On the Work Function Algorithm for two state task systems

Yossi Azar * Uriel Feige † Suman Nath ‡

June 16, 2007

Abstract

We revisit the known work function online algorithm *WFA* for metrical task systems in the special case when there are only two states. It is well known that this algorithm is 3 competitive. We offer a slightly modified version of this algorithm, and show that our version exactly mimics the migration pattern of the best possible offline algorithm, though the timing of the migration events is somewhat different in these two algorithms. We use this to give a refined bound on the cost of *WFA* in terms of the cost of the optimal offline algorithm. Specifically, we partition the cost of the optimal algorithm to a fixed cost F, inefficiency cost I and migration cost M. The cost of our algorithm is at most $F + I + 3M$. This implies that our version of *WFA* does not only have a worst case competitive ratio at most 3 (which is best possible) but additionally enjoys better competitive ratios on "typical" input sequences.

The current study was motivated by practical problem. We present preliminary experimental evidence that indicates that the algorithm works well in practice.

1 Introduction

Overview: We consider a task system with two states. Our study is motivated by a specific scheme that involves two different states (mechanisms) for accessing data in external memory. One state is optimized for reading data quickly, whereas the other is optimized for writing data quickly. There is a

*azar@tau.ac.il. Microsoft Research, Redmond and Tel-Aviv University.

†uriel.feige@microsoft.com. Microsoft Research, Redmond.

‡sumann@microsoft.com. Microsoft Research, Redmond.

cost involved in switching between states. If a sequence of requests (either read or write) arrives, the optimal way of serving it may involve switching between states multiple times, depending on whether the sequence has alternating long subsequences of (mostly) read and (mostly) write operations.

Related work: Technically, the problem that we study is a generalization of file migration on two states and a special case of metrical task systems, and we build upon earlier work on those topics. There is a $2n-1$ competitive algorithm for metrical task system on n states [5]. For file migration there is a 3 competitive algorithm for complete graphs and trees [3]. Both these earlier algorithms imply a competitive ratio of 3 for our problem. This problem has a simple well known lower bound of 3 on the competitive ratio. (The notion of competitive ratio that we use will be defined shortly.) For more related work on file migration see for example [1, 2, 6, 7].

Our results: All our work and results refer only to task systems with two states. We present a simplified and slightly modified version of the known *work function algorithm WFA* for metrical task systems. (For previous work on the work function algorithm and its "standard" version, see [4] for example). We provide a new proof that *WFA* is 3-competitive in the worst case. In fact, we give a refined bound on the cost of *WFA* in terms of the cost of the optimal offline algorithm. Specifically, we partition the cost of the optimal algorithm to a fixed cost F, inefficiency cost I and migration cost M. The cost of our algorithm is at most $F + I + 3M$. This bound is especially informative, as it shows that the competitive ratio is much better than 3 on many natural distributions of request sequences, and provides an easy methodology of computing the competitive ratio of the algorithm in these cases. As our study was motivated by a specific application we present some preliminary experimental evidence to support our theoretical analysis.

Paper structure: The paper is organized as follows. Section 2 includes formal definitions and notations. The optimal (offline) solution is characterized in section 3. Our version of the work function algorithm and its analysis are presented in Section 4. Some extensions are discussed in Section 5.

2 Definitions and notations

The system has two states: state S_1 and state S_2. A sequence of n requests (x_i, y_i) for $1 \leq i \leq n$ arrives online. Request i can be served at cost x_i in

state S_1 and at cost y_i in state S_2. The serving algorithm may migrate from state S_1 to S_2 at cost M_1 and from state S_2 to S_1 at cost M_2. Upon the arrival of a request, the algorithm has one of two options: either to serve the request at the current state, or to migrate to the other state and then serve the request. (Observe that there is no need to allow the algorithm to first serve the request and then migrate, because this last migration can always be appended at the beginning of the next request.) The information available to an offline algorithm when making a decision is the whole sequence (past and future requests). The information available to an online algorithm is all past requests and the current request, but the algorithm has no information regarding future requests. (Later we shall also discuss how our online algorithms can be adapted to take advantage of a limited amount of lookahead.) The total cost of an algorithm on a sequence of requests is the cost of serving the requests plus the cost of migration. Our goal is to serve the sequence at a minimum total cost.

Given a sequence σ of requests and an algorithm A, we may divide the total cost of A on σ (denoted by $A(\sigma)$) into three components.

The first component is a property of σ alone and is independent of A. We call it the *fixed* cost of σ, and denote it by $F(\sigma)$. It is the sum of costs of serving all requests, if every request is served on the state in which it costs less. Formally, $F(\sigma) = \sum_i \min[x_i, y_i]$.

The second component is the *inefficiency* A_I that results from A serving some requests at their more costlier state. Formally, let z_i be an indicator variable that is 0 if A serves request i at the cheaper state for request i and 1 if A serves request i at the costlier state for request i. Then $A_I(\sigma) = \sum_i z_i \cdot |x_i - y_i|$.

The third component is the *migration* cost A_M which is the total cost of all migrations performed by algorithm A.

Altogether, we have that $A(\sigma) = F(\sigma) + A_I(\sigma) + A_M(\sigma)$, and only the *overhead* quantities $A_I(\sigma) + A_M(\sigma)$ are actually under control of the algorithm.

Let $OPT(\sigma)$ denote the optimal total cost on a sequence σ of requests. Hence $OPT(\sigma) = F(\sigma) + OPT_I(\sigma) + OPT_M(\sigma)$.

We measure the performance of an online algorithm by comparing it to the optimal offline algorithm. We shall now define the notions of competitive ratio that we will be using. For brevity, let us use C to denote the *commute* cost $M_1 + M_2$. This quantity will appear both in our algorithm, and as an additive term in the definition of the competitive ratio. (This additive term is required to handle boundary effects at the beginning and end of σ.)

One way of defining the competitive ratio of an algorithm A is as follows.

An algorithm A has competitive ratio ρ if for every sequence σ, $A(\sigma) \leq \rho OPT(\sigma) + O(C)$. However, in the context of task systems it is customary to use a stronger definition ρ_O, based only on the overhead payed by the algorithm on top of the fixed cost portion $F(\sigma)$. This leads to requiring $A_I(\sigma) + A_M(\sigma) \leq \rho_O(OPT_I(\sigma) + OPT_M(\sigma)) + O(C)$.

With respect to the above "overhead competitive ratio" measure, previous work showed that there are online algorithms (including the work function algorithm) that are 3-competitive (namely, $\rho_O = 3$), and no online algorithm can have a better competitive ratio. However, in this work, we shall use an even stronger notion of competitive ratio. Rather than allowing ρ_O to multiply both the inefficiency cost and the migration cost of OPT, we introduce a competitive ratio measure ρ_M that involves only the migration cost.

Definition 2.1 *Algorithm A is said to have* strong competitive ratio ρ_M, *if for every sequence of requests σ,*

$$A_I(\sigma) + A_M(\sigma) \leq OPT_I(\sigma) + \rho_M OPT_M(\sigma) + O(C)$$

Observe that the optimal offline algorithm may not be unique. It may happen that at some points there would be an option either to migrate or to serve a subsequence of requests at a more costlier state, giving the same total costs using either option. In definition 2.1 we may assume the among all optimal solutions, the one with the smallest number of migration events is chosen.

We show that the work function algorithm achieves $\rho_M = 3$. The advantage of using the measure ρ_M (rather than the previously used overhead competitive ratio ρ_O) is that it more clearly shows how the relative performance of the algorithm improves as a function of properties of the request sequence. The larger the total inefficiency cost $OPT_I(\sigma)$ is compared to the total migration cost $OPT_M(\sigma)$, the closer the ratio $A(\sigma)/OPT(\sigma)$ is to 1, even if the fixed cost $F(\sigma)$ is 0.

3 The optimal offline algorithm

To analyze our online algorithm, it would be helpful to first characterize the optimal offline algorithm. In general metrical task systems, dynamic programming is used in order to find the optimal solution to a given sequence. However, in our two state setting there is a simple rule that always produces

the optimal solution, and among all optimal solutions, the one with fewest migrations.

For request i, let δ_i denote the difference $x_i - y_i$. Hence if $\delta_i < 0$ it is better to serve request i at state S_1 and if $\delta_i > 0$ it is better to serve request i at state S_2. (Without loss of generality, we assume that $\delta_i \neq 0$, because when $\delta_i = 0$ no sensible algorithm will pay more than the fixed cost.) We will use the following convention regarding sequences of requests. Assume that OPT is at state S_2 after it served the final request of σ. Then σ is extended by additional requests, all with $\delta_i > 0$, until the sum of δ_i over all additional requests exceeds C (the commute cost). Likewise, if OPT is at state S_1 after it served the last request, then the sequence of requests is extended by additional requests, all with $\delta_i < 0$, until the sum of δ_i over all additional requests exceeds $-C$. In either case, OPT needs to pay only the fixed costs over this extension, whereas algorithm A might pay some overhead above the fixed costs. Hence the competitive ratio may only get worse by this extension, and hence our convention regarding the existence of the extension can be made without loss of generality.

Cancelling the fixed costs, a sequence of requests can be viewed as a sequence of δ_i. For every $2 \leq t \leq n+1$, define $\Delta(t) = \sum_{i=1}^{t-1} \delta_i$, and define $\Delta(1) = 0$. Point $i \leq n$ is a local minimum of the function Δ if $\delta_{i-1} < 0$ and $\delta_i > 0$. Point $i \leq n$ is a local maximum of the function Δ if $\delta_{i-1} > 0$ and $\delta_i < 0$.

Proposition 3.1 *Every optimal algorithm will migrate from state S_1 to state S_2 only at time steps i that are local minima of Δ, and from state S_2 to state S_1 only at time steps i that are local maxima of Δ.*

Proof: In any other case, either migrating one step earlier or one step later results in lower total cost. Details omitted. □

The following definition involves the key feature of optimal algorithms for the two state task system.

Definition 3.2 *Let i be a time step where Δ has a local minimum. Let $j > i$ be the first time step after i in which $\Delta(j) > \Delta(i) + C$, and $j = \infty$ if no such time step exists. Let $k > i$ be the first time step after i in which $\Delta(k) \leq \Delta(i)$, and $k = \infty$ if no such time step exists. (Observe that it cannot be the case that $j = k = \infty$, by our convention regarding extending the request sequence.) The local minimum at i is said to be* significant *if $j < k$, and* insignificant *if $j > k$. Likewise, a local maximum at i is significant if after i the function Δ reaches a value strictly lower than $\Delta(i) - C$ before it returns to a value of $\Delta(i)$ or higher.*

Lemma 3.3 *If the optimal solution (specifically, the one with fewest migrations among all optimal solutions) is at state S_1, it will migrate to state S_2 at the first time step $j \geq i$ that is a significant local minimum of Δ (if one exists). Similarly, if it is at state S_2 at time step i, it will migrate to state S_2 at the first time step $j \geq i$ that is a significant local maximum of Δ (if one exists).*

Proof: First we show that if the optimal solution is at state S_1 in a significant local minimum of Δ then it will migrate to S_2. Let i denote the time step of this local minimum, let $j > i$ be the first time step in which $\Delta(j) > \Delta(i) + C$ and let $k > i$ be the first time step in which $\Delta(k) \leq \Delta(i)$ (and $k = \infty$ if no such time step exists). Observe that $k > j$ because i is a significant local minimum. The optimal solution must migrate from S_1 to S_2 not later than time step j, because it costs less to migrate from S_1 to S_2 at time step i, serve all request up to request $j-1$ at S_2, and then migrate back to S_1, then it costs to serve all requests from i to $j-1$ at state S_1. Given that the optimal solution must migrate, the migration that results in smallest inefficiency cost is at time i, because $k > j$.

Now we show that the optimal solution need never migrate from S_1 at a time step i that is not a significant local minimum of Δ. Observe that by Proposition 3.1, i can be assumed to be a local minimum of Δ (though an insignificant one). Let $k > i$ be the first time step in which $\Delta(k) \leq \Delta(i)$. There is no point in migrating from S_1 to S_2 at time step i and returning by time step k, because the commute cost C is at least as large as any saving that can be achieved in the inefficiency cost (because Δ never exceeds $\Delta(i) + C$ between time steps i and k). Likewise, there is no point in migrating at time step i and not returning to S_1 by time step k, because then the inefficiency cost would not be larger if the migration is postponed until time step k. Hence we may assume that the optimal solution does not migrate at all at time step i.

The proof regarding migration from S_2 to S_1 is analogous to the proof regarding migration from S_1 to S_2, and is omitted. □

4 An online algorithm

Having characterized the optimal offline algorithm, our online algorithm has a very intuitive interpretation. It stays in its current state until the first time step in which it is certain that the optimal offline algorithm must have migrated (perhaps several time steps earlier) from this state. At that point the online algorithm also migrates. As such, its migration cost is identical

to that of the optimal solution, but it suffers a larger inefficiency cost, due to the lag in migration time compared to the optimal algorithm. We now provide more details.

The **work function algorithm** *WFA*.

- Initialization. Start at state S_1 and set the value of the counter f to $f(0) = 0$.
- At every time step i, upon arrival of a request (x_i, y_i) with $\delta_i = x_i - y_i$, do the following.
  - If at state S_1, let $f(i) = \max[0, f(i-1) + \delta_i]$. If $f(i) \leq C$, serve the request at state S_1. If $f(i) > C$, migrate to state S_2, serve the request there, and change $f(i)$ to $f(i) = 0$.
  - If at state S_2, let $f(i) = \max[0, f(i-1) - \delta_i]$. If $f(i) \leq C$, serve the request at state S_2. If $f(i) > C$, migrate to state S_1, serve the request there, and change $f(i)$ to $f(i) = 0$.

For those familiar with the "standard" version of the work function algorithm (as described in [4], for example), let us remark that it is not hard to show that our version of the algorithm is almost equivalent to the standard version. Our version is simplified compared to the standard version in the sense that instead of keeping two separate counters, one for each state, it keeps only one counter f, which (up to some fixed offset) is equal to the difference between the two counters. The only difference between our version of the algorithm and the standard version is in the behavior when $f(i) = C$ – in this case the standard version would migrate whereas our version would not.

Theorem 4.1 *Let σ be an arbitrary request sequence on which the optimal solution starts at state S_1. Recall that $OPT(\sigma) = F(\sigma) + OPT_I(\sigma) + OPT_M(\sigma)$. Then the total cost of the work function algorithm is at most $WFA(\sigma) \leq OPT(\sigma) + 2OPT_M(\sigma) + C$.*

Proof: Both *OPT* and *WFA* pay the same fixed cost. As we have seen, both pay exactly the same migration cost, because the migrations of *WFA* follow those of *OPT* (with a delay). Hence the only difference is due to inefficiency cost. This difference is concentrated on those intervals in which *OPT* has migrated and *WFA* has not yet migrated. In each such interval, the difference in inefficiencies between the two algorithms is at most C (because *WFA* migrates the moment that f exceeds C, but pays

for the request that caused f to exceed C in the new state, and hence does not incur an inefficiency cost on this request). For every two consecutive intervals, *WFA* pays at most $2C$ more inefficiency cost compared to OPT, but both pay also a migration cost of C. This gives the term $2OPT_M(\sigma)$ in Theorem 4.1. The extra term of C comes from the possibility that the total number of migrations is odd, and OPT finishes at state S_2. (Note that *WFA* will finish at the same state as OPT, due to our convention regarding extending the sequence σ.) $\square$

Observe that in the statement of Theorem 4.1 we assumed that both OPT and *WFA* start processing σ at the same state S_1. This assumption can be removed by adding a term of $O(C)$ to the cost of *WFA*. The initial conditions (starting state and initial value $f(0)$) affect the cost $WFA(\sigma)$ only by an additive term of $O(C)$. This follows from the fact that after the first significant local extremum (minimum or maximum), at the point where Δ already changed by more than C compared to its value at the local extremum, the state *WFA* is at and the value of f (which necessarily is 0 at this point) become independent of the initial conditions.

Corollary 4.2 *The strong competitive ratio ρ_M of the work function algorithm is* 3.

Proof: The upper bound of 3 follows from Definition 2.1, Theorem 4.1, and the discussion that follows the theorem. It is easy to construct examples showing that the strong competitive ratio of *WFA* is no better than 3. $\square$

It is known that for every deterministic algorithm for the two state task system, there is a request sequence in which the strong competitive ratio (and even the overhead competitive ratio) is not significantly better than 3. For completeness, we explain how such a request sequence can be generated. The sequence is generated in an inductive fashion. Having generated a sequence of length t, simulate A on this sequence and observe the state that A ends up in. Choose request $t+1$ as a request that costs slightly more at A's current state than in the other state. On such a sequence, the overhead cost of A is roughly (up to an additive term of $O(C)$) equal to the sum of overhead costs of the following three algorithms: always stay at S_1, always stay at S_2, always migrate if the current request costs less at the other state. Hence one of these algorithms has overhead cost roughly a third of that of A.

5 Extensions

Corollary 4.2 extends to a *window based* competitive ratio. That is, within every time window, algorithm *WFA* is 3-strongly competitive with the optimal solution on this particular time window (regardless of whether this optimal solution is part of an optimal solution on the whole request sequence).

It is easy to extend the work function algorithm to cases where lookahead is available. If the algorithm can see some (limited number) of future requests, it can use this information to shorten the delay between the local extremum and the time that the algorithm migrates, and by this reduce the inefficiency costs. In more details, at time i, with lookahead up to time k, the algorithm can compute ahead of time the value of f for all time steps between i and k. If at any of these time steps (say, time step j), the value of f is above C, the algorithm need not wait until time step j for migrating. Instead, it migrates at a time step $i \leq \ell \leq j$ in which $f(\ell)$ is smallest (breaking ties in favor of later time steps).

It is relatively easy to upper bound the competitive ratio of *WFA* on particular sequences of requests and on some natural distributions. For example, assume that σ is random sequence in which each δ_i independently is equally likely to be either ± 1. Then it is not hard to see that in expectation, $OPT_M(\sigma) = O(OPT_I(\sigma)/C)$, implying that the overhead competitive ratio of *WFA* is $1 + O(1/C)$, which converges to 1 as the commute cost C grows. If the sequence of request is biased (e.g., δ_i is more likely to be $+1$ than -1), the overhead competitive ratio converges to 1 exponentially fast as C grows.

6 Simulations

In general, we believe that our analysis presents an easy methodology for characterizing the performance of *WFA* both in worst case instances and in other instances. Nevertheless, we present some simulation results. They are presented for two reasons. One reason is that even though we understand well the algorithm, our understanding of properties of typical request sequences is not as good, and it may be informative to see at least one such sequence. Hence, we simulated our algorithm on a sequence of requests generated by the application that motivated this work. The other reason to present these experimental results is that they illustrate some interesting features of our algorithm.

In our simulation, the two states of the system corresponded to two data structures – one (that we call R) in which read operations are more efficient, and one (that we call W) in which write operations are more efficient. In state R the read cost is 24 and the write cost is 763. In state W the read cost is 120 and the write cost is 48. The cost to migrate from R to W is 1200, and the cost to migrate from W to R is 800. These costs correspond to Micro Joule (μJ) energy units.

The setting used in order to generate the sequence of requests is as follows. It was generated by inserting $30,000$ items into a B^+-tree (that has up to 64 items per node). Each insertion involves a sequence of read and write operation on various nodes of the tree. The items to insert were temperature readings collected from a set of sensors. These readings were given with very high precision, so all items were distinct. Given the complete B^+-tree, we sampled at random roughly half its nodes. From each such node we extracted the list of read/write operations that occurred at that node. Finally, we concatenated all these lists to generate one sequence of roughly $50,000$ read/write requests.

We compared the performance of six different algorithms on this sequence of requests.

- OPT. This is the the optimal offline algorithm.
- *WFA*. This is our online algorithm.
- R. This algorithm always stays at state R.
- W. This algorithm always stays at state W.
- $MIGRATE$. This algorithm always serves a request at the state in which it is cheaper to serve it, ignoring migration costs. Hence it migrates very often.
- $LOCAL$. This algorithm computes only the costs of serving a request at its current state (ignoring the cost at the other state), and migrates (and resets the cost to 0) once the accumulated cost exceeds the commute cost. The worst case competitive ratio of this algorithm is 3 (similar to *WFA*), and this algorithm is included so as to illustrate the point that algorithms with similar worst case competitive ratios may have very different typical performance, supporting the need for more refined measures (such as the strong competitive ratio used in this paper).

The results appear in figure 1. As predicted by the analysis, the migration cost of *WFA* is essentially the same as that of OPT. The small difference (in the 4th significant digit) is due to the fact that OPT was allowed to start from the optimal starting state, whereas *WFA* started from an arbitrary state. The analysis predicts that the inefficiency cost of *WFA* will be equal to that of OPT plus twice the migration cost. However, it turned out to be even smaller. The reason for this is the fact that costs had values that are not infinitesimally small, leading to situations where *WFA* could migrate at time steps i where $f(i-1)$ was strictly smaller than C, cutting down on the inefficiency cost. The overhead cost of W turned out to be much better than that of R, due to the asymmetric nature of the cost structure for read and write operations. In fact, W performed not much worse than OPT, and hence the fact that *WFA* outperformed W is very encouraging. In comparison, this would not have happened if the sequence of requests was generated independently at random, because for such sequences either R or W is the optimal online algorithm (in terms of expected total cost). As predicted, *WFA* performed better than $LOCAL$ (even though both have the same worst case competitive ratio), but it is interesting to note that $LOCAL$ performed better than R. The online algorithm $MIGRATE$ performed much worse than all other algorithms, which shows that the request sequence was not simply composed of alternating long blocks of reads and of writes.

	Fix	**Inefficiency**	**Migration**
OPT	4.988	3.86	0.2322
WFA	4.988	4.25	0.2329
R	4.988	24.72	0
W	4.988	6.68	0
MIGRATE	4.988	0	52.6217
LOCAL	4.988	7.75	6.7224

Figure 1: Table of the various cost each algorithms

References

[1] Susanne Albers and Hisashi Koga. Page migration with limited local memory capacity. In *Workshop on Algorithms and Data Structures*, pages 147–158, 1995.

[2] Yair Bartal, Moses Charikar, and Piotr Indyk. On page migration and other relaxed task systems. *Theoretical Computer Science*, 268(1):43–66, 2001.

[3] D. Black and D. Sleator. "Competitive algorithms for replication and migration problems." *Technical report CMU-CS-89-201*, Department of Computer Science, Carnegie-Mellon University, 1989.

[4] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[5] A. Borodin, N. Linial and M. Saks. "An optimal on-line algorithm for metrical task system." *JACM 39(4)*, 745–763, 1992.

[6] Chrobak, Larmore, Reingold, and Westbrook. Page migration algorithms using work functions. In *ISAAC: 4th International Symposium on Algorithms and Computation*, 1993.

[7] Jeffery Westbrook. Randomized algorithms for multiprocessor page migration. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 7, pages 135–150, 1992.