A Distributed List Update Problem

Sunil M. Shende
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588.
Email: sunil@calypso.unl.edu

Rahul Simha
Department of Computer Science
College of William and Mary
Williamsburg, VA 23185.
Email: simha@cs.wm.edu

Abstract

The list update problem is studied in a distributed environment where the set of items is partitioned across two processors and in which the cost of accessing an item is a combination of a list searching cost and a communication cost. A natural generalization of the Move-To-Front (**MTF**) algorithm, called Distributed Move-To-Front (**DMTF**), is analyzed under two models of information available to the algorithm. In the global knowledge model (**DLU**[**GK**]), the algorithm knows the partition of items in advance, while in the local knowledge model (**DLU**[**LK**]), this information is unknown to the algorithm. Both competitive and expected value analyses are given. It is shown that for an n-item set, the performance of **DMTF** is a factor of $O(n)$ worse in the local knowledge model than in the global knowledge model.

1 Introduction

The list update problem, the problem of rearranging a singly-linked list to improve average access times, has been subject to considerable analysis in the literature. Each analysis has usually fallen into one of two categories: an expected value analysis using independent accesses drawn from a distribution over the elements in the list [3, 6, 8, 10, 11, 14], or a competitive analysis in which no assumption is made about the distribution of accesses [2, 9, 13]. In particular, for the well-known Move to Front (MTF) heuristic (which always moves the currently accessed item to the front of the list) the following facts have been shown: the expected performance of MTF is no worse than twice that of the optimal static arrangement, and MTF is 2-competitive with the optimal off-line algorithm, that is, it performs within a factor of two of the optimal algorithm on every sequence of accesses. Both these approaches have resulted in several generalizations of the basic list update problem [1, 4, 12] – see [7] for a survey.

In this paper, we consider a Distributed List Update (DLU) problem, a generalization of the list update problem in which a singly linked list is partitioned over two processors and accesses to items are generated from both processors. We study a distributed version of the MTF heuristic (called Distributed MTF, or DMTF) under two models, one in which only local knowledge is known and one in which some global knowledge about the partition is known. We present both expected value and competitive analyses of the DLU problem. Interestingly, as in the single processor case, our main result is borne out by both types of analysis: we show that on lists of total size n, the performance of the local knowledge version of DMTF is a factor of $O(n)$ worse than that of the global knowledge version.

The **DLU** problem is concerned with maintaining a set of items partitioned across two processors: the subset on each processor is arranged in the form of a rearrangeable linked list. Let L be a set of n items, in which each item contains a unique *key* along with some satellite data. For the remainder of the paper, we will assume that an item is identified with its key and that $L = \{1, 2, \ldots, n\}$ describes the items. At any given time, L is partitioned over two processors with each partition arranged in a linked list on its respective processor. Requests for inserting, deleting and accessing items arrive one at time at either of the two processors. An access request is satisfied by finding the item requested on one of the two lists, followed by a possible rearrangement of the set partition and the order of items in each partition. An insertion or a deletion requires an access followed by an insertion or deletion of the item, depending on whether or not the item was found on one of the two lists. Since accesses are the most general of the three types of requests, we shall assume that every request is an access and omit the type of the request; the analysis can be easily extrapolated to insertions and deletions.

Any algorithm for **DLU** incurs two kinds of costs on each request: the cost of searching through the lists and the communication cost incurred when transmitting information from one processor to another. We shall assume that the latter cost is always a fixed constant c per communication round. The list access cost depends on the number of items scanned before locating the requested item, and any re-adjustments made locally to the lists. In particular, the algorithm is allowed to re-adjust the lists during each access using *free* and *paid* exchanges. Briefly, a free exchange allows an item on a list to be moved closer to the front of the list or to the front of the other list at no extra cost; any other move is termed a paid exchange and has unit cost (for a more complete discussion on this model, see [13]). Note that a free exchange that transfers an item to the other list still incurs a communication cost of c. Also, we will be interested in analyzing only the list access and communication costs for this problem; we assume the requests will be satisfied one after another. Thus, we do not consider issues of parallelism, concurrency and queueing; our view corresponds to serializations of system behaviors.

A request sequence is a sequence of tuples $\rho = (\alpha_1,\ l_1),\ (\alpha_2,\ l_2),\ \ldots(\alpha_m,\ l_m)$ where the k^{th} tuple consists of the requested item, $\alpha_k \in L$, and the processor at which the request originated, $l_k \in \{1, 2\}$. On any request, $(\alpha_k,\ l_k)$, on processor l_k, an algorithm X locates α_k on one of the two lists. The cost incurred by the algorithm on the request is the cost of searching one or both of the lists along with the cost of communication (if the search moves over to the other processor). The net cost to the algorithm over the entire sequence ρ is denoted by $C_X(\rho)$. If X is an on-line algorithm (one that cannot see future requests) and if $C_{OPT}(\rho)$ denotes the optimal cost of processing a request sequence ρ, X is said to be γ-*competitive* if $\lim_{m\to\infty} \sup_{\rho:|\rho|=m} \frac{C_X(\rho)}{C_{OPT}(\rho)} \leq \gamma$.

We shall study the **DLU** problem under two different informational models. Consider an access to an item α. In each case, the algorithm is oblivious of the precise position of α on its list. The following property distinguishes the two models:

- In the model **DLU**[**LK**] (Distributed List Update with Local Knowledge), the algorithm does not know whether α is on processor 1 or 2. Consequently, when an access request originates on some processor, the list on that processor is searched first. If the item is not found there, then the search moves to the other list.

- In the model **DLU**[**GK**] (Distributed List Update with Global Knowledge), an access request can directly cause searching to be initiated on the list which contains the requested item. However, a communication cost is still incurred if the access causes a search of a list on a different processor than the one which originated the request.

Loosely speaking, the partition is 'known' in **DLU**[**GK**] but not in **DLU**[**LK**]. Note that **DLU**[**LK**] is likely to be more practical; the **DLU**[**GK**]model captures the situation where for a large keys, it may be effective to store a simple hash table representing the partition knowledge. In either case, it is desirable to have items migrate to the processor from which accesses predominate and also take advantage of locality in the request sequence.

For any algorithm X denote by L^i_X, the ordered list of items on processor i ($i = 1, 2$); the *configuration* for algorithm X is denoted by the pair (L^1_X, L^2_X). Figure 1 provides a simple example of a configuration of the items, $\{1, 2, 3, 4, 5\}$. The positions of items 3, 4 and 5 on their lists, for example, are 1, 3 and 2 respectively.

Processor	Linked list
1	1 5 4
2	3 2

Figure 1: A configuration of a **DLU** algorithm

In the configuration shown in Figure 1, the request $(3,1)$ would cost at least $(3 + c + 1 + c) = 2c + 4$ in the **DLU**[**LK**] model, since list 1 will be searched unsuccessfully before the search moves across to list 2. The corresponding least cost in the **DLU**[**GK**] model is $(c + 1 + c)$ because global knowledge allows the search to move to list 2 immediately. Note that there are two charges for communication: once to initiate search on the other processor, and then to "return" the result of the search back to the requesting processor (the algorithm may still not physically move the item across to the other processor).

We consider the following *distributed* version of the well-known Move-to-Front heuristic for updating a single list: **DMTF** always moves the currently accessed item to the front of the list on the processor from which the access came. For the example described above, **DMTF** would incur costs $(4 + 2c)$ and $(1 + 2c)$ respectively in the models **DLU**[**LK**] and **DLU**[**GK**]; the resulting configuration would consist of items 3, 1, 5 and 4 on the first list and item 2 by itself on the second list.

2 Competitive analysis

2.1 The local knowledge model

We first focus on the competitiveness of **DMTF** in model **DLU**[**LK**]. Consider the following initial configuration for n items: list 1 contains the lone item 1, while the remaining items 2 through n appear on list 2 in increasing order. Suppose that the first two requests are $(1,2)$ followed by $(1,1)$, i.e. the item 1 is first requested from processor 2 and then from processor 1. The former request $(1,2)$ costs **DMTF** a total of $n-1+c+1+c=n+2c$ units, and leaves list 1 empty and list 2 with the items 1 through n in order. The latter request reestablishes the starting configuration at a cost of $2c+1$.

Now, consider the sequence of $2m$ requests $\rho = (1,2),\ (1,1),\ (1,2),\ (1,1)\ldots$. Clearly the net cost to **DMTF** is exactly $m(n+2c+2c+1)$. On the other hand, the optimal strategy for this sequence under model **DLU**[**LK**] is to move item 1 to processor 2 and leave it there. It is easy to see that the first request costs $n+2c+1$, but thereafter, every request costs either 1 or $c+1$ depending on whether the request is an even-numbered or odd-numbered one. The optimal cost yields a value of $n+2c+1+m+(m-1)(c+1)=n+(m+2)c+2m-c$. It is easy to see that the amortized cost for **DMTF** is $O(n)$ per request whereas the amortized cost for the optimal algorithm is just $O(1)$. Thus,

Lemma 1 **DMTF** *is not competitive in the model* **DLU**[**LK**].

2.2 The global knowledge model

In this section, we show that **DMTF** is competitive for **DLU**[**GK**]. Since global knowledge is assumed, **DMTF** can always start searching for α on the list that contains α regardless of whether the request for α originated on the processor containing α or on the other processor. In the latter case, we simply charge **DMTF** the extra cost of communication. We shall analyze the communication and list processing costs for **DMTF** separately and show that these costs are within a constant factor of 4 of the the corresponding costs incurred by any other algorithm X.

Theorem 1 **DMTF** *is 4-competitive in the model* **DLU**[**GK**].

Proof: First, we show that DMTF's list costs are competitive. Consider the configurations of **DMTF** and X at some instant in time. Let $\phi(\alpha)$ denote the number of elements that would be scanned by **DMTF** but not by X when both search for α in their lists (note that in the **DLU**[**GK**] model, the origin for the request only affects the communication cost). Also, let k_α and i_α denote the respective positions of α in their lists in the configuration for **DMTF** and for X.

We will follow the banker's method to amortize costs of the algorithm over a sequence of requests. Two kinds of accounts are established for **DMTF** . The first is a general credit account, Φ, to be shared by the two lists of **DMTF** . The second kind of account is a family of debit accounts, one per item: the account corresponding to item α is denoted by $D(\alpha)$. We shall enforce the following invariants on the accounts just prior to any request:

> **Invariant 1**: The account Φ has at least $\sum_\alpha \phi(\alpha)$ credits.
> **Invariant 2**: For each item α, the account $D(\alpha)$ has at most k_α debits.

To process a request, we shall borrow from the bank to pay for the access cost; we will also borrow from the bank or individual debit accounts to maintain Invariants 1 and 2; the net borrowings plus the net debit in the accounts $D(\alpha)$ is an upper bound on the net cost of **DMTF** .

Without loss of generality, consider an access for element α from processor 1. Observe that DMTF's list search cost on a request to item α is exactly k_α. Let $E(\alpha)$ be the set of elements excluding α, that are scanned by both algorithms while searching for α, and define for $i = 1, 2$: $y_i = \mid E(\alpha) \cap L^i_{DMTF} \mid$. In words, among common elements scanned, y_1 of them are on DMTF's list 1 and y_2 of them are on list 2. Note that for every request, at least one of y_1 or y_2 is zero depending on whether α is on list 2 or list 1 of **DMTF** . There are four possible cases that we analyze next.

1. $\alpha \in L^1_X,\ \ \alpha \in L^1_{DMTF}$: Before α is moved to the front of processor 1 by DMTF, there are $k_\alpha - 1 - y_1$ elements scanned during the search for α by **DMTF** but not by X. These will no longer be scanned by **DMTF** in a search for α. Hence, $\phi(\alpha)$ decreases by $k_\alpha - 1 - y_1$ to zero. For each $\beta \in (E(\alpha) \cap L^1_{DMTF})$, the value of $\phi(\beta)$ increases by 1 after the move since unlike X, **DMTF** will have to scan α before locating β. Thus, the total change in Φ is $y_1 - (k_\alpha - 1 - y_1)$. Borrowing

an additional k_α units to pay for the search, the total borrowing required is $2y_1 + 1$. Since $y_1 < i_\alpha$, the total borrowing is less than $2i_\alpha$.

2. $\alpha \in L_X^1, \ \ \alpha \in L_{DMTF}^2$: In this case, $\phi(\alpha)$ was $k_\alpha - 1 - y_2$ before the move and becomes zero as a result of the move. In addition,

   (a) For each $\beta \in L_{DMTF}^1, \ \ \phi(\beta)$ can increase by at most 1 since α is now at the front of that list. This increase will be paid for by borrowing 1 unit from the corresponding $D(\beta)$ account and crediting it to the general account.

   (b) For each β behind α's old position on list 2, credit the corresponding account $D(\beta)$ by 1 unit.

   Thus, the total borrowing including the list search cost is $-(k_\alpha - 1 - y_2) + k_\alpha$, and hence less than $2i_\alpha$ (since $y_2 < i_\alpha$).

3. $\alpha \in L_X^2, \ \ \alpha \in L_{DMTF}^1$: The analysis is similar to that of case 1.

4. $\alpha \in L_X^2, \ \ \alpha \in L_{DMTF}^2$: The analysis is similar to that of case 2.

It is clear that Invariant 1 has been maintained at all times. Moreover, during each request, we have also maintained Invariant 2 for all the items except α, the item accessed, because any item β's debit is no larger than the number of items ahead of it in its list, i.e., no larger than k_β. To enforce the invariant for α, we must reduce α's debits from at most k_α to 1 after the move. This can at most double the borrowing for the cost of the operation (k_α). Next, each free exchange by X can only decrease the general account by 1 unit (which comes as a free credit to DMTF) and each paid exchange by X can only increase the general account by 1 (paid for by X's cost in performing the paid exchange). Then, following the arguments of Theorem 1 in [13], DMTF's list costs are within a factor of 4 of X's list costs.

Finally, we show that the net communication cost incurred by **DMTF** is at most twice the corresponding cost for X. For any configuration of **DMTF** and X's lists, define $\phi_c(\alpha)$, the communication potential for α, as follows: $\phi_c(\alpha) = 2c$ if **DMTF** and X have α on different processors; $\phi_c(\alpha) = 0$ otherwise. At step i, let $\Phi_c(i) = \sum_\alpha \phi_c(\alpha)$

be the total communication potential, and let $T_X(i)$ and $T_{DMTF}(i)$ denote the respective communication costs incurred by X and **DMTF** . If X does not move any item across processors, then by analyzing four cases similar to the four in the list costs, it is easy to show that $T_{DMTF} + \Phi_c(i) - \Phi_c(i-1) \leq 2T_X(i)$. Now consider the change in potential when X moves an item across processors; each such change causes the potential to increase by at most $2c$. Summing all such changes, the difference between the final and initial potential is at most $2nc$, where n is the number of elements. Dividing by m, the number of requests, and taking limits establishes the 2-competitiveness of the communication costs. ∎

Thus, there is a factor of $O(n)$ increase in the competitive ratio going from the model **DLU[GK]** to the model **DLU[LK]**, an increase also found in the expected value analysis in the next section. We remark that, in the proof above, the general account when specialized to a the single processor case corresponds to the potential function in [13].

3 Expected value analysis with independent accesses

We now analyze the cost of DMTF using the independent access model. Define the following events:

$$
\begin{aligned}
A &= \text{“Current access originated on processor 1”} \\
B &= \text{“Current access originated on processor 2”} \\
A_i &= \text{“Item } i \text{ is on processor 1”} \\
B_i &= \text{“Item } i \text{ is on processor 2”} \\
C_i &= \text{“Current access is for item } i\text{”}
\end{aligned}
$$

Using $P[\cdot]$ to denote probability, under the independent access model we will assume that $p_i = P[C_i \cap A]$ and $q_i = P[C_i \cap B]$ are given as inputs to the problem. Note that $\sum_{i=1}^{n}(p_i + q_i) = 1$ and that $q_i = 0$ for the single processor case.

Observe that $P[C_i \cap A | C_i] = \frac{p_i}{p_i+q_i}$ and $P[C_i \cap B | C_i] = \frac{q_i}{p_i+q_i}$. Then, the 'steady-state' probability of finding item i on processor 1 when using DMTF is found by solving the Markov chain whose transition matrix is

$$\begin{bmatrix} \frac{p_i}{p_i+q_i} & \frac{q_i}{p_i+q_i} \\ \frac{q_i}{p_i+q_i} & \frac{p_i}{p_i+q_i} \end{bmatrix}.$$

We obtain $P[A_i] = \frac{p_i}{p_i+q_i}$. Similarly, $P[B_i] = \frac{q_i}{p_i+q_i}$.

Next, let Z_L denote the cost of an access in steady-state under the local-knowledge model. Let $\mathbf{E}[\cdot]$ denote expectation; the following theorem gives an expression for $\mathbf{E}[Z_L]$.

Theorem 2

$$\mathbf{E}[Z_L] = 1 + (n - 1 + 2c) \sum_i \frac{p_i q_i}{p_i + q_i} + \sum_i \sum_{k \neq i} \frac{p_i p_k + q_i q_k}{p_i + p_k + q_i + q_k}$$

Proof: Write the expected value of Z_L as follows:

$$\begin{aligned} \mathbf{E}[Z_L] &= \sum_i \mathbf{E}[Z_L | C_i \cap A]\, p_i + \mathbf{E}[Z_L | C_i \cap B]\, q_i && (1) \\ &= \sum_i \left[\left(\mathbf{E}[Z_L | C_i \cap A | A_i] \frac{p_i}{p_i + q_i} + \mathbf{E}[Z_L | C_i \cap A | B_i] \frac{q_i}{p_i + q_i} \right) p_i \right. \\ &\quad \left. + \left(\mathbf{E}[Z_L | C_i \cap B | A_i] \frac{p_i}{p_i + q_i} + \mathbf{E}[Z_L | C_i \cap B | B_i] \frac{q_i}{p_i + q_i} \right) q_i \right]. && (2) \end{aligned}$$

Next, following the approach in [6] observe that

$$\mathbf{E}[Z_L | C_i \cap A | A_i] = 1 + \sum_{k \neq i} \mathbf{E}\left[U_k^{(1)}\right]$$

where, for $j = 1, 2$:

$$U_k^{(j)} = \begin{cases} 1, & \text{if item } k \text{ is before item } i \text{ on processor j} \\ 0, & \text{otherwise.} \end{cases}$$

Now, item k is *before* item i if and only if the previous access to either one was an access to item k from processor 1. Thus,

$$\mathbf{E}\left[U_k^{(1)}\right] = P\left[U_k^{(1)} = 1\right] = \frac{p_k}{p_k + q_k + p_i + q_i}.$$

Similarly, let

$$V_k^{(j)} = \begin{cases} 1, & \text{if item } k \text{ is before item } i \text{ on processor j} \\ 0, & \text{otherwise.} \end{cases}$$

Then,

$$\mathbf{E}\left[V_k^{(1)}\right] = P\left[V_k^{(1)} = 1\right] = \frac{p_k}{p_k + q_k}$$

and

$$\mathbf{E}\left[Z_L | C_i \cap A | B\right] = \left(\sum_{k \neq i} \mathbf{E}\left[V_k^{(1)}\right]\right) + \left(1 + \sum_{k \neq i} \mathbf{E}\left[U_k^{(1)}\right]\right) + c$$

where the first term is the cost of searching through the item on processor 1, the second term is the search cost on processor 2 (where the item is found) and the third term, c, is simply the communication cost. Thus,

$$\begin{aligned}
\mathbf{E}\left[Z_L | C_i \cap A | A_i\right] &= 1 + \sum_{k \neq i} \frac{p_k}{p_i + p_k + q_i + q_k} \\
\mathbf{E}\left[Z_L | C_i \cap A | B_i\right] &= 1 + c + \sum_{k \neq i} \frac{p_k}{p_k + q_k} + \frac{q_k}{p_i + p_k + q_i + q_k} \\
\mathbf{E}\left[Z_L | C_i \cap B | A_i\right] &= 1 + c + \sum_{k \neq i} \frac{q_k}{p_k + q_k} + \frac{p_k}{p_i + p_k + q_i + q_k} \\
\mathbf{E}\left[Z_L | C_i \cap B | B_i\right] &= 1 + \sum_{k \neq i} \frac{q_k}{p_i + p_k + q_i + q_k}
\end{aligned}$$

Putting these together in equation (2) and rearranging terms, we get our result. ∎

A similar expression for the global-knowledge case can be derived. Let Z_G denote the cost in this case and note that the $V_k^{(1)}$ terms all become zero. The same derivation then leads to the following expression.

Theorem 3

$$\mathbf{E}\left[Z_G\right] = 1 + 2c\sum_i \frac{p_i q_i}{p_i + q_i} + \sum_i \sum_{k \neq i} \frac{p_i p_k + q_i q_k}{p_i + p_k + q_i + q_k}.$$

Note that the difference is $O(n)$ (achieved, for example, when $p_i = q_i = \frac{1}{2n}$). Thus, the expected value analysis appears to confirm the results from the earlier competitive analysis. Note also that both expressions are generalizations of the corresponding single-processor case [10] (Taking $q_i = 0$ for all i gives the expression for the single-processor case).

4 Conclusions

In this paper, we have studied the problem of distributed list management for two processors. We use two models for analyzing a simple distributed strategy, *viz.* the Distributed Move-To-Front (**DMTF**) algorithm. Our investigation was motivated by the general problem of maintaining distributed data structures across a collection of processors: naturally, it would be interesting to analyze the problem in a variety of other situations. We hope our results will be a starting point for deriving efficient algorithms for managing more general data structures such as k-ary search trees on more than two processors.

References

[1] F.d'Amore, A.Marchetti-Spaccamela and U.Nanni. The Weighted List Update Problem and the Lazy Adversary. TR-92-011, International Computer Science Institute, Berkeley, February 1992.

[2] J.L.Bentley and C.C.McGeoch. Amortized Analyses of Self-Organizing Sequential Search Heuristics. *CACM*, Vol. 28, No. 4, April 1985, pp. 404-411.

[3] J.R.Bitner. Heuristics that Dynamically Organize Data Structures. *SIAM J. Computing*, Vol. 8, No. 1, February 1979, pp. 82-109.

[4] P.F.Dietz and D.D.Sleator. Two Algorithms for Maintaining Order in a List. *Proc. ACM STOC*, 1987, pp. 365-372.

[5] G.H.Gonnet, I.Munro and H.Suwanda. Exegesis of Self-Organizing Linear Search. *SIAM J. Computing*, Vol. 10, No. 3, August 1981, pp. 613-637.

[6] W.J.Hendricks. An Account of Self-Organizing Systems. *SIAM. J. Comput.*, Vol. 5, No. 4, Dec 1976, pp. 715-723.

[7] J.H.Hester and D.S.Hirschberg. Self-Organizing Linear Search. *Computing Surveys*, Vol. 17, No. 3, September 1985, pp. 295-311.

[8] M.Hofri and H.Shachnai. Self-Organizing Lists and Independent References: A Statistical Synergy. *J. Algorithms*, Vol. 12, 1991, pp. 533-555.

[9] S.Irani, N.Reingold, J.Westbrook and D.D.Sleator. Randomized Competitive Algorithms for the List Update Problem. *Symp. Disc. Alg.*, 1991, pp. 251-260.

[10] J.McCabe. On a Serial File with Relocatable Records. *Operations Research*, Vol. 12, 1965, 609-618.

[11] R.Rivest. On Self-Organizing Sequential Search Heuristics. *CACM*, Vol. 19, 1976, pp. 63-67.

[12] R.Simha and A.Majumdar. On Lookahead in the List Update Problem. *Inf. Proc. Lett.*, Vol. 50, 1994, pp. 105-110.

[13] D.D.Sleator and R.E.Tarjan. Amortized Efficiency of List Update and Paging Rules. *CACM*, Vol. 28, No. 2, February 1985, pp. 202-208.

[14] R.S.Valiveti and B.J.Oommen. The Move-to-Front List Organizing Heuristic for Non-Stationary Query Distributions. *Proc. Int. Symp. Comp. Info. Sci.*, 1991, Antalya, Turkey, pp. 105-114.