Write Optimization, Popcorn, and Maybe Dynamic Optimality

Michael A. Bender, Guy Even, Martín Farach-Colton, Przemysław Uznański, and ...
hopefully Rob and Alex

Abstract

Our goal is to analyze online paging in an external memory setting

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Problem Definition: Static Set with Search Queries

Let $\mathcal{U}$ denote a universe with a linear order. Let $S \subseteq \mathcal{U}$ denote a subset of $\mathcal{U}$. Assume that S is stored in a search tree $T = (V, E)$ with the following properties:

1. Every internal node $v \in V$ has two parts:
   - a ***buffer*** that can store B items from $\mathcal{U}$, and
   - a list of λ pointers of the form (x, p), where $x \in \mathcal{U}$, and p is a pointer to a child of $u \in V$.
2. Every leaf node $v \in V$ contains a set of B items.

The contents of the leaves of T are fixed. Thus, tree T stores a static set S (no insertions or deletions). However, the contents of the buffers in internal nodes can vary.

Our goal is to manage these buffers to reduce the cost of I/O accesses.

We consider an online setting in which the input consists of a sequence of search requests $\sigma = \{r_t\}_{t=1}^{\infty}$, where every request r_i is an element in S (the set stored by search tree T).

Notation. Let $\mathsf{path}(v)$ denote the path in the search tree from the root to v. et $W(v)$ denote the set of items stored in v (if v is an internal vertex, then $W(v)$ is the contents of the buffer of v, if v is a leaf, then $W(v)$ is the contents of the subset stored in v). Let $S_{t+1}(v)$ denote the set $W(v)$ after the request r_t is processed. For an item $x \in A$, let $\mathsf{leaf}(x)$ denote the leaf in which x is stored.

Every request r_t is processed by finding the first vertex $v_t \in \mathsf{path}(\mathsf{leaf}(r_t))$ such that $r_t \in S(v_t)$. Thus the number of I/O's needed to process request r_t is the length of the subpath $\mathsf{path}(v_t)$.

As a result of request r_t both the algorithm and the optimal solution may change the contents of the buffers along the subpath $\mathsf{path}(v_t)$ provided that:

$$\forall v \in \mathsf{path}(v_t) : S_t(v) \subseteq \bigcup_{u \in \mathsf{path}(v_t)} S_{t-1}(u).$$

Our goal is to design an online algorithm for this problem with a constant competitive ratio.

1.1 Extension to Insertions and Deletions

Introducing insertion and deletion requests means that the search tree T is not fixed because the set it stores changes over time. Thus the problem becomes a combination of self-adjusting and caching.

We focus on write-optimized data structures in which insertion and deletion requests are "lazy". Namely, an insert request $r_t = (\mathsf{insert}, x)$ is served by adding x to $S(\mathsf{root})$. A delete request $r_t(\mathsf{delete}, x)$ is served by adding a "tombstone" $\bar{x}$ to $S(\mathsf{root})$. [One resolves a meeting between $\bar{x}$ and x in $W(v)$ by giving precedence to the newer element. If x is newer, then $\bar{x}$ is removed from $W(v)$. If $\bar{x}$ is newer, then x is removed from $W(v)$. If v is a leaf, then $\bar{x}$ is also removed from $W(v)$.]

1.2 Element Types

A buffer may contain the following types of elements (where $x \in \mathcal{U}$):

1. $(x, original)$ - this means that this is an original copy of x that is a witness for $x \in A$. It may be deleted only by a tombstone element $\bar{x}$.
2. $(x, replica)$ - this means that this is a replica of an element $(x, original)$. We refer to such elements as replicants.
3. $\bar{x}$ - a tombstone for x.
4. Empty intervals $((x_1, x_2), empty)$ where $x_1, x_2 \in A$.

2 Constant-Competitive Ratio for Queries

In this section we show that popcorning is constant optimal compared to an optimal offline popcorning algorithm. In this section, we just deal with point queries. Later on, we'll extend to insertions and finger queries.

▶ Claim 1. $\mathrm{OPT}_B(queries) = \Theta(layer - inclusion - \mathrm{OPT}_{c \cdot B}(queries))$

Proof. The proof is a based on simulating an optimal solution that uses buffers of size B with a layer-inclusive solution with buffers that are slightly bigger. ◀

▶ Claim 2. Bottom-up popcorn LRU is layer inclusive.

Proof. Assume otherwise, that there is an item x that is contained in a parent node v but not in a corresponding child node w. The only way this happens is by evicting x from w without evicting from v at particular step t. Since x is evicted from w, there are B newer items $x_1, x_2, \ldots, x_B$ in v, with x_B accessed at exactly time t. Since v handles a superset of items that w was handling, all items were handled by v. Since x was not evicted from v, $x_1, x_2, \ldots, x_B$ were not evicted as well, a contradiction since $B + 1$ exceeds capacity of v. ◀

▶ Claim 3. Bottom-up popcorn LRU with buffer size $c'B$ is $O(1)$-competitive wrt $layer - inclusion - opt_{c \cdot B}(queries))$

▶ Claim 4. move to top combined with greedy flushing is not worse than Bottom-up popcorn LRU

We conclude that

▶ **Theorem 1.** *move to top combined with greedy flushing is with buffer size $c'cB$ is $O(1)$-competitive with OPT_B with respect to queries.*

2.1 Layering

▶ **Definition 2.** The i'th level in a a tree T rooted at r is the set of nodes whose distance from r equals i.

We denote the i'th level by $\mathsf{level}_i(T)$. We omit T when it is clear from the context.

▶ **Definition 3.** The j'th *layer* of a rooted tree T is defined by

$$\mathsf{layer}_j \triangleq \bigcup_{i \in (2^{j-1}, 2^j]} \mathsf{level}_i.$$

2.2 Layer Inclusive Property

▶ **Definition 4.** The buffers in a search tree satisfy the *layer inclusion property* if the following two properties hold:

1. For every replicant y and every layer j, y appears at most once in $\{W(()v) \mid v \in \mathsf{layer}_j\}$.
2. For every $j \geq 1$, $\bigcup_{v \in \mathsf{layer}_{j-1}} \subseteq \bigcup_{v \in \mathsf{layer}_j}$.

3 TO DO

1. consider predecessor and successor queries
2. consider finger bounds

Acknowledgments. We would like to thank ourselves for some illuminating discussions at early stages of the work. We would also like to thank ourselves for making ourselves aware of some of the central theorems in this paper, without which this paper would never have been written.

Finally, we would like to thank our husbands, wives, and version-2 girlfriends for supporting our travel without too much unnecessary kvetching.