2. Übungsblatt

Ausgabe: 24. Oktober 2016 · Besprechung: Ab 31. Oktober 2016

1 Buffer Manager — Verdrängungsstrategien

In der Vorlesung wurde die Funktionsweise des *Buffer Managers* diskutiert. Dabei ist für die Wahl der zu überschreibenden *Frames* eine *Verdrängungsstrategie* notwendig. Die wohl bekannteste Verdrängungsstrategie ist LRU ("Least Recently Used"). In Datenbanksystemen werden jedoch oft auch andere Strategien verwendet, etwa *"clock"* und *"LRU-k"*. Letztere ist eine LRU-Variante, die die k letzten Zugriffe berücksichtigt[1].

- Beschreibe die Funktionsweise beider Verdrängungsstrategien.
- Welche Vor- und Nachteile haben sie jeweils gegenüber einer reinen LRU-Stategie?

2 Verdrängungsstrategien — Implementation

Schreibe ein Programm das eine Sequenz von `pin()`- und `unpin()` Aufrufen aus einer Datei liest und auf einem simulierten Buffer Manager mit n Frames ausführt. Das Programm soll für jeden Aufruf entscheiden, ob eine Seite nachgeladen werden muss, und wenn ja, in welchem Frame sie platziert wird. Der Buffer Manager soll mindestens zwei unterschiedliche Verdrängungsstrategien implementieren.

- Wir stellen die Datei `buffer_manager.trace` bereit die ein Protokoll der Buffer Manager-Aufrufe einer transaktionalen Workload[2] beinhaltet. Die Attribute `Pin/Unpin` und `reln` beschreiben die jeweilige Aktion und die Seitennummer.
- Variiere n und werte die Effizienz der Verdrängungsstrategien mit Hilfe der *Hitrate* aus. Die Hitrate ist das Verhältnis der `pin()` Aufrufe, die eine gepufferte Seite anfordern, zu den `pin()` Aufrufen, die ein Nachladen von der Festplatte erfordern.

[1] Details zu dieser Strategie finden sich in *"The LRU-K page replacement algorithm for database disk buffering."* von O'neil et al. (ACM SIGMOD 1993).

[2] Zur Generierung der Daten wurde der TPC-C Benchmark mit PostgresSQL ausgeführt.

Hinweise:

- Es reicht für einfache Verdrängungsstrategien nur die `pin()` Aufrufe zu berücksichtigen.
- Das folgende `C++`-Codefragment kann genutzt werden, um das Protokoll zu lesen:

```
#include <fstream>
int main () {
  std::ifstream file;
  file.open ("buffer_manager.trace");
  std::string action, dump; int page_no;
  while(file) {
    file >> action; file >> dump; file >> dump; file >> page_no;
    // action und page_no beinhalten hier die relevanten Werte
    getline(file, dump);
  }
  file.close();
  return 0;
}
```