaja de las capacidades de recopilación de información del servidor.

En un entorno de red en el que la red 640 de comunicaciones o bus es Internet, por ejemplo, los objetos 610, 612, etc. informáticos pueden ser servidores de Web con los que otros los objetos o dispositivos 620, 622, 624, 626, 628, etc. informáticos se comunican a través de cualquiera de una serie de protocolos conocidos, tales como el protocolo de transferencia de hipertexto (HTTP). Los objetos 610, 612, etc. informáticos que actúan como servidores también pueden servir como clientes, por ejemplo, los objetos o dispositivos 620, 622, 624, 626, 628, etc. informáticos, como puede ser característico de un entorno informático distribuido.

DISPOSITIVO INFORMÁTICO A MODO DE EJEMPLO

Como se ha mencionado, de manera ventajosa, las técnicas descritas en el presente documento pueden aplicarse a cualquier dispositivo. Por lo tanto, puede entenderse que se contemplan dispositivos de mano, potables y otros dispositivos informáticos y objetos informáticos de todo tipo para su uso en relación con las diversas realizaciones. En consecuencia, el siguiente ordenador remoto de fin general que se describe a continuación en la figura 7 es un ejemplo de un dispositivo informático.

Las realizaciones pueden implementarse en parte a través de un sistema operativo, para su uso por un desarrollador de servicios para un dispositivo u objeto, y/o incluirse dentro del software de aplicación que opera para realizar uno o más aspectos funcionales de las diversas realizaciones descritas en el presente documento. El software puede describirse en el contexto general de instrucciones ejecutables por ordenador, tales como los módulos de programa, ejecutándose por uno o más ordenadores, tales como estaciones de trabajo clientes, servidores u otros dispositivos. Los expertos en la materia apreciarán que los sistemas de ordenador tienen una variedad de configuraciones y protocolos que pueden usarse para comunicar datos, y por lo tanto, no se considera limitante ninguna configuración o protocolo específico.

Por lo tanto, la figura 7 ilustra un ejemplo de un entorno 700 de sistema informático adecuado en el que pueden implementarse uno o más aspectos de las realizaciones descritas en el presente documento, aunque, como se ha dejado claro anteriormente, el entorno 700 de sistema informático es solo un ejemplo de un entorno informático adecuado y no pretende sugerir ninguna limitación en cuanto al ámbito de uso o funcionalidad. Además, el entorno 700 de sistema informático no está destinado a interpretarse como que tiene alguna dependencia en relación con uno cualquiera o una combinación de componentes ilustrados en el entorno 700 de sistema informático a modo de ejemplo.

Haciendo referencia a la figura 7, un dispositivo remoto a modo de ejemplo para implementar una o más realizaciones incluye un dispositivo informático de fin general en la forma de un ordenador 710. Los componentes del ordenador 710 pueden incluir, pero no se limitan a, una unidad 720 de procesamiento, una memoria 730 de sistema, y un bus 722 de sistema que acopla diversos componentes del sistema incluyendo la memoria del sistema a la unidad 720 de procesamiento.

El ordenador 710 incluye normalmente una variedad de medios legibles por ordenador y puede ser cualquier medio disponible al que puede accederse por el ordenador 710. La memoria 730 de sistema puede incluir un medio de almacenamiento informático en la forma de memoria volátil y/o no volátil, tal como memoria de solo lectura (ROM) y/o memoria de acceso aleatorio (RAM). A modo de ejemplo, y no de limitación, la memoria 730 de sistema también puede incluir un sistema operativo, programas de aplicación, otros módulos de programa, y datos de programa.

Un usuario puede introducir órdenes e información en el ordenador 710 a través de unos dispositivos 740 de entrada. Una pantalla u otro tipo de dispositivo de visualización también está conectado al bus 722 de sistema a través de una interfaz, tal como la interfaz 750 de salida. Además de un monitor, los ordenadores también pueden incluir otros dispositivos periféricos de salida, tales como unos altavoces y una impresora, que puede conectarse a través de la interfaz 750 de salida.

El ordenador 710 puede operar en un entorno de red o distribuido usando conexiones lógicas a uno o más ordenadores remotos, tales como el ordenador 770 remoto. El ordenador 770 remoto puede ser un ordenador personal, un servidor, un encaminador, un PC de red, un dispositivo par u otro nodo de red común, o cualquier otro dispositivo de consumo o transmisión de medios remotos, y puede incluir cualquiera o todos los elementos descritos anteriormente con respecto al ordenador 710. Las conexiones lógicas representadas en la figura 7 incluyen una red 772, como una red de área local (LAN) o una red de área amplia (WAN), pero también puede incluir otras redes/buses. Tales entornos de red se localizan comúnmente en hogares, oficinas, redes informáticas para toda la empresa, intranets e Internet.

Como ha mencionado anteriormente, mientras que se han descrito las realizaciones a modo de ejemplo en relación con diferentes dispositivos informáticos y arquitecturas de red, los conceptos subyacentes pueden aplicarse a cualquier sistema de red y a cualquier dispositivo o sistema informático en el que puede desearse mejorar la eficiencia de uso de los recursos.

Además, hay varias formas de implementar la misma o similar funcionalidad, por ejemplo, una API apropiada, una caja de herramientas, un código de controlador, un sistema operativo, un control, un objeto de software independiente o descargable, etc., lo que permite a las aplicaciones y servicios tener ventaja de las técnicas proporcionadas en el presente documento. Por lo tanto, las realizaciones en el presente documento se contemplan a partir del punto de vista de una API (o de otro objeto de software), así como a partir de un objeto de software o hardware que implementa una o más realizaciones como se describen en el presente documento. Por lo tanto, las diversas realizaciones descritas en el presente documento pueden tener aspectos que sean totalmente en hardware, parcialmente en hardware y parcialmente en software, así como en software.

La expresión “a modo de ejemplo” se usa en el presente documento para significar que sirve como un ejemplo, caso o ilustración. Para evitar dudas, el objeto desvelado en el presente documento no está limitado por tales ejemplos. Además, cualquier aspecto o diseño descrito en el presente documento como “a modo de ejemplo” no debe interpretarse necesariamente como preferido o ventajoso sobre otros aspectos o diseños, ni tiene la intención de impedir estructuras a modo de ejemplo equivalentes y técnicas conocidas por los expertos en la materia. Además, en la medida en que se usan los términos “incluye”, “tiene”, “contiene”, y otras palabras similares, para evitar dudas, dichos términos están destinados a incluirse de una manera similar a la expresión “que comprende” como una palabra de transición abierta sin excluir ningún elemento adicional u otros elementos cuando se emplea en una reivindicación.

Como se ha mencionado, las diversas técnicas descritas en el presente documento pueden implementarse en relación con hardware o software o, donde sea apropiado, con una combinación de ambos. Tal como se usa en el presente documento, los términos “componente”, “módulo”, “sistema” y similares se pretende asimismo que se refieran a una entidad relacionada con ordenadores, ya sea un hardware, una combinación de hardware y software, software, o software en ejecución. Por ejemplo, un componente puede ser, pero no se limita a ser, un procedimiento que se ejecuta en un procesador, un procesador, un objeto, un ejecutable, un hilo de ejecución, un programa, y/o un ordenador. A modo de ilustración, tanto una aplicación que se ejecuta en el ordenador como el ordenador pueden ser un componente. Uno o más componentes pueden residir dentro de un procedimiento y/o un hilo de ejecución y un componente pueden estar localizados en un ordenador y/o distribuidos entre dos o más ordenadores.

Los sistemas mencionados anteriormente se han descrito con respecto a la interacción entre varios componentes. Puede apreciarse que tales sistemas y componentes pueden incluir estos componentes o subcomponentes especificados, algunos de los componentes o sub-componentes especificados y/o componentes adicionales, y de acuerdo con diversas permutaciones y combinaciones de los anteriores. Los sub-componentes también pueden implementarse como componentes acoplados comunicativamente a otros componentes en lugar de incluidos dentro de los componentes padres (jerárquicos). Además, puede observarse que uno o más componentes pueden combinarse en un único componente que proporciona una funcionalidad agregada o divididos en varios sub-componentes separados, y que una cualquiera o más capas intermedias, tal como una capa de gestión, puede proporcionarse para acoplarse comunicativamente a tales sub-componentes con el fin de proporcionar una funcionalidad integrada. Cualquiera de los componentes descritos en el presente documento también puede interactuar con uno o más de los otros componentes no descritos específicamente en el presente documento, pero en general conocidos por los expertos en la materia.

En vista de los sistemas a modo de ejemplo descritos en el presente documento, las metodologías que pueden implementarse de acuerdo con el objeto descrito también pueden apreciarse haciendo referencia a los diagramas de flujo de las diversas figuras. Mientras que por fines de simplicidad de la explicación, las metodologías se muestran y describen como una serie de bloques, debería entenderse y apreciarse que las diversas realizaciones no están limitadas por el orden de los bloques, en la medida en que algunos bloques pueden producirse en diferentes órdenes y/o simultáneamente con otros bloques de lo que se muestra y se describe en el presente documento. Donde no es secuencial, o ramificado, el flujo se ilustra a través de un diagrama de flujo, puede apreciarse que varias otras ramas, trayectorias de flujo, y órdenes de los bloques pueden implementarse para conseguir el mismo resultado o similar. Por otra parte, algunos bloques ilustrados son opcionales en la implementación de las metodologías descritas a continuación en el presente documento.

CONCLUSIÓN

Aunque la invención es susceptible de diversas modificaciones y construcciones alternativas, ciertas realizaciones ilustradas de la misma se muestran en los dibujos y se han descrito anteriormente en detalle. Sin embargo, debería entenderse que no hay intención de limitar la invención a las formas específicas desveladas, sino que por el contrario, la intención es cubrir todas las modificaciones, construcciones alternativas y equivalentes que caen dentro del ámbito de la invención.

Además de las diversas realizaciones descritas en el presente documento, debería entenderse que pueden usarse otras realizaciones similares o modificaciones y pueden hacerse adiciones a la realización(s) descrita para realizar la misma o equivalente función de la realización(s) correspondiente sin desviarse de la misma. Aún más, múltiples chips de procesamiento o múltiples dispositivos pueden compartir el rendimiento de una o más funciones descritas en el presente documento, y de manera similar, el almacenamiento puede efectuarse a través de una pluralidad de dispositivos. Por consiguiente, la invención no debe limitarse a cualquier realización única, sino más bien debe interpretarse en su ámbito de acuerdo con las reivindicaciones adjuntas.

REIVINDICACIONES

1. Un sistema que comprende un servicio (116) de índice hash que comprende un dispositivo (124) de almacenamiento primario y un dispositivo (128) de almacenamiento secundario, en el que:

el dispositivo de almacenamiento secundario comprende un índice (126) hash estructurado en registros mantenido en el dispositivo de almacenamiento secundario, en el que las entradas del índice hash estructurado en registros comprenden valores hash de fragmentos de datos y metadatos asociados con cada valor hash;
el dispositivo de almacenamiento primario comprende una tabla (122) de índices compactos que incluye las firmas compactas representativas de los valores hash en el índice hash, y para cada firma compacta, un puntero a una localización del valor hash correspondiente en el índice hash; y
el servicio de índice hash se configura para acceder a la tabla de índices compactos para buscar una firma compacta que corresponde a un valor hash solicitado para buscar, y para devolver un resultado de no encontrado, si la firma compacta no se encuentra en la tabla de índices compactos, y para seguir el puntero para buscar una entrada en el índice hash estructurado en registros que posiblemente contiene el valor hash solicitado si la firma compacta se encuentra en la tabla de índices compactos.

2. El sistema de la reivindicación 1, en el que el servicio de índice hash está configurado además para acceder a una caché de sesión para buscar un valor hash y devolver los metadatos asociados con ese valor hash si se encuentra en la caché de sesión, con el fin de que no sea necesario acceder al índice hash estructurado en registros de los metadatos si se encuentra en la caché de sesión.

3. El sistema de la reivindicación 1 que comprende además, una caché de búsqueda anticipada en el dispositivo de almacenamiento primario que incluye valores hash y entradas de metadatos almacenados en la caché del índice hash estructurado en registros, en el que el servicio de índice hash está configurado además para acceder a la caché de búsqueda anticipada para buscar un valor hash y devolver los metadatos asociados con ese valor hash si se encuentra en la caché de búsqueda anticipada, con el fin de que no se necesite acceder al índice hash estructurado en registros de los metadatos si se encuentra en la caché de búsqueda anticipada.

4. Un procedimiento realizado al menos en parte en al menos un procesador, que comprende:

mantener un índice (126) hash en un dispositivo (128) de almacenamiento secundario, en el que las entradas del índice hash incluyen valores hash, calculándose cada valor hash a partir de un fragmento de datos deduplicado, y asociándose con metadatos mediante los que puede localizarse el fragmento de datos deduplicado;
mantener una tabla (122) de índices compactos en un dispositivo (124) de almacenamiento primario que incluye firmas compactas representativas de los valores hash en el índice hash, y para cada firma compacta, un puntero a una localización del valor hash correspondiente en el índice hash; y
acceder (510) a la tabla de índices compactos para buscar una firma compacta que corresponde a un valor hash solicitado proporcionado en una solicitud, y devolver (522) un resultado de no encontrado en respuesta a la solicitud si ninguna de las firmas compactas se encuentra en la tabla de índices compactos, o seguir (514) uno o más punteros para determinar (516) si una entrada en el índice hash contiene el valor hash solicitado si la firma compacta se encuentra en la tabla de índices compactos.

5. El procedimiento de la reivindicación 4 que comprende, además, mapear cada valor hash hasta un máximo de dos o más entradas en la tabla de índices compactos, en la que cada entrada contiene una firma única.

6. El procedimiento de la reivindicación 5 que comprende, además, insertar una firma compacta en la tabla de índices compactos usando un algoritmo basado en una hash de cuco para evitar colisiones hash.

7. El procedimiento de la reivindicación 6 que comprende, además, escribir al menos una firma compacta en una tabla de desbordamiento en lugar de insertar la firma compacta en la tabla de índices compactos cuando están ocupadas todas las entradas de la tabla de índices compactos, o cuando se han intentado un número suficiente de relocalizaciones de cuco, o ambos.

8. Uno o más medios legibles por ordenador que tienen instrucciones ejecutables por ordenador, que cuando se ejecutan en un ordenador se adaptan para realizar todas las etapas de las reivindicaciones 4 a 7.

9. El uno o más medios legibles por ordenador de la reivindicación 8 que tienen instrucciones ejecutables por ordenador adicionales, que cuando se ejecutan en el ordenador se adaptan para realizar las etapas, que comprenden:

mantener una caché de búsqueda anticipada en el dispositivo de almacenamiento primario que incluye valores hash y entradas de metadatos almacenados en la caché del índice hash; y
acceder a la caché de búsqueda anticipada para buscar el valor hash solicitado, y devolver metadatos en respuesta a la solicitud si el valor hash solicitado se encuentra en la caché de búsqueda anticipada.

10. El uno o más medios legibles por ordenador de la reivindicación 9, que tienen además unas instrucciones ejecutables por ordenador, que cuando se ejecutan en el ordenador se adaptan para realizar las etapas, que

comprenden la realización de la etapa de acceder a la tabla de índices compactos si el valor hash solicitado no se encuentra en la caché de búsqueda anticipada.

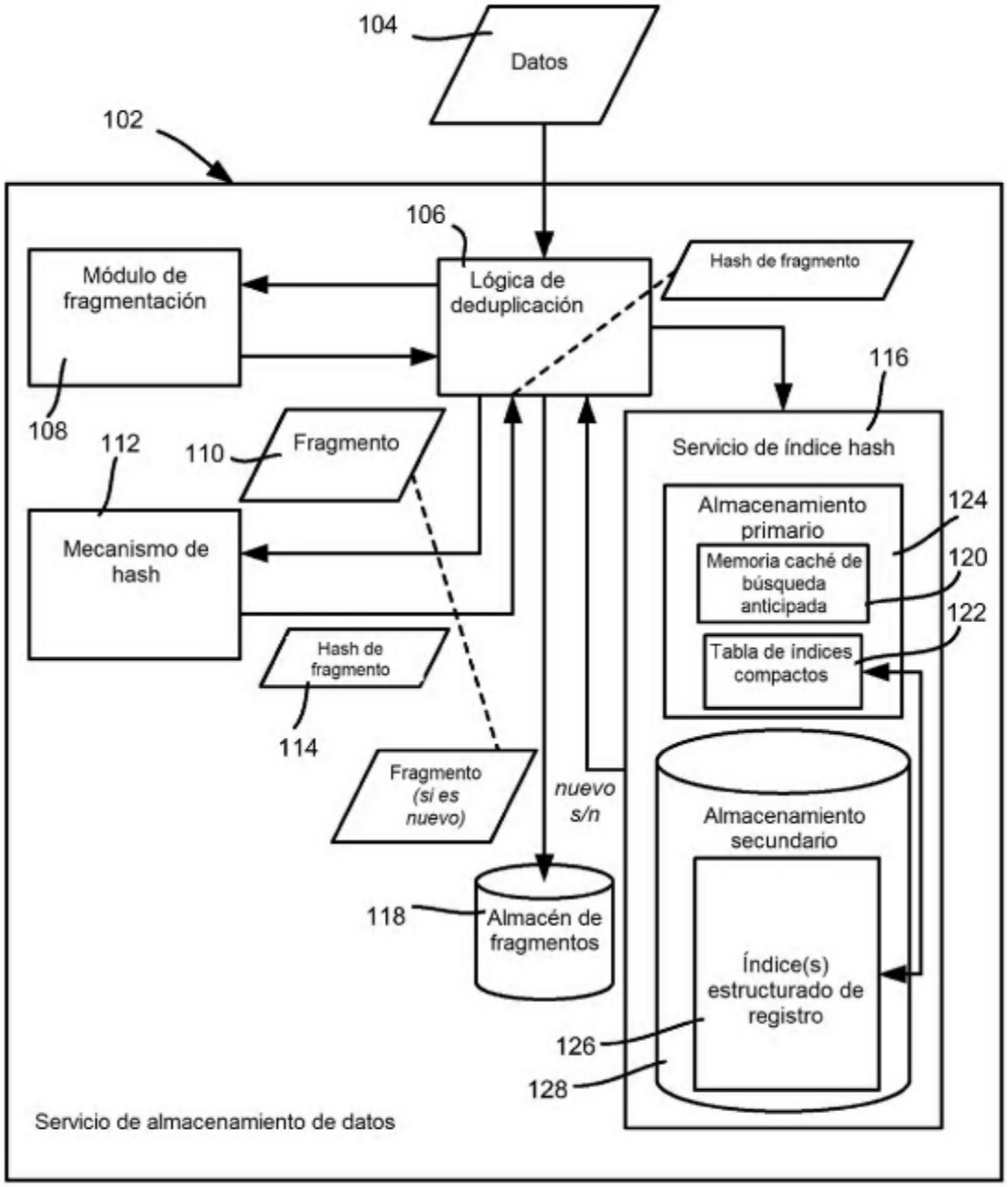
104
Datos
102
106
Módulo de fragmentación
Lógica de deduplicación
Hash de fragmento
116
108
112
110
Fragmento
Servicio de índice hash
Almacenamiento primario
124
Mecanismo de hash
Memoria caché de búsqueda anticipada
120
122
Tabla de índices compactos
Hash de fragmento
114
Fragmento (si es nuevo)
nuevo s/n
Almacenamiento secundario
118
Almacén de fragmentos
Índice(s) estructurado de registro
126
128
Servicio de almacenamiento de datos

FIG. 1

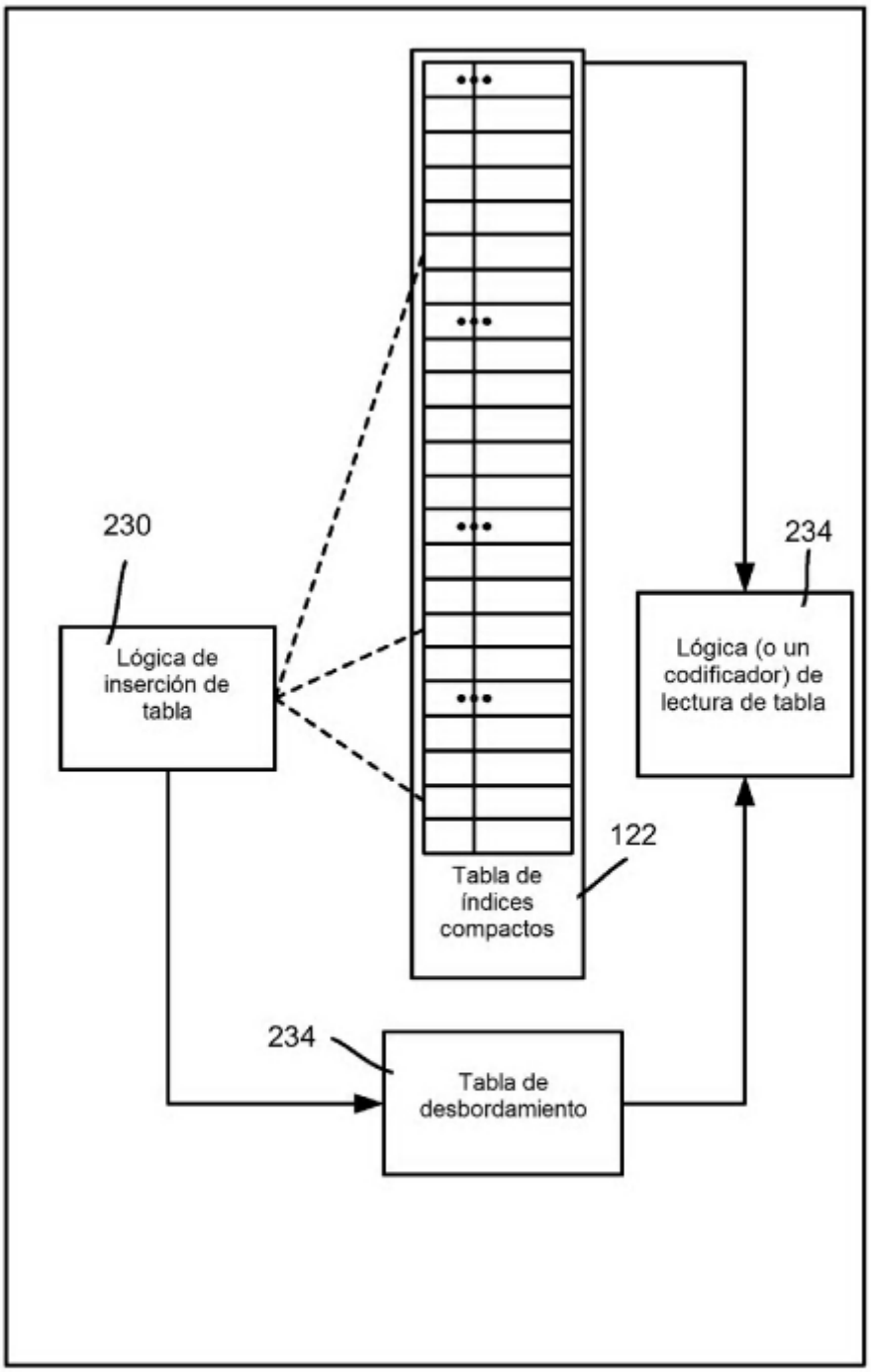
230
Lógica de inserción de tabla
234
Lógica (o un codificador) de lectura de tabla
122
Tabla de índices compactos
234
Tabla de desbordamiento

FIG. 2

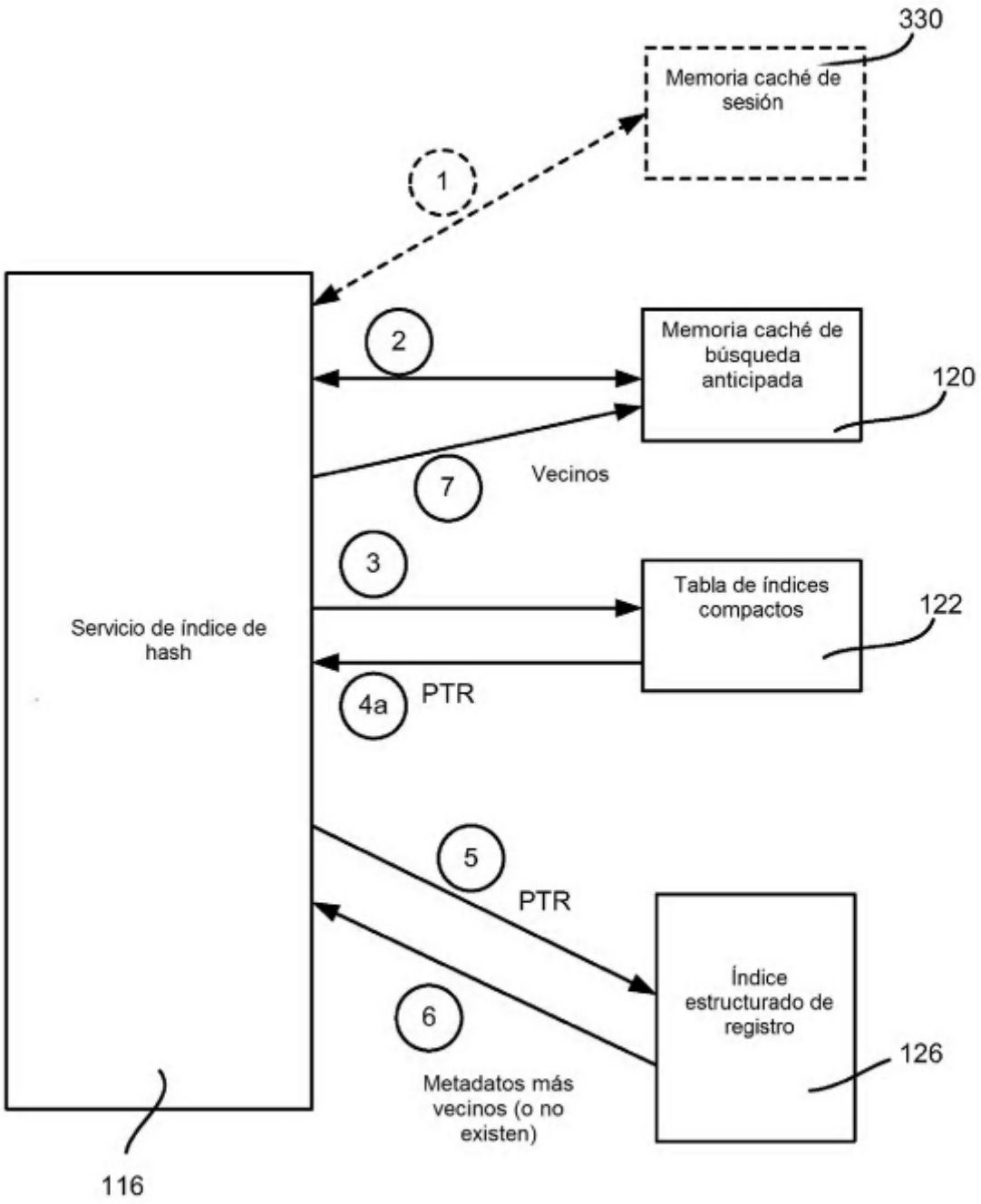
330
Memoria caché de sesión
1
2
Memoria caché de búsqueda anticipada
120
7
Vecinos
3
Tabla de índices compactos
122
Servicio de índice de hash
4a
PTR
5
PTR
Índice estructurado de registro
126
6
Metadatos más vecinos (o no existen)
116

FIG. 3

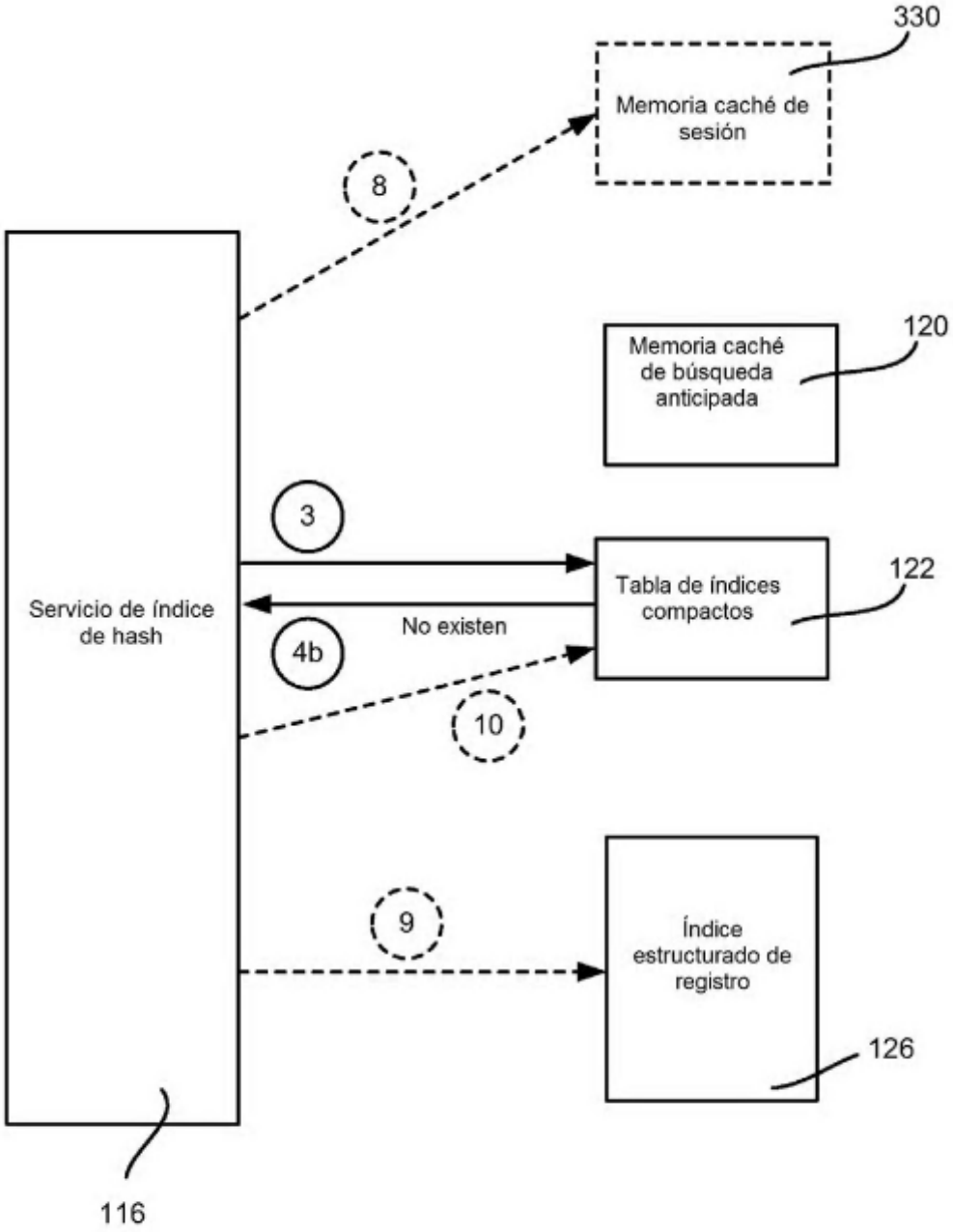
330
Memoria caché de sesión
8
120
Memoria caché de búsqueda anticipada
3
Servicio de índice de hash
Tabla de índices compactos
122
No existen
4b
10
9
Índice estructurado de registro
126
116

FIG. 4

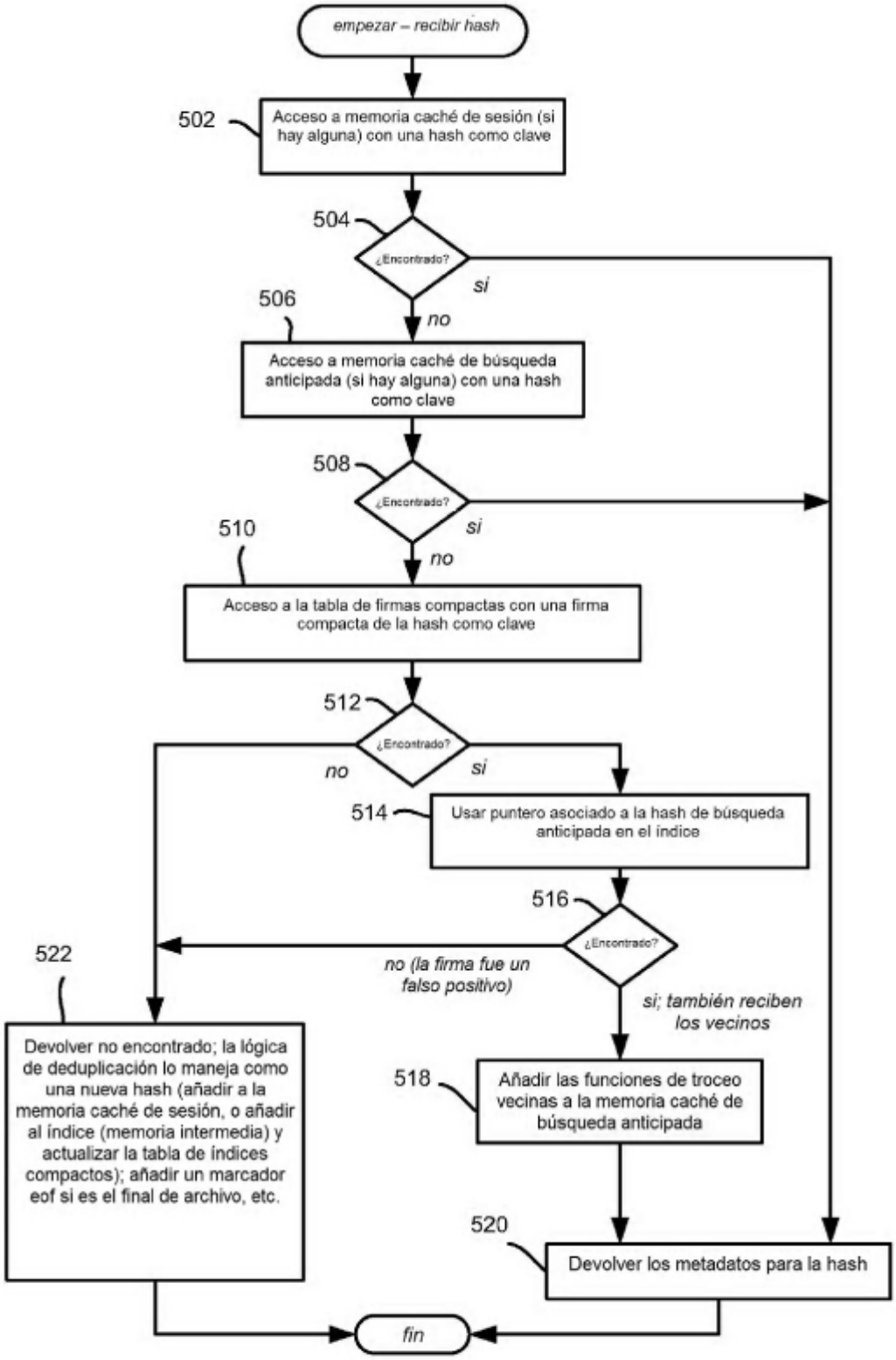
empezar – recibir hash
502
Acceso a memoria caché de sesión (si hay alguna) con una hash como clave
504
¿Encontrado?
si
no
506
Acceso a memoria caché de búsqueda anticipada (si hay alguna) con una hash como clave
508
¿Encontrado?
si
no
510
Acceso a la tabla de firmas compactas con una firma compacta de la hash como clave
512
¿Encontrado?
no
si
514
Usar puntero asociado a la hash de búsqueda anticipada en el índice
516
¿Encontrado?
no (la firma fue un falso positivo)
si; también reciben los vecinos
522
Devolver no encontrado; la lógica de deduplicación lo maneja como una nueva hash (añadir a la memoria caché de sesión, o añadir al índice (memoria intermedia) y actualizar la tabla de índices compactos); añadir un marcador eof si es el final de archivo, etc.
518
Añadir las funciones de troceo vecinas a la memoria caché de búsqueda anticipada
520
Devolver los metadatos para la hash
fin

FIG. 5

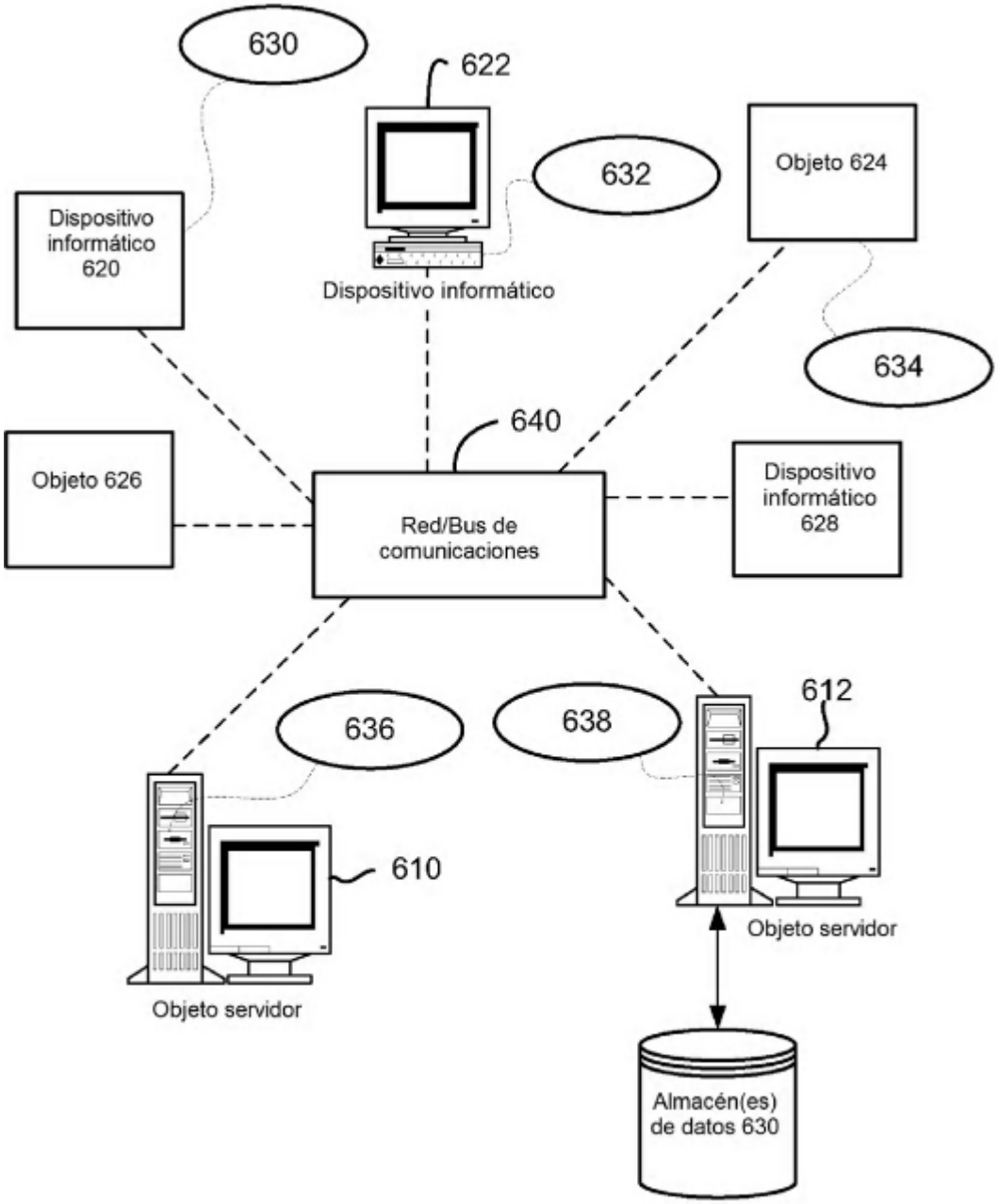
630
622
632
Objeto 624
Dispositivo informático 620
Dispositivo informático
634
640
Objeto 626
Red/Bus de comunicaciones
Dispositivo informático 628
636
638
612
610
Objeto servidor
Objeto servidor
Almacén(es) de datos 630

FIG. 6

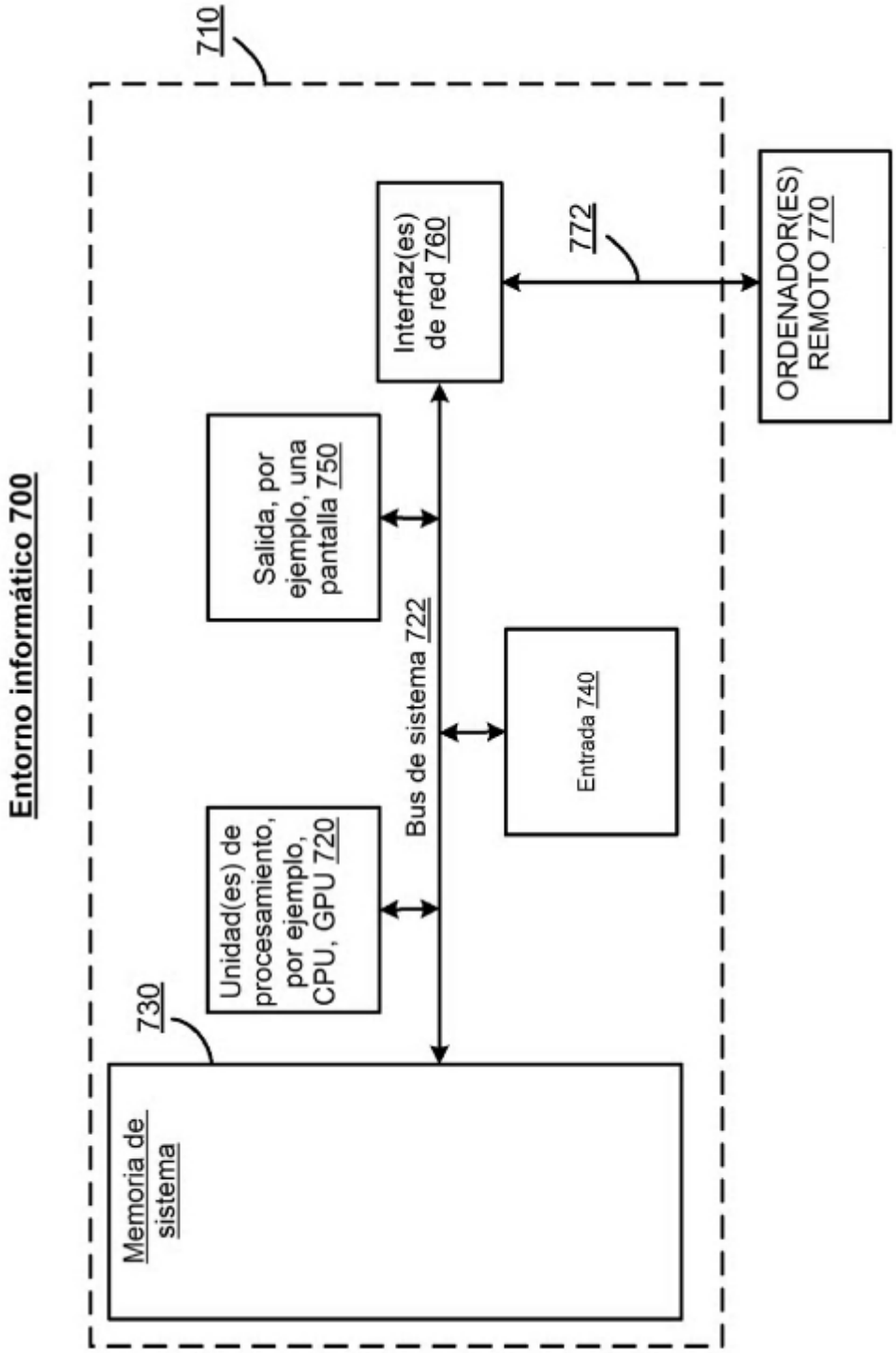
Entorno informático 700
710
Memoria de sistema
730
Unidad(es) de procesamiento, por ejemplo, CPU, GPU 720
Bus de sistema 722
Salida, por ejemplo, una pantalla 750
Entrada 740
Interfaz(es) de red 760
772
ORDENADOR(ES) REMOTO 770

FIG. 7