Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
Arbeitsgruppe Datenbanksysteme

Wintersemester 2016/2017
11. 11. 2016

Prof. Dr. Bernhard Seeger
Dipl. Inf. Marc Seidemann
Michael Körber M.Sc.

Übungen zur Vorlesung
Implementierung von Datenbanksystemen

Abgabe: 20. 11. 2016,
bis **spätestens** 23:59 Uhr
über die ILIAS Plattform

Übung 2

Aufgabe 2.1: Lokalität (4 Punkte)

Gegeben sei der folgende Referenzstring einer Transaktion:

```
AABBECFCCCDAAFFGLGGIMIMIMEEFGEF
```

Bestimmen Sie:

a) die aktuellen Lokalität für `t=5`, `t=15`, `t=22` und die Fenstergröße `6`.

b) die durchschnittliche Lokalität für die Fenstergröße `6`.

c) die LRU-Stacktiefenverteilung. Erstellen Sie hierfür ein Diagramm, in dem die Anzahl der Zugriffe gegen die Position im Stack aufgetragen ist.

Aufgabe 2.2: Ersetzungsstrategien (8 Punkte)

Gegeben sei der folgende Referenzstring einer Transaktion:

```
ABAABAACDBACED
```

Nehmen Sie an, dass ein Puffer mit drei Pufferrahmen zu Verfügung steht. Skizzieren Sie die Veränderung des Puffers unter Verwendung der folgenden Seitenersetzungsstrategien:

a) FIFO

b) LFU

c) LRU

d) CLOCK

Geben Sie zudem für jede Strategie die entsprechende Fehlseitenrate an.

Aufgabe 2.3: LRU-2 (6 Punkte)

Gegeben sei der folgende Referenzstring einer Transaktion:

```
BAABCDBEBAACDBE
```

Es sei angenommen, dass die Seitenanforderungen im Sekundentakt anfallen. Weiterhin sei ein Datenbankpuffer mit einer Größe von 4 Frames angenommen, bei dem die LRU-2-Strategie angewendet wird. Die Korrelationszeit beträgt 3 Sekunden. Skizzieren Sie den Status des Puffers sowie der Hilfsstrukturen `HIST(p,i)` und `LAST(p)` bei jeder Seitenanforderung der obigen Transaktion.

Aufgabe 2.4: 2Q Ersetzungsstrategie **(16 Punkte)**

In der Vorlesung haben Sie die LRU-k-Strategie kennen gelernt. In der Praxis liefert der Fall $k = 2$ bereits sehr gute Ergebnisse. In dieser Aufgabe sollen Sie eine Ersetzungsstrategie implementieren, die im Vergleich zu LRU-2 leichter zu implementieren ist und ähnlich gute Ergebnisse liefert.

Die 2Q-Strategie basiert auf zwei Listen. Die erste Liste (A1) wird mit Hilfe der FIFO-Strategie, die zweite (Am) mit Hilfe der LRU(-1)-Strategie verwaltet. Zunächst wird geprüft, ob eine angeforderte Seite bereits in einer der Listen verwaltet wird. Ist dies nicht der Fall, wird die Seite am Beginn von A1 einsortiert. Sollte die Liste bereits gefüllt sein, so wird die älteste Seite aus A1 entfernt (bei A1 handelt es sich um eine klassische Queue). Ist die Seite in A1 vorhanden, so wird sie an den Anfang von Am verschoben. Den Artikel zum 2Q-Verfahren finden Sie im ILIAS-System.

Für die Implementierung der Ersetzungsstrategie verwenden wir die Klasse `xxl.core.io.Buffer`. Ein Buffer bestimmt mit Hilfe der (abstrakten) Methode `Buffer.Slot::victim()`, welche Seite als nächstes ausgelagert werden soll. Seiten können im Puffer über den Aufruf von `fix()` angefordert werden. Sollte kein freier Rahmen mehr zur Verfügung stehen, so wird die von `victim()` zurückgelieferte Seite aus dem Puffer ausgelagert.

a) Erweitern Sie zunächst die Klasse Buffer so, dass die aktuelle Fehlseitenraten abgefragt werden kann. Die Fehlseitenrate gibt das Verhältnis von nicht im Puffer vorhandenen zu angeforderten Seiten an. Implementieren Sie hierzu eine abstrakte Klasse `FSRBuffer<O,I,E>`, die von `Buffer<O,I,E>` erbt. FSRBuffer pflegt zwei Zähler (Integer): Einen für die Gesamtzahl der Seitenzugriffe, und einen Zweiten für die Anzahl von Zugriffen auf nicht gepufferte Seiten. Des Weiteren soll eine Methode `public double getFSR()` implementiert werden, die die aktuelle Fehlseitenrate zurück gibt. Um die benötigten Informationen ermitteln zu können, muss die Methode `fix()` überschrieben werden.

b) Erstellen Sie eine Klasse `SimpleTwoQueueBuffer<O,I,E>`, die von `FSRBuffer<O,I,E>` erbt, mit folgenden Datenfeldern: `private Queue<Slot> a1, private Queue<Slot> am` und `private int kin`. Die Größe von `kin` soll 25% der Slots betragen. Die 2Q-Strategie soll mit Hilfe der Methoden `victim()` und `fix()` implementiert werden. `fix()` überschreibt dabei die Methode der Oberklasse:

```
@Override
protected Buffer<O,I,E>.Slot fix(O owner, I id,
        Function<? super I,? extends E> obtain) throws IllegalStateException {
    Slot result = (Slot) super.fix(owner, id, obtain);
    // TODO Queues aktualisieren
    return result;
}
```

`victim()` liefert gemäß der (simplen) 2Q-Strategie die auszulagernde Seite (Slot) auf Basis der Listen `am` und `a1`. Als Queue-Implementierung können Sie die Klasse `java.util.ArrayDeque` verwenden.

c) Das in Aufgabenteil b implementierte Verfahren berücksichtigt keine korrelierten Zugriffe. Dies kann jedoch durch eine Aufteilung von a1 erreicht werden. Erstellen Sie dazu eine Klasse `TwoQueueBuffer<O,I,E>`, die wieder von `FSRBuffer<O,I,E>` erbt. Anstelle von a1 besitzt die neue Klasse zwei Queues: `private Queue<Slot> a1in` und `private Queue<Slot> a1out`. Zusätzlich wird noch ein Datenfeld `private int kout` benötigt. a1in bildet nun die Korrelationszeit ab. Solange sich eine angeforderte Seite in a1in befindet, gilt der Zugriff als korreliert. a1in und a1out werden nach dem FIFO-Prinzip verwaltet. Sobald eine Seite aus a1in ausgelagert wird, wird diese in a1out eingefügt. Sollten eine Seite in a1out referenziert werden, so wird sie nach am verschoben. Implementieren Sie die oben beschriebene Ersetzungsstrategie unter Beachtung der Hinweise zu Aufgabenteil b.

d) Erweitern Sie die Klasse `xxl.core.io.LRUBuffer` zu `FSRLRUBuffer`, indem Sie diesen um die Messung der Fehlseitenrate erweitern (analog zu Aufgabenteil a).

e) Wir wollen nun die in den vorigen Aufgabenteilen erstellten Puffer testen. Simulieren Sie dazu das Szenario aus Aufgabe 2.2 und geben Sie die Fehlseitenraten für `LRUBuffer`, `SimpleTwoQueueBuffer` sowie `TwoQueueBuffer` auf der Konsole aus. Testen Sie zudem kin für jeweils 15%, 25% und 50%.