UNIVERSITÄT DES SAARLANDES
Fachbereich 14 – Informatik
Lehrstuhl Prof. Weikum

Datenbanksysteme WS 1998/99

Übung 6 — Beispiellösung zu Aufgabe 3

Aufgabe 1: LFU-Verdrängungsstrategie

Betrachten Sie das folgende Pufferersetzungsverfahren, das unter dem Namen LFU (Least Frequently Used) bekannt ist.

> *Es wird jeweils diejenige Seite im Puffer ersetzt, die seit ihrer Einlagerung in den Puffer am wenigsten häufig referenziert wurde.*

Zur Implementierung dieses Verfahrens genügt ein einfacher Zähler für jede Seite im Puffer. Überlegen Sie sich die Vor- und Nachteile dieses Verfahrens.

Aufgabe 2: Implementierung einer LRU-Pufferverwaltung

Sie sind an der Implementierung eines Datenbanksystems beteiligt und sollen die Teilkomponente *ReplacementManager* der Pufferverwaltung implementieren. Der ReplacementManager soll drei Operationen (in C-Syntax)

- ```
  PAGE*     fix(unsigned pageId);
            /* Dient zum Fixieren einer Seite im Puffer. Damit wird die
               Seite im virtuellen Speicher direkt adressierbar.          */
  ```
- ```
  void       unfix(PAGE* pageToUnfix);
             /* Dient zum Freigeben einer Seite im Puffer.
                Damit wird die Seite verdraengbar                         */
  ```
- ```
  PAGE*     getReplacementVictim();
             /* Dient zum Ermitteln des naechsten zu verdraengenden Opfers.*/
  ```

zum Fixieren und Freigeben einer Seite im Puffer bzw. zum Ermitteln des nächsten Verdrängungskandidaten anbieten. Die Datenstruktur *PAGE* beschreibt den Aufbau einer Seite im Puffer und enthält bisher die Identifikation der Seite, eine Markierung, ob sie seit der Einbringung in den Puffer modifiziert wurde und den Datenbereich der Seite. Die Deklaration für PAGE in C-Syntax sieht folgendermaßen aus:
  ```

```
typedef struct
{
        unsigned        pageId;                 /* ID der Seite im Puffer    */
        bool            dirty;                  /* Markierung,
                                                   ob Seite modifiziert wurde*/
        char            data[PAGE_SIZE];        /* Datenbereich              */
} PAGE;
```

Ihr Chef gibt Ihnen vor, daß Sie eine LRU-Verdrängungsstrategie implementieren sollen und eine Seite durch Aufruf von *fix* als referenziert (*used*) gilt. Eine referenzierte Seite soll solange unverdrängbar sein, bis für sie ein *unfix* aufgerufen wird. Die Operation *getReplacementVictim* soll die nach LRU-Reihenfolge als nächste zu verdrängende, verdrängbare Seite zurückliefern. Die ausgesuchte Seite soll für den ReplacementManager dann als verdrängt gelten, d.h. nach Aufruf von getReplacementVictim ist die zurückgelieferte Seite logisch nicht mehr im Puffer.

Der Pufferaufbau ist bereits teilweise durch die folgende Struktur *BUFFER* festgelegt. Der Puffer steht Ihrem Modul als globale Variable *TheBuffer* zur Verfügung.

```
struct BUFFER
{
        unsigned        capacity;       /* Kapazitaet des Puffers (# Seiten)*/
        unsigned        freePages;      /* Anzahl freier Seiten im Puffer   */
        PAGE    page[MAX_PAGES];        /* Seiten im Puffer                 */

} TheBuffer;
```

Zusätzlich wird Ihnen (glücklicherweise) die Datenstruktur HashTable zur Verfügung gestellt, zusammen mit den Operationen

```
PAGE*     getEntry( HashTable h, unsigned pageId );
          /* Liefert Eintrag innerhalb der Tabelle h an der Stelle 'pageId' */

void      insertEntry( HashTable h, unsigned pageId, PAGE* value );
          /* Fuegt Eintrag (pageId,PAGE*) in Tabelle h ein                  */

void      removeEntry( HashTable h, unsigned pageId );
          /* Entfernt Eintrag mit Schluessel 'pageId' aus Tabelle h         */
```

zum Ermitteln, Einfügen und Entfernen von Einträgen [Seiten-Id, Zeiger auf Seite] in einer Hash-Tabelle h.

a) Die Operation *fix* muß dafür sorgen, daß die angeforderte Seite im Puffer fixiert wird, d.h. daß sie unverdrängbar wird. Erst wenn *unfix* aufgerufen wird, kann die Seite aus dem Puffer verdrängt werden. Erklären Sie, warum die Seite nach Aufruf von *fix* nicht aus dem Puffer verdrängt werden darf.

b) Überlegen Sie sich eine effiziente Datenstruktur für die Verwaltung des Puffers mit einer LRU-Verdrängungsstrategie. Sie dürfen dazu die vorhandenen Datenstrukturen *BUFFER* und *PAGE* beliebig erweitern.

c) Geben Sie in C- oder PASCAL-ähnlicher Syntax eine Implementierung für die Operationen *fix*, *unfix* und *getReplacementVictim* an. Die Implementierung soll dabei die in a) entwickelten Datenstrukturen verwenden.

d) Ist es denn sinnvoll, daß eine Seite in dem Moment als referenziert gilt, wenn *fix* aufgerufen wird? Welchen Nachteil hat diese Vorgehensweise?

Diese Seite ist leer.

Diese Seite ist leer.

Diese Seite ist leer.

Diese Seite ist leer.

Aufgabe 3: 5-Minuten-Regel

Betrachten Sie eine *OLTP-Anwendung* (Online Transaction Processing), bei der sehr viele Benutzer quasi-gleichzeitig vom Datenbank-Server zu bedienen sind, die einzelnen Transaktionen aber sehr simpel sind. Zur Vereinfachung nehmen Sie an, daß eine Transaktion nur einen einzigen Seitenzugriff macht - dies wird manchmal als *'Rifle-Shot'-Transaktion* bezeichnet. Die *Antwortzeit* der einzelnen Transaktion ist also unkritisch, da maximal nur ein *Platten-I/O* anfällt. Dagegen ist es entscheidend, daß der Datenbank-Server einen ausreichend hohen *Durchsatz* hat (Transaktionen pro Sekunde), um die Benutzerlast zu bewältigen. Da der Durchsatz einer Magnetplatte bei einer *mittleren Bedienzeit* S (z.B. $10ms$ für einen *Random-I/O*) maximal $\frac{1}{S}$ beträgt, muß der Server ggf. mehrere Platten verwenden und die Daten über diese verteilen. Nehmen Sie an, daß in diesem Fall die Zugriffslast perfekt gleichmäßig über die Platten verteilt wäre, daß also kein Lastungleichgewicht entsteht. Alternativ kann der Durchsatz durch Einsatz eines geeignet großen Datenbankpuffers im Hauptspeicher erhöht werden. Nehmen Sie an, daß ein Seitenzugriff, der vom Datenbankpuffer bedient werden kann, keinerlei Ausführungskosten verursacht. Als Extremfall könnte also ein Server, der die ganze Datenbank im Hauptspeicher hält, einen beliebig hohen Durchsatz erreichen (was natürlich stark simplifiziert ist, da auch die CPU zum Engpaß werden kann).

Nehmen Sie an, die OLTP-Last greife insgesamt auf eine Relation mit $M = 1000000$ Seiten zu jeweils 16 KB zu. Von der Platzkapazität würde dafür eine einzige Platte ausreichen; diese könnte aber bezüglich des Durchsatzes überfordert sein. Der zu bewältigende Durchsatz betrage $X = 1000$ Transaktionen pro Sekunde = 1000 Seitenzugriffe pro Sekunde. Die mittlere Bedienzeit einer Magnetplatte sei $S = 10$ ms. Eine Magnetplatte koste 1000 DM, der RAM für eine Seite, also 16 KB Hauptspeicher, koste 5 Pfennig. Die Zugriffe auf die Datenbankseiten seien nicht gleichverteilt, sondern folgen einer sogenannten *Zipf-Verteilung*, bei der die *relative Häufigkeit der Zugriffe auf Seite i* gegeben ist durch:

$$f(i) = \frac{\left(\frac{1}{i}\right)}{\sum_{i=1..M}\left(\frac{1}{i}\right)}$$

(Die Seitennummern sind also o.B.d.A. nach absteigender Zugriffshäufigkeit sortiert.)

Wie müssen Sie den Server konfigurieren, damit er die Last bewältigen kann und dabei möglichst wenig kostet. Gehen Sie in folgenden Schritten vor:

a) Ermitteln Sie die die Anzahl der Zugriffe pro Sekunde für die Seite i.

Lösung:

$f(i)$ gibt die relative Häufigkeit der Zugriffe auf die Seite i an.

Die Ankunftsrate X_i , d.h. die Anzahl der Seitenzugriffe auf i pro Sekunde, ist dann: $X_i = X \cdot f(i)$.

b) Nehmen Sie an, daß die Seiten $1..j$ ständig im Datenbankpuffer gehalten werden und die Seiten $(j+1)..M$ auf Platten liegen und nicht vom Puffer profitieren. Die Anzahl der Platten muß so gewählt werden, daß die Seitenzugriffe pro Sekunde für die Seiten $(j+1)..M$ bewältigt werden können. Wieviele Platten benötigen Sie - in Abhängigkeit von j, M und S ?

Lösung:
Durchsatz pro Platte: $\frac{1}{S}$

Ankunftsrate für die Seiten $(j+1)\ldots M$:
$X \cdot \sum_{k=j+1}^{M}\left(f\left(k\right)\right)$

$\Rightarrow$ Anzahl der benötigten Platten:

$$D = \left\lceil \frac{X \cdot \sum_{k=j+1}^{M}(f(k))}{\frac{1}{S}} \right\rceil = \left\lceil S \cdot X \cdot \sum_{k=j+1}^{M}\left(f\left(k\right)\right) \right\rceil$$

c) Ermitteln Sie die Gesamtkosten der unter b) ermittelten Konfiguration (Kosten der Platten und des für den Datenbankpuffer verwendeten Hauptspeichers).
Lösung:
Gesamtkosten C :

$$C = D \cdot 1000 + j \cdot 0.05 \quad \text{[in DM]} =: g(j) \text{ “Kostenfunktion”}$$

d) Wie ist der Schwellwert j zu wählen, damit die Gesamtkosten des Servers minimal werden? Ab welcher Zugriffshäufigkeit pro Sekunde für eine Seite ist es für den Server kostengünstiger, diese auf Platte anstatt im Hauptspeicher zu halten? Diese Überlegung ist in der Literatur als 5-Minuten-Regel bekannt, weil es - mit der damaligen Technologie (1987) - günstiger war, eine Seite im Hauptspeicher zu halten, wenn sie mindestens einmal in 5 Minuten referenziert wurde (also eine Zugriffshäufigkeit von mindestens $\frac{1}{300}$ pro Sekunde hat).
Lösung:
Bestimme j, so daß $C = g(j)$ minimal:
Wegen der absteigenden Sortierung der $f(i)$ genügt es, die Grenzkosten ($\triangle C$) zu betrachten, die entstehen, wenn Seite j im Speicher bzw. auf der Platte gehalten wird.
Bei neutralen Grenzkosten ist der “break even”-Punkt erreicht, ab dem es günstiger ist, Seite j auf Platte zu halten:

$$\triangle C = g(j-1) - g(j) \overset{!}{=} 0$$

$$\Rightarrow \triangle C = \underbrace{S \cdot X \cdot f(j) \cdot 1000}_{\substack{Kosten,\ wenn \\ Seite\ j\ auf\ Platte}} - \underbrace{0.05}_{\substack{Kosten,\ wenn \\ Seite\ j\ im\ Speicher}} \overset{!}{=} 0$$

$$\Rightarrow X \cdot f(j) = \frac{0.05}{1000 \cdot S} = \frac{0.05}{1000 \cdot 0.01} = 0.05 * 10^{-2} \quad [s^{-1}]$$

D.h. bei Zugriffen, die häufiger als alle 200 Sekunden erfolgen, sollte die Seite im Speicher gehalten werden.