Journal of Computer and System Sciences 82 (2016) 627–653

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

On list update with locality of reference ☆

Susanne Albers [a,*,1], Sonja Lauer [b]

[a] Department of Computer Science, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
[b] Department of Computer Science, University of Freiburg, Georges Köhler Allee 79, 79110 Freiburg, Germany

ARTICLE INFO

Article history:
Received 10 July 2012
Accepted 30 January 2015
Available online 13 January 2016

Keywords:
Data structures
Self-organizing lists
Online algorithms
Competitive analysis
Locality of reference

ABSTRACT

We present a comprehensive study of the list update problem with locality of reference. More specifically, we present a combined theoretical and experimental study in which the theoretically proven and experimentally observed performance guarantees of algorithms match or nearly match. Firstly, we introduce a new model of locality of reference that closely captures the concept of runs. Using this model we develop refined theoretical analyses of popular list update algorithms. Secondly, we present an extensive experimental study in which we have tested the algorithms on traces from benchmark libraries. The theoretical and experimental bounds differ by just a few percent. Our new theoretical bounds are substantially lower than those provided by standard competitive analysis. It shows that the well-known Move-To-Front strategy exhibits the best performance. Its refined competitive ratio tends to 1 as the degree of locality increases. This confirms that Move-To-Front is the method of choice in practice.

© 2015 Elsevier Inc. All rights reserved.

1. Introduction

The list update problem is one of the most extensively studied online problems, with a tremendous body of literature published over the past 40 years. The problem has been investigated with respect to both average-case and worst-case competitive analysis. We refer the reader to [1,4–6,11,17,26,29,31,34–36,39] for a selection of some key results.

The list update problem consists in maintaining a set of items as an unsorted linear list. More specifically, a linear linked list of items is given. A list update algorithm is presented with a sequence of *requests* that must be served in their order of occurrence. Each request specifies an item in the list. In order to serve a request, a list update algorithm must *access* the requested item, i.e. it has to start at the front of the list and search linearly through the items until the desired item is found. Accessing the i-th item in the list incurs a cost of i. Immediately after an access, the requested item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called *free exchanges*. All other exchanges of two consecutive items in the list cost 1 and are called *paid exchanges*. The goal is to serve the request sequence so that the total cost is as small as possible. We emphasize that this is the standard cost model, see also [36]. Of particular interest are *online algorithms* that serve each request without knowledge of any future requests.

While early work on the list update problem evaluated online algorithms assuming that requests are generated according to probability distributions, research over the past 20 years has focused on *competitive analysis* [36]. Here an online algorithm is compared to an optimal offline algorithm. Given a request sequence σ, let $A(\sigma)$ denote the cost incurred by online

☆ An extended abstract of this paper appeared in *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP)*, 2008.

* Corresponding author.
E-mail addresses: albers@in.tum.de (S. Albers), sonja.lauer@informatik.uni-freiburg.de (S. Lauer).

[1] Work supported by the German Research Foundation, grant AL 464/7-1.

http://dx.doi.org/10.1016/j.jcss.2015.11.005
0022-0000/© 2015 Elsevier Inc. All rights reserved.

algorithm A in serving σ, and let $OPT(\sigma)$ denote the optimum offline cost. Algorithm A is called c-competitive if there exists a constant α such that $A(\sigma) \leq c \cdot OPT(\sigma) + \alpha$ holds for all σ and all size lists.

In 1985 Sleator and Tarjan proved that the *Move-To-Front* algorithm is 2-competitive [36]. This elegant strategy simply moves an item to the front of the list whenever it is requested. Since then, algorithms with an improved competitiveness have been developed. While the competitive ratios are of course constant, there is a substantial gap between the theoretical bounds and the performance ratios of the algorithms observed in practice. Moreover, *Move-To-Front* often outperforms other strategies, see e.g. [10,11]. The reason is that competitive analysis considers arbitrary request sequences, whereas sequences arising in practice have a special structure: They exhibit locality of reference, meaning that at any point in time only a small set of items is referenced.

There has been considerable research interest in studying the paging problem with locality of reference [3,7,13,19,24, 25,30,32] because, in paging, the gap between the theoretical and experimental performance values is even super-constant. However, hardly any work has been presented for the classical list update problem. In fact, references [10,28] point out that locality is an essential aspect in the list update problem and that a good model is required to properly evaluate the performance of algorithms.

Previous results: We focus on the results that have been developed in the framework of competitive analysis. As mentioned above Sleator and Tarjan [36] showed that *Move-To-Front* is 2-competitive. This is the best factor deterministic online algorithms can achieve [31]. Bachrach and El-Yaniv [9] devised deterministic *MRI* and *PRI* families of algorithms. These families attain competitive ratios of 2 and 3, respectively. We next turn to randomized algorithms. The first randomized strategy was presented by Irani [29]. Her *Split* algorithm is 1.9375-competitive. Reingold et al. [34] presented an elegant *BIT* algorithm that is 1.75-competitive. This factor is substantially below the deterministic bound of 2. The *BIT* algorithm can be generalized to a family of *Counter* strategies [34]. A *Timestamp* family of algorithms was developed in [1]. It achieves a competitiveness equal to the Golden Ratio $\Phi \approx 1.62$. The best randomized algorithm currently known is *COMB* which is 1.6-competitive [4]. Interestingly, *COMB* is a combination of *BIT* and a (deterministic) element of the *Timestamp* family. The factor of 1.6 is close to best lower bound of 1.50084 developed by Ambühl et al. [6] on the performance of randomized list update algorithms.

Experimental studies for the list update problem have been presented by Rivest [35], Bentley and McGeoch [11] and Bachrach et al. [10]. They analyzed popular algorithms on request sequences generated by probability distributions and Markov sources, on sequences derived from text and Pascal files as well as on sequences extracted from the Calgary Corpus [16]. The results are not unanimous. A conclusion is that the ranking of algorithms depends on the degree of locality in the input.

The only prior work addressing list update with locality of reference was a paper by Angelopoulos et al. [8]. They adapted a locality model [3] introduced for the paging problem and proved that *Move-To-Front* is superior to other algorithms.

Our contribution: We present a comprehensive study of the list update problem with locality of reference. The goal is to provide a refined analysis of the problem in which theoretical and empirical results match or nearly match. To this end our study integrates theoretical and experimental work.

First, in Section 2, we introduce a new model of locality of reference that is based on the natural concept of *runs*. A run is a sequence of requests to the same item. We define a number of parameters that characterize request sequences in terms of the occurrence of long runs. Using these parameters we will be able to accurately estimate the performance of list update algorithms. We also define a model of so-called λ-*locality* that characterizes classes of input sequences with respect to their degree of locality. Loosely speaking, the more long runs there are, the higher the locality. As we shall see, our new concepts properly capture locality of reference in the list update problem, both from a theoretical and practical point of view.

In Section 3 we present refined theoretical analyses of list update algorithms. We concentrate on the most popular strategies that have received the most attention recently, namely *Move-To-Front*, *BIT* and *COMB*. In order to be able to analyze *COMB*, we have also evaluated a member of the *Timestamp* family. Of course, we have also investigated an optimal offline strategy. For each algorithm we have analyzed the total service cost incurred on a request sequence, where cost is expressed in terms of our new locality parameters. Interestingly, for *Move-To-Front* our cost analysis is exact, i.e. our locality model is powerful enough to exactly quantify *Move-To-Front*'s service cost on any request sequence. Furthermore, for each online algorithm, we have evaluated its performance relative to that of an optimal offline algorithm. Here *Move-To-Front* achieves an excellent performance ratio and responds well to locality of reference: The competitiveness even tends down to 1 as the degree of locality increases. This does not hold true for the other online algorithms.

In Section 4, we present a comprehensive experimental study in which we have evaluated our list update algorithms on real-world traces from benchmark libraries. Obviously, the list update problem is a solution to the classic dictionary problem. In this context, in practice, requests are memory accesses. Secondly, list update has interesting applications in data compression, see e.g. [12,15]. For instance, the open source data compression program `bzip2` relies on *Move-To-Front* encoding in combination with a preceding Burrows–Wheeler transformation. Therefore, in our experiments we consider as input (a) memory access strings (47 traces) and (b) sequences arising in data compression routines (44 traces). In our tests we first analyze the traces with respect to their locality characteristics. It shows that the parameters introduced in Section 2 are indeed sensible.

Next, in the experiments, for each algorithm and each input sequence, we have computed the total service cost. Furthermore, for each online algorithm and each input, we have determined the *experimentally observed competitiveness*, which is the total service cost of the algorithm divided by the total cost incurred by an optimal offline strategy. Since the offline

Download English Version:

https://daneshyari.com/en/article/10332684

Download Persian Version:

https://daneshyari.com/article/10332684

Daneshyari.com