Das Offline-Caching Problem

Wolfgang Mulzer

Gegeben seien ein Hauptspeicher mit n Datenwörtern und ein Cache, der k Wörter zwischenspeichern kann.

Wir wollen eine Zugriffsfolge $d_1, d_2, \ldots, d_n$ von n Anfragen auf den Hauptspeicher verarbeiten. Der Zugriff muss über den Cache erfolgen, d.h., ein Datum muss bei einem Zugriff im Cache vorhanden sein. Ist dies nicht der Fall, so müssen wir auf den Hauptspeicher zugreifen und das Datum nachladen.

Bei einer *Ersetzungsstrategie* handelt es sich um eine Strategie, die entscheidet, welche Daten im Cache vorgehalten werden sollen. Bei jedem Zugriff können wir ein Element aus dem Cache entfernen und durch ein Element aus dem Hauptspeicher ersetzen. Die Strategie sagt uns, wie dies von statten gehen soll. Ziel ist, die Anzahl der Hauptspeicherzugriffe zu minimieren.

Eine Ersetzungstrategie heißt *reduziert*, wenn sie nur dann ein Element aus dem Hauptspeicher lädt, wenn auf dieses zugegriffen wird und es nicht im Cache vorhanden ist. Die Anzahl der Hauptspeicherzugriffe entspricht dann der Anzahl der Cache-Misses.

Behauptung 1. *Jede Ersetzungsstrategie S lässt sich in eine reduzierte Strategie überführen, die höchstens so viele Speicherzugriffe durchführt wie S.*

Nun betrachten wir die folgende reduzierte Erzetzungstrategie SFF (furthest in the future Regel): Bei jedem Cache-Miss, entferne dasjenige Element aus dem Cache, dessen nächster Zugriff am weitesten in der Zukunft liegt.

Satz 2. *Die Strategie SFF minimiert die Anzahl der Hauptspeicherzugriffe.*

Der Beweis benutzt das folgende

Lemma 3 (Austauschlemma)**.** *Sei $d_1, d_2, \ldots, d_n$ eine Folge von Speicherzugriffen und S eine reduzierte Ersetzungstrategie, die mit SFF bei den ersten j Zugriffen bereinstimmt. Dann existiert eine reduzierte Strategie S', die mit SFF bei den ersten $j+1$ Zugriffen übereinstimmt, und höchstens so viele Hauptspeicherzugriffe durchführt wie S.*

Proof. Betrachte den Zugriff d_{j+1}. Da S und SFF bis jetzt dasselbe getan haben, muss der Cache für beide Strategien bei diesem Zugriff den gleichen Inhalt besitzen.

Wenn d_{j+1} im Cache vorhanden ist, so tun SFF und S nichts (weil S reduziert ist), und wir können $S' = S$ wählen.

Wenn d_{j+1} nicht im Cache vorhanden ist, so müssen SFF und S ein Element aus dem Cache entfernen und es durch d_{j+1} ersetzen. Wenn beide Strategien das gleiche Element wählen, so setzen wir wieder $S' = S$ und sind fertig.

Also nehmen wir an, dass S das Element e entfernt, während SFF das Element f entfernt. Wir beschreiben nun die Konstruktion von S'. In den ersten $j+1$ Schritten stimmt S' mit SFF überein. Danach verhält sich S' zunächst genauso wie S. Solange die Elemente e und f keine Rolle spielen, sind die Caches von S und S' fast gleich, bloß dass der Cache von S das Element f enthält und der Cache von S' das Element e. Ein Problem gibt es erst, wenn zum ersten Mal einer der folgenden Fälle eintritt:

1. Zugriff auf ein Element $g \neq e, f$, das nicht im Cache ist, und S entfernt f. Dann entfernt S' das Element e. Danach sind die Caches gleich, und S' kann sich wie S verhalten. Die Anzahl der Cache-Misses von S' und S ist gleich.

2. Zugriff auf das Element e. Das ergibt einen Cache Miss bei S, aber nicht bei S'. Nehmen wir an, S entfernt das Element g aus dem Cache, um e einzulagern. Es gibt zwei Fälle:

   (a) $g = f$: Dann sind hinterher die Caches von S und S' gleich, und S' kann sich danach komplett wie S verhalten. Die Strategie S' hat sogar weniger Cache-Misses als S.

   (b) $g \neq f$: Nun entfernt S' das Element g aus dem Cache und lädt statt dessen f. Danach sind die Caches wieder gleich und S' kann von nun an S simulieren und hat die gleiche Anzahl von Speicherzugriffen. Leider ist das so konstruierte S' keine reduzierte Strategie. Wir können aber Behauptung 1 benutzen, um aus S' eine reduzierte Strategie zu machen, ohne die Anzahl der Speicherzugriffe zu erhöhen und ohne die ersten $j + 1$ Ersetzungen zu ändern.

3. Zugriff auf das Element f. Dieser Fall kann nicht eintreten, da wir nach Konstruktion von SFF erst auf e zugreifen, bevor wir auf f zugreifen.

□

Um die Optimalität von SFF zu beweisen, kann man nun eine optimale reduzierte Strategie S^* nehmen und mit dem Austauschlemma sukzessive in SFF überführen. Da sich die Anzahl der Speicherzugriffe nicht erhöht, ist auch SFF optimal. Man kann auch eine optimale reduzierte Strategie S^* nehmen, die auf einem maximalen Präfix von Zugriffen mit SFF übereinstimmt. Wenn $S^* = \text{SFF}$ ist, sind wir fertig. Der Fall $S^* \neq \text{SFF}$ ist aber unmöglich, weil wir sonst durch das Austauschlemma die Länge der Übereinstimmung erhöhen könnten, ohne die Anzahl der Speicherzugriffe zu ändern.