DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

19 de Junio de 2013 — Bases de Datos

Integrante	LU	Correo electrónico
Agustina Ciraco	630/06	agusciraco@gmail.com
Nadia Heredia	589/08	heredianadia@gmail.com
Pablo Antonio	290/08	pabloa@gmail.com
Vanesa Stricker	159/09	vanesastricker@gmail.com

Índice

1. Introducción

En este trabajo práctico, analizaremos el impacto de las diferentes estrategias para manejar los buffer pools. Nos concentraremos en el módulo Buffer Manager.

Las diferentes estrategias que consideraremos son

- un solo buffer
- múltiples buffers (distribución de Oracle)

Veremos las ventajas y desventajas de cada uno según la situación.

1.1. Formato

Por último, daremos nuestras conclusiones.

2. Algoritmo de Touch Count

En esta sección se presentará el algoritmo de touch-count, dando primero un breve resumen, luego un contexto y por último exponiendo una explicación.

2.1. Resumen

En esta subsección se dará un breve resumen del funcionamiento general del algoritmo de touch-count.

El algoritmo de touchcount fue introducido en Oracle 8i, para resolver las alteraciones del cache producidas por scans muy largos.

Como en muchas estrategias de reemplazo de páginas, se mantiene una lista LRU, con las páginas ordenadas según su frecuencia de uso. En un extremo están las más frecuentemente usadas (extremo MRU), mientras que en el otro están las menos frecuentemente usadas (extremo LRU).

Se hace que los bloquen ganen su derecho a estar y permanecer en el extremo MRU de la lista LRU. Es muy deseable estar en ese extremo para no ser desalojado.

Se introduce el concepto de *touch-count*, donde cada buffer tiene un contador. Cuando un buffer es tocado, este contador se incrementa. Oracle trata de que este contador se incremente solo una vez cada 3 segundos (para que no se incremente de más).

El mecanismo mediante el cual los buffers se van moviendo en la lista LRU sigue las siguientes reglas

- Un nuevo buffer se ubica en la posición `_db_percent_hot_default` de la lista LRU. Esta constante establece dónde se encuentra la mitad, que deja hacia un lado los buffers que se consideran que son de alto uso (**hot region**), y del otro los de bajo uso (**cold region**).
- Cuando el valor de touch-count de un buffer es mayor a `_db_aging_hot_criteria` (generalmente 2), se mueve ese bloque al extremo MRU de la lista. Una vez que un buffer es desplazado al extremo MRU, su contador de touch-count es reseteado a cero.
- Si un buffer en el extremo MRU no es usado nuevamente, se mueve al extremo LRU. En este caso, un buffer en la mitad de la lista pasará de la **cold region** a la **hot region**. El valor de touch count de ese buffer se establece en 1.

2.2. Contexto

En esta subsección se aclararán los problemas existentes con los algoritmos anteriores (al de touch-count) de reemplazo de páginas.

Antes de la creación del algoritmo de touch count, Oracle tenía un algoritmo presuponía la existencia de buffers pequeños (lo cual tenía sentido por la época en la que se desarrolló). Con el avance del tiempo, esta suposición de que los buffers eran pequeños dejó de ser cierta, y el algoritmo se tornó obsoleto.

Un problema que existía era en el caso de realizar un full-table scan, ya que cada bloque de la tabla era colocado en cache, removiendo los bloques existentes, entre los cuales se encontraban los más utilizados. Esto destruía toda optimización posible sobre el buffer de bloques.

Por ejemplo, si la tabla de que cual se leía tenía 600 bloques, y en cache existían 500 bloques, se hacía una limpieza total de cache, ya que todos los bloques existentes previamente en cache, eran removidos para hacer lugar a los de la tabla. Esto ocurría en todos los casos en los que se realizara un full-table scan, aún cuando los bloques fueran leídos solamente una vez.

Para atacar este problema, se implementó un algoritmo de LRU modificado, que posicionaba los bloques de un full-table scan en el extremo LRU de la lista LRU (para que fueran descartados primero). Además sólo se permitía que una cantidad limitada de estos bloques ocupara el buffer, amortiguando la pérdida de bloques existentes.

Sin embargo, existía otro problema, esta vez con los index range scans de gran tamaño. Sólo soportaba full-table scans y no index scans.

Para mantener la eficiencia en las operaciones, es necesario contar con un algoritmo que sea rápido, y flexible. Deberá hacer que los buffers se ganen el derecho de mantenerse en cache.

2.3. Funcionamiento

Cada buffer tiene asignado un contador (touch-count). Cada vez que un bloque es tocado, el contador es incrementado. El valor de este contador representa la "popularidad" de cada bloque.

Existen dos áreas definidas, el área de alto uso (**hot**), y la de bajo uso (**cold**). Todos los buffers en la **hot region** son denominamos **hot buffers**, y los pertenecientes a la **cold region**, **cold buffers**.

Existe un marcador de midpoint entre las dos regiones. Este se va moviendo para asegurar una cantidad apropiada de buffers en la **hot region** y **cold region**. Por default, la división es a la mitad.

Cuando un proceso del servidor lee un bloque de disco, y lo coloca en el buffer cache, dicho buffer es ubicado en el medio de la cadena, es decir entre las hoy y cold regions.
Esta acción es conocida como midpoint insertion. Es importante notar que el bloque cargado no se considera como el primero de los más recientemente usados, sino que deberá ganarse su lugar incrementando su contador.

En teoría, cada vez que un bloque es tocado por el motivo que sea, su contador es incrementado. Sin embargo, en la práctica esto no ocurre, ya que habrá ocasiones en las cuales un bloque tenga un pico de intereés y luego no sea necesitado por un tiempo.

Para atacar este problema, Oracle permite que este contador sea incrementado a lo sumo cada tres segundos (este valor es configurable según las necesidades del usuario). Es decir, que no se incrementará cada vez que sea tocado, si esto ocurre muy seguido (aunque ocurra más de una vez en 3 segundos).

Cuando un proceso del servidor busca buffers libres para colocar un bloque en cache, debe buscar un buffer cuyo contenido no difiera con el correspondiente en disco. Si difiere, el buffer es denominado "dirty".
Si en la búsqueda de bloques, se encuentra alguno cuyo contador es mayor a dos, se mueve dicho bloque al extremo LRU de la lista y se setea el contador en 0. El contador de dicho bloque deberá ser incrementado en el corto plazo para no ser sacado.

Cuando un bloque pasa de la **cold region** a la **hot region**, su contador será reseteado a uno. Como en el caso anterior, el contador de dicho bloque deberá ser incrementado en el corto plazo para no ser sacado.

3. Detalles de Implementación

En esta sección se explican algunos detalles de las implementaciones de los algoritmos.

4. Decisiones

En esta sección explicaremos algunas de las decisiones tomadas durante la realización del trabajo práctico.

5. Resultados

En esta sección presentaremos los resultados obtenidos al utilizar diferentes trazas.

Se presenta para cada traza el hit rate de cada buffer.

5.1. Trazas utilizadas

6. Resultados

6.1. Comparación

6.2. Conclusiones

En esta subsección presentaremos nuestras observaciones sobre las comparaciones explicadas en la subsección anterior.