Technical Report no. 2005:18

Competitive Freshness Algorithms for Wait-free Data Objects

Peter Damaschke, Phuong Hoai Ha, Philippas Tsigas

CHALMERS | GÖTEBORG UNIVERSITY

Department of Computing Science and Engineering
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden

Göteborg, 2005

Department of Computing Science and Engineering
Division of Computing Science
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden

Technical Report no. 2005:18
ISSN: 1652-926X

Göteborg, Sweden, October 2005.

Abstract

Wait-free concurrent data objects are widely used in multiprocessor systems and real-time systems. Their popularity results from the fact that they avoid locking and that concurrent operations on such data objects are guaranteed to finish in a bounded number of steps regardless of the other operations interference. The data objects allow high access parallelism and guarantee correctness of the concurrent access with respect to its semantics. In such a highly-concurrent environment, where write-operations update the object state concurrently with read-operations, the age/freshness of the state returned by a read-operation is a significant measure of the object quality.

In this paper, we first propose a freshness measure for wait-free concurrent data objects. Subsequently, we model the freshness problem as an online problem and present two algorithms for the problem. The first one is an optimal deterministic algorithm with freshness competitive ratio $\sqrt{\alpha}$, where α is a function of execution-time upper-bound of wait-free operations. The second one is a competitive randomized algorithm with freshness competitive ratio $\frac{\ln \alpha}{1+\ln 2-\frac{2}{\sqrt{\alpha}}}$.

Keywords: wait-free synchronization, online algorithms, concurrent data structures.

1 Introduction

Freshness is a significant property for shared data in general and achieves great concerns in databases [9, 14, 2] as well as in caching systems [13, 10, 12]. Briefly, freshness is a yardstick to evaluate how fresh/new a value of shared data returned by a read-operation is, where the shared data is updated and read concurrently. Read-operations on shared data are always expected to return the newest/freshest values of the data.

Although concurrent data objects play a significant role in multiprocessor systems, they create challenges on consistency. In concurrent environments like multiprocessor systems, consistency of a shared data object is guaranteed mostly by mutual exclusion, a form of locking. However, mutual exclusion degrades the system's overall performance due to lock convoying, i.e. other concurrent operations cannot make any progress while the access to the shared object is blocked. Mutual exclusion also contains risks of deadlock and priority inversion. To address these problems, researchers have proposed *non-blocking algorithms* for shared data objects. Non-blocking methods do not involve mutual exclusion, and therefore do not suffer the problems that blocking can cause. Non-blocking algorithms are either lock-free or wait-free. *Lock-free* algorithms guarantee that regardless of both the contention caused by concurrent operations and the interleaving of their sub-operations, always at least one operation will progress. However, there is a risk for starvation as progress of other operations could cause one specific operation to never finish. *Wait-free* [5] algorithms are lock-free and moreover they avoid starvation. In a wait-free algorithm every operation is guaranteed to finish in a limited number of steps, regardless of actions of other concurrent operations. Non-blocking algorithms have been shown to be of big practical importance [16, 17], and recently NOBLE, which is a non-blocking inter-process communication library, has been introduced [15].

However, to the best of our knowledge there has been no research on the freshness of non-blocking concurrent data objects in multiprocessor systems. For the non-blocking objects, although read-operations are allowed to return any value written by other concurrent operations, they are preferred to return the freshest/latest one of these valid values. Simpson [8, 7] suggested a freshness specification for a single-writer-to-single-reader asynchronous communication mechanism, which is different from atomic register suggested by Lamport [11]. Simpson's communication model with a single writer and a single reader is not suitable for fully concurrent shared objects where many readers and many writers can concurrently access the objects.

In this paper, we define and attack the freshness problem for wait-free shared objects. We model the problem as an online problem and then present two algorithms for it. The first one is an optimal deterministic algorithm, which was inspired by an online search algorithm called *reservation price policy* [4]. The algorithm achieves a competitive ratio $\sqrt{\alpha}$, where α is a function of execution-time upper-bound of wait-free operations. The second is a competitive randomized algorithm with competitive ratio $\frac{\ln \alpha}{1+\ln 2-\frac{2}{\sqrt{\alpha}}}$. The randomized algorithm is nearly optimal since it results from an elaboration on the EXPO search algorithm [4] that $O(\ln \alpha)$ is an asymptotically optimal competitive ratio [3].

The rest of this paper is organized as follows. Section 2 briefly introduces the concept of competitive ratio, which will be used throughout the paper. Section 3 describes the freshness problem and models it as an online problem. Section 4 presents the optimal deterministic algorithm for the freshness problem. Section 5 presents the randomized algorithm. Finally, Section 6 concludes the paper.

2 Preliminaries

In this section, we give a brief introduction to the competitive ratio of online algorithms that will appear frequently in this paper.

Online problems are optimization problems, where the input is received online and the output is produced online so that the cost of processing the input is minimum or the outcome is best. If we know the whole input in advance, we may find an *optimal offline algorithm* OPT processing the whole input with the minimum cost. In order to evaluate how good an online algorithm is, the concept of *competitive ratio* is suggested.

Competitive ratio : An online algorithm ALG is considered competitive with a competitive ratio c (or *c-competitive*) if there exists a constant β such that for any finite input I [1]:

$$ALG(I) \leq c \cdot OPT(I) + \beta \tag{1}$$

where $ALG(I)$ and $OPT(I)$ are the costs of the online algorithm ALG and the optimal offline algorithm OPT to service input I, respectively. The competitive ratio is a well-established concept and the comparison with the optimal off-line algorithm is natural in scenarios where either absolute performance measures are meaningless or assumption on known probability distributions of some inputs is not feasible.

A popular way to analyze an online algorithm is to consider a game between an *online player* and a malicious *adversary*. In this game, i) the online player applies the online algorithm on the input generated by the adversary and ii) the adversary with the knowledge of the online algorithm tries to generate the worst possible input for the player. The input processing costs are very expensive for the online algorithm but still inexpensive for the optimal offline algorithm.

Adversary : For deterministic online algorithms, the adversary with knowledge of the online algorithms can generate the worst possible input to maximize the competitive ratio. However, the adversary cannot do that if the online player uses randomized algorithms. In randomized algorithms, depending on whether the adversary can observe the output from the online player to construct the next input, we classify the adversary into different categories. The adversary that constructs the whole input sequence in advance regardless of the output produced by the online player is called *oblivious adversary*. A randomized online algorithm is *c-competitive* to an oblivious adversary if

$$E[ALG(I)] \leq c \cdot OPT(I) + \beta \tag{2}$$

where $E[ALG(I)]$ is the expected cost of the randomized online algorithm ALG on the input I.

The competitive analysis that uses the competitive ratio as a yardstick to evaluate algorithms is a valuable approach to resolve the problems where i) if we had some information about the future, we could have found an optimal solution, and ii) it is impossible to obtain that kind of information.

3 Problem and Model

Linearizability [6] is the correctness condition for concurrent objects. It requires that operations on the objects appear to take effect atomically at a point of time in their execution interval. This allows a read operation to return any of values written by concurrent write operations, which is illustrated by Figure 1.

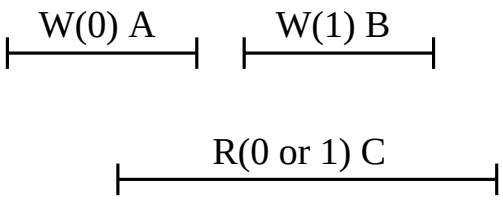

Figure 1. Concurrent reading and writing

We use "W(x) A" ("R(x) A") to stand for a write (read) operation of value x to (from) a shared register by process A. It is correct for C to return either 0 or 1 with respect to linearizability. However, from freshness point of view we prefer C

to return 1, the newer/fresher value of the register. The freshness problem is to find a solution for read operations to obtain the freshest value from a shared object. Intuitively, if a read operation lengthens its execution interval by putting some delay between the invocation and the response, it can obtain a fresher value but it will respond slower from application point of view. Therefore, the freshness problem is to design read-operations that both respond fast and return fresh values.

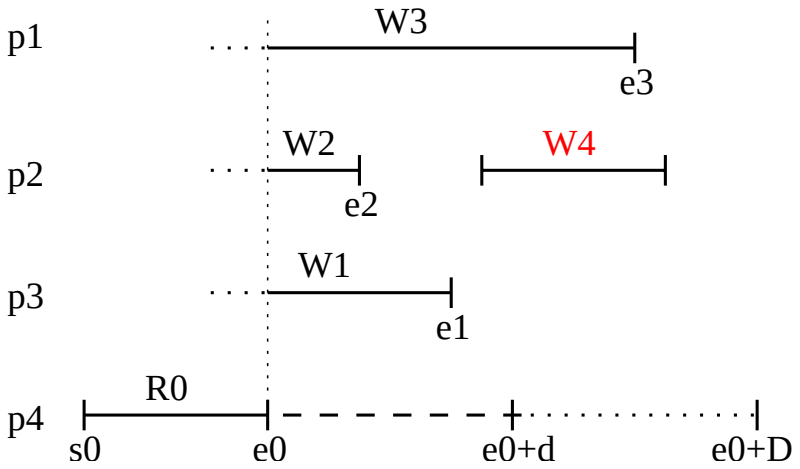

Figure 2. Freshness problem

The freshness problem is illustrated by Figure 2. In the illustration, a read operation R_0 occurs concurrently to three write operations W_1, W_2 and W_3 on a shared object. In this paper, read/write operations imply operations on the same object. The actual execution interval of a operation i is defined from the time s_i the operation starts to the time e_i it takes effect. A time axis runs from left to right. The value returned by R_0 becomes fresher if there are more endpoints e_i appear in the interval $[s_0, e_0]$. In the illustration, if R_0 delays the time-point e_0 to $e'_0 = e_0 + d$, the execution interval $[s_0, e'_0]$ includes two more endpoints e_1 and e_2 and thus the value returned is newer. This implies that R_0 needs to find the time delay d so as to maximize the freshness value $f_d = \frac{k(\#we_d)}{h(d)}$, where $\#we_d$ is the number of new write-endpoints earned by delaying R_0's read-endpoint an interval d and k, h are increasing functions that depend on real applications. Each application may specify its own functions k and h according to its purpose.

Assume that the shared object supports a function for read operations to check how many write operations (with their timestamp) are ongoing at a time[1]. A write-timestamp wt shows the *start-point* of the corresponding write operation whereas a read-timestamp rt shows the *endpoint* of the corresponding read operation. The timestamp objective is to help R_0 ignore W_4 due to $rt_0 < wt_4$. Note that R_0 only needs to consider write-endpoints of write operations that occur concurrently to R_0 in its original execution interval $[s_0, e_0]$, e.g. R_0 will ignore W_4. Therefore, in the freshness problem, the number of concurrent write operations that have not finished at the original read-endpoint e_0 is known and is called M. This number is also the total number of considered write-endpoints, i.e. $\#we \leq M$.

The most challenging issue in the freshness problem is that the endpoints of concurrent write operations appear unpredictably. In order to analyze the problem, we consider it as an online game between a player and an adversary where the malicious adversary decides when to place the write-endpoints e_i on-the-fly and the player (the read operation) decides when she should stop and place her read-endpoint e'_0. The online game starts at the original read-endpoint e_0 and the player knows the total number of write-endpoints M that the adversary will use throughout the game. At a time t, the player knows how many of M endpoints have been used by the adversary so far, i.e. $\#we_t$, (by comparing M with the number of ongoing write operations that ran concurrently with the original read operation) and computes the current freshness value $f_t = \frac{k(\#we_t)}{h(t)}$. For each f_t observed, without knowledge of how the value will vary in the future, the player must decide whether she accepts this value and stops or waits for a better one. In this online game, the player's goal is to minimize the competitive ratio $c = \frac{f_{max}}{f_{chosen}}$, where f_{chosen} is the freshness value chosen by the player and f_{max} is the best value in this game, which is chosen by the adversary. The duration of this game D is the upper bound of execution time of the wait-free read/write operations and is known to the player. This implies that all the M write-endpoints must appear at a time-point in the interval,

[1]This can be done by adding a list of timestamps of ongoing write operations to the shared object. The timestamp can be achieved via an atomic-increment counter.

i.e. $\#we_D = M$.

In summary, we define the freshness problem as follows. Let M be the number of ongoing wait-free write operations at the original read-endpoint e_0 of a wait-free read operation and D be the execution-time upper-bound of these wait-free read/write operations. The read operation needs to find a delay $d \leq D$ for its new endpoint e'_0 so as to achieve an optimal freshness value $f_d = \frac{k(\#we_d)}{h(d)}$, where $\#we_d$ is the number of write-endpoints earned by the delay d and k, h are increasing functions that depend on real applications. We assume the time is discrete, where a time unit is the period with which the read operation regularly checks the number of ongoing write operations on the shared object. The extended read operation is still wait-free with an execution-time upper-bound $2D$.

The rest of this paper presents two competitive online algorithms for the freshness problem. The first one is an optimal deterministic algorithm with competitive ratio $\sqrt{\alpha}$, where $\alpha = \frac{h(D)}{h(1)}$. The second one is a nearly-optimal randomized algorithm with competitive ratio $\frac{\ln \alpha}{1+\ln 2-\frac{2}{\sqrt{\alpha}}}$. Note that the competitive ratios do not depend on k and M, the parameters related to the number of endpoints.

4 Optimal Deterministic Algorithm

Modeling the freshness problem as an online game, we observe that the freshness problem is a variant of the online search problem [4]. In the online search problem, a player searches for the maximum (minimum) price in a sequence of prices that unfolds daily. For each day i, the player observes a price p_i and she must decide whether to accept this price or to wait for a better one. The search game ends when the player accepts a price, which is also the result.

In the freshness problem, in addition to the fact that the player is searching for the best in a sequence of freshness values that unfolds sequentially in a foreknown range, there are more restrictions on how the adversary can vary the freshness value f_t at a time t:

$$\frac{f_{t-1} * h(t-1)}{h(t)} = \frac{k(\#we_{t-1})}{h(t)} \leq f_t = \frac{k(\#we_t)}{h(t)} \leq \frac{k(M)}{h(t)} \tag{3}$$

The restrictions come from the fact that the adversary cannot remove the endpoints she has placed, i.e. $\#we_{t-1} \leq \#we_t \leq M$, where $\#we_t$ is the number of endpoints that have appeared until a time t, and the freshness value at the time t is $f_t = \frac{k(\#we_t)}{h(t)}$, where k, h are increasing functions. Since $1 \leq t \leq D$, from Equation (3) it follows $f_t \leq \frac{k(M)}{h(1)}$. On the other hand, since M ongoing write-operations must end at time-points in the interval D, the player is ensured a freshness value $f_{min} = \frac{k(M)}{h(D)}$ by just waiting until $t = D$. Therefore, the player considers to stop at a freshness value f_t only if $f_t \geq \frac{k(M)}{h(D)}$. We have $\frac{k(M)}{h(D)} \leq f_t \leq \frac{k(M)}{h(1)}$.

Inspired by an online search algorithm called *reservation price policy* [4], we suggest a competitive deterministic algorithm for the freshness problem. Then, we prove that the algorithm is optimal, i.e., no deterministic algorithm can achieve a better competitive ratio.

Deterministic Algorithm: The read operation accepts the first freshness value that is not smaller than $f^* = \frac{k(M)}{\sqrt{h(1)h(D)}}$.

Indeed, let f^* be the threshold for accepting a freshness value and f_{max} be the highest value chosen by the adversary. The player (the read operation) waits for a value $f_t \geq f^*$. If such a value appears in the interval D, the player accepts it and returns it as the result. Otherwise, when waiting until the time D, the player must accept the value $f_{min} = \frac{k(M)}{h(D)}$.

Case 1: If the player chooses a big value as f^*, the adversary will choose $f_{max} < f^*$, causing the player to wait until the time D and accept the value $f_{min} = \frac{k(M)}{h(D)}$. The competitive ratio in this case is $c_1 = \frac{f_{max}}{\frac{k(M)}{h(D)}} < \frac{f^*}{\frac{k(M)}{h(D)}}$.

Case 2: If the player chooses a small value as f^*, the adversary will place f^* at a time t, causing the player to accept the value and stop. Right after that, the adversary places all M endpoints, achieving a value $f_{max} = \frac{k(M)}{h(t)} \leq \frac{k(M)}{h(1)}$ (equality occurs when the adversary chooses $t = 1$). The competitive ratio in this case is $c_2 = \frac{\frac{k(M)}{h(1)}}{f^*}$.

In order not to be fooled by the adversary, the player should choose f^* so as to make $c_1 = c_2$, which results in $f^* = \frac{k(M)}{\sqrt{h(1)h(D)}}$ and the competitive ratio $c = c_1 = c_2 = \sqrt{\frac{h(D)}{h(1)}}$.

Let $\alpha = \frac{h(D)}{h(1)}$. This leads to the following corollary.

Corollary 4.1. *The suggested deterministic algorithm is competitive with competitive ratio* $c = \sqrt{\alpha}$*, where* $\alpha = \frac{h(D)}{h(1)}$.

We now prove that there is no deterministic algorithm for the freshness problem that achieves a better competitive ratio than $\sqrt{\alpha}$.

For convenience, we consider the game on a real axis for the logarithm of freshness values, abbreviated LF in the following. Let t be the time, initially $t = 1$. We normalize the LF axis in such a way that freshness $\frac{k(M)}{h(D)}$ corresponds to point 0 and freshness $\frac{k(M)}{h(1)}$ to point $\ln \frac{h(D)}{h(1)}$ (or $\ln \alpha$). That is, going one unit upwards on the LF axis increases the freshness by factor e (Euler's number). (All this is done for convenience only. Scaling factors do not affect the competitive ratios.)

We also introduce some parameters that characterize the status of a game. At any moment, let f be the maximum LF the adversary has already reached during the history of the game. But initially we set $f = 0$, since $\frac{k(M)}{h(D)}$ is the guaranteed freshness of the adversary. That means $f \geq 0$ at any time. Let g be the maximum LF the adversary can still achieve at a given time. (This corresponds to freshness $k(M)/h(t)$ at time t unless f is already larger, in which case we have $g = f$. However in the latter case the game is over, without loss of generality: The adversary cannot gain more and would therefore decrease freshness as quickly as possible to make the player's position worse, hence the player should stop now. The dotted polyline in Figure 3 illustrates the case $f = g(t)$ in which the player should stop at t.) It is also convenient to consider the time-function $h(t)$ on a logarithmic axis too, where $h(1)$ corresponds to point 0 and $h(D)$ corresponds to point $\ln \frac{h(D)}{h(1)}$ $(= \ln \alpha)$. Note that, on the logarithmic time axis, g decreases at unit speed, starting at point $\ln \alpha$. Finally, let c denote the current LF. We remark that c can decrease at most at unit speed but can jump upwards arbitrarily as long as $c \leq g$.

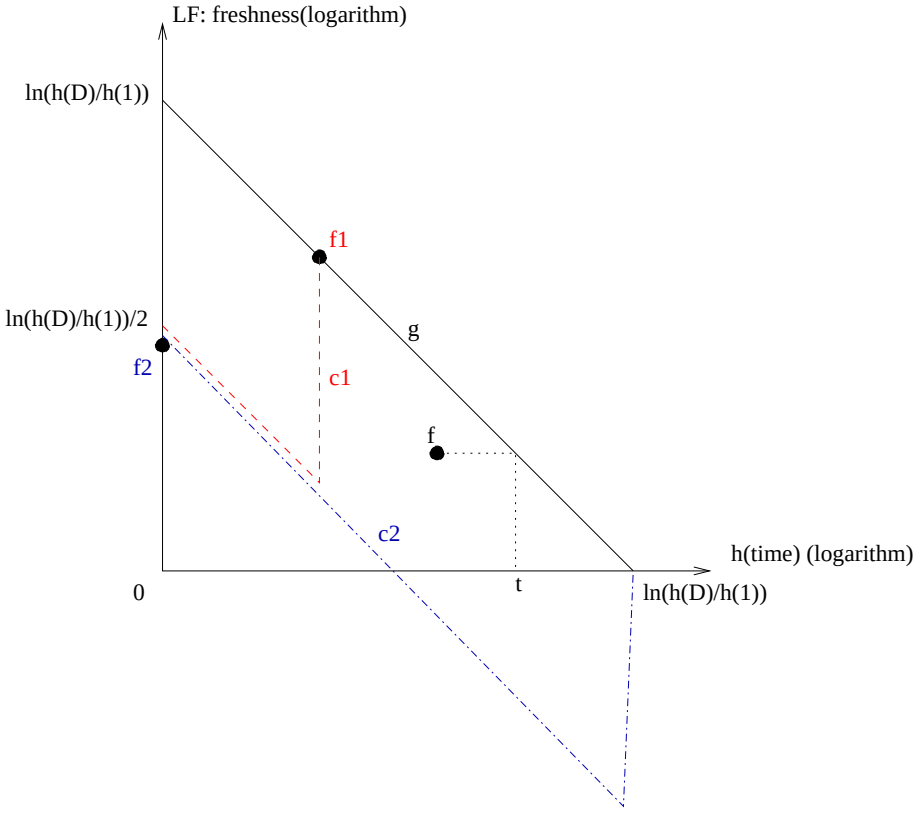

Figure 3. Illustration for the proof of Theorem 4.1

Theorem 4.1. *The optimal deterministic competitive ratio is asymptotically (subject to lower-order terms)* $\sqrt{\alpha}$*, where* $\alpha = \frac{h(D)}{h(1)}$.

Proof. To prove the theorem, we only need to show one of the adversary's strategies against which no online deterministic algorithm can achieve a competitive ratio better than $\sqrt{\alpha}$.

The adversary starts with $c = \frac{\ln \alpha}{2} = \frac{\ln \frac{h(D)}{h(1)}}{2}$. Then she decreases c at unit speed until the player stops. Immediately after this moment, c jumps to g if $c > 0$ at stop time (1), otherwise c keeps on decreasing at unit speed (2). Clearly, we have constantly $g - c = \frac{\ln \alpha}{2}$ until the stop. Let p be the player's LF. In case (1) we finally have $f = g$, hence $f - p = g - c = \frac{\ln \alpha}{2}$ (cf. the dashed polyline c_1 in Figure 3). In case (2), f has still its initial value $\frac{\ln \alpha}{2}$ whereas $p \leq 0$, hence $f - p \geq \frac{\ln \alpha}{2}$ (cf. the line c_2 in Figure 3). Thus the competitive ratio is at least $e^{\frac{\ln \alpha}{2}} = \sqrt{\alpha}$. The player can achieve this competitive ratio by applying the deterministic algorithm given above. □

This result shows that a deterministic player cannot take advantage of the constraints on the behavior of freshness in time (compared to online search on unrestricted sequences of profit values).

5 Competitive Randomized Algorithm

In this section, we present a competitive randomized algorithm for the freshness problem. The algorithm achieves a competitive ratio $c = \frac{\ln \alpha}{1 + \ln 2 - \frac{2}{\sqrt{\alpha}}}$, where $\alpha = \frac{h(D)}{h(1)}$.

As discussed in the previous section, our problem is a restricted case of online search. We model the problem by a game between an (online) player and an adversary. The adversary's profit is the highest freshness ever reached. The player's profit is the freshness value at the moment when she stops. Note that for a player running a randomized strategy, the profit is the expected freshness value, with respect to the distribution of stops resulting from the strategy and input. We shall make use of a known simple transformation of (randomized) online search to (deterministic) one-way trading [4]: The player has some budget of money she wants to exchange while the exchange rates may vary over time. Her goal is to maximize her gain. The transformation is given as follows: The budget corresponds to probability 1, and exchanging some fraction of money means to stop the game with exactly that probability. Note that a deterministic algorithm for online search has to exchange all money at *one* point in time. The transformation was inspired by a well-known competitive randomized algorithm EXPO [4]. Applying the EXPO algorithm on the freshness problem achieves a competitive ratio $\ell \frac{2^{\ell - 1 + 1/\ln 2}}{2^{\ell - 1 + 1/\ln 2} - \frac{1}{\ln 2}}$, where $\ell = \log_2 \alpha$. That means for the freshness problem our randomized algorithm is better than the EXPO algorithm by a constant factor $\frac{1 + \ln 2}{\ln 2}$ when α becomes large.

Theorem 5.1. *There is a randomized algorithm for the freshness problem with competitive ratio* $\frac{\ln \alpha}{1 + \ln 2 - \frac{2}{\sqrt{\alpha}}}$, *where* $\alpha = \frac{h(D)}{h(1)}$

Proof. We start with some conventions. We imagine that the money, both exchanged and non-exchanged, is "distributed" on the LF axis. Formally, the allocation of money on the LF axis at any time is described by two non-negative real *density functions* S and T, where $S(x)$ is the density of not yet exchanged money in point x of the LF axis, $T(x)$ is similarly defined for the money that has been already exchanged. What functions S and T specifically are, and how they are modified by the opponents' actions, will be described below. Let the total amount of money be $\ln \alpha$ by convention. (Recall that scaling factors do not influence the competitive ratio.)

The *value* of every piece of *exchanged* money is the freshness value of its position on the LF axis. Note that the total value of exchanged money defined in this way, i.e. the integral over the value-by-density product, is the player's profit in the game. Moreover, the player can temporarily have some of the money in her *pocket*.

The idea of the strategy is to guarantee some concentration of exchanged money immediately below the final f, either some constant minimum density of T or, even better, a constant amount at one point not too far from f. We want to keep T simple in order to make the calculations simple. (The well-known δ_x symbol used below denotes the distribution with infinite density at a single point x but with integral 1 on any interval that contains x. We also use the same notations f, g, c as earlier.) Locating much money instantaneously is risky because c may jump upwards, and then this money has little value compared to the adversary's. On the other hand, since c decreases at most with unit speed, the player may completely abstain from exchanging money as long as c is increasing, and wait until c goes down again. These preliminary thoughts lead to the following strategy.

In the beginning, let the not-yet-exchanged money be located on the LF axis on interval $[0, \ln \alpha]$ with density 1, that is, we have $S = 1$ on this interval. Remember that g decreases at unit speed. The player puts the money above g in her pocket. Whenever f increases, she also puts the money below the new f in her pocket. Hence we always have $S = 1$ on $[f, g]$, and $S = 0$ outside. The player continuously locates exchanged money on the LF axis, observing the following rule: *If you have money in your pocket and c is positive and decreasing, and $T(c) < 2$ at the current c, then set $T(c) := 2$. If the game is over*

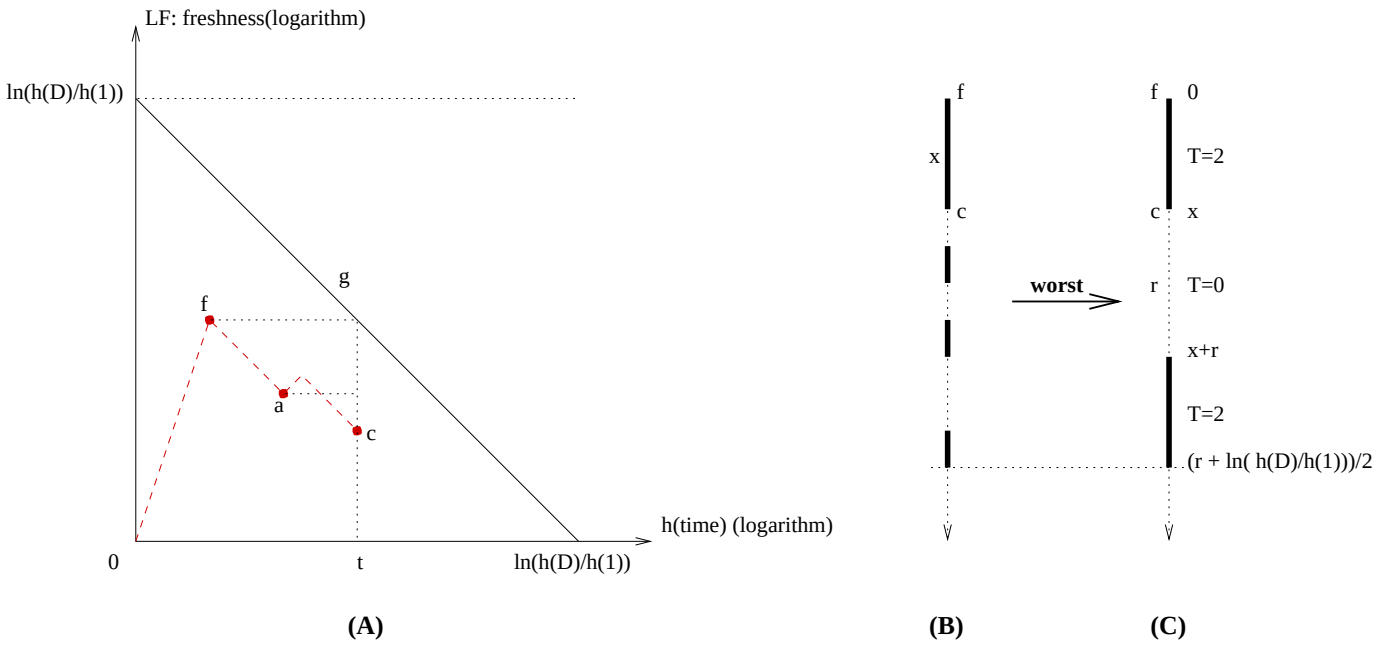

Figure 4. Illustration for the randomized algorithm

(because of $f = g$) and not all money is exchanged yet, put the rest r on the current c. Note that the adversary must set the final c nonnegative.

Filling-up density T to 2 is always possible, by the following argument: The player uses the one unit of money from S that she gets per time unit from the region above the falling g, and the money from S that she got directly from the current points c when f went upwards.

Obviously, the player produces a density function T that is constantly 2 on certain intervals and 0 outside, plus some component $r\delta_c$. We make some crucial observations regarding the final situation: (1) T has density 2 on interval $(c, f]$, or we have $c = f$. (2) The *gaps* with $T = 0$ between the "$T = 2$ intervals" have a total length not exceeding r.

These claims follow easily from the strategy: (1) Either c begins decreasing, starting from the last f, and T is filled up to 2 all the time when $c > 0$, as we saw above, or the final c equals the final f. (2) Whenever f went upwards, the player has taken from S the money corresponding to the increase of f, and later she has transferred it to T and located it at the same points again. Hence, only on intervals not "visited" again by c we have $T = 0$, and the money taken from S on these intervals is still in the player's pocket and thus contributes to r.

Figure 4-(A) illustrates the player's behavior. The dashed line represents a variation of c in a game; point c is the final value of c when the game ends, i.e. $f = g(t)$. For all values v on the LF axis between f and a and between a and c, the player sets $T(v) = 2$.

Using (1),(2) we now analyze the profit the player can guarantee herself. Remember that the value of exchanged money located on the LF axis decreases exponentially. Let $x = f - c$ (final values). Both r and x depend on the input, i.e., the behavior of c in time. The total amount of money is fixed, it equals $\ln \alpha$. For any fixed r, x, the worst case is now that the gaps in T sum up to the maximum length r and are as high as possible on the LF axis, that is, immediately below point c, because in this case all exchanged money outside $[c, f]$ has the least possible value. That is, T has only one gap, namely interval $[c - r, c]$.

Figure 4-(C) illustrates the worst case corresponding to an instance -(B), where solid lines represent ranges on the LF axis with $T = 2$. In the worst case, the adversary shifts all solid lines except for $[c, f]$ to the lowest possible position so as to minimize the player's profit.

Hence, a lower bound on the player's profit, divided by the value at f, is given by

$$\min_{r,x} \left(2 \int_0^x e^{-t} dt + r e^{-x} + 2 \int_{x+r}^{(r+\ln \alpha)/2} e^{-t} dt \right),$$

where we started integration (with $t = 0$) at point f and go down the LF axis (cf. Figure 4-(C)). Verify that, in fact, $\int T dt = \ln \alpha$. The above expression evaluates to

$$2 + (r - 2 + 2e^{-r})e^{-x} - 2e^{-(r+\ln \alpha)/2} > 2 + (r - 2 + 2e^{-r})e^{-x} - 2/\sqrt{\alpha}.$$

For any fixed x, this is minimized if $2e^{-r} = 1$, that is, $r = \ln 2$. Since now $r - 2 + 2e^{-r} = \ln 2 - 2 + 1 < 0$, the worst case is $x = 0$, which gives $1 + \ln 2 - 2/\sqrt{\alpha}$. The adversary earns $\ln \alpha$ times the value at f. □

6 Conclusions

To the best of our knowledge, this paper is the first paper that defines the freshness problem for wait-free data objects. Within this paper, we have modeled the freshness problem as an online problem and then have presented two online algorithms to solve it. The first one is an optimal deterministic algorithm with freshness competitive ratio $\sqrt{\alpha}$, where α is a function of execution-time upper-bound of wait-free operations. The function α is specified by real applications according to their purpose. The second is a randomized algorithm with freshness competitive ratio $\frac{\ln \alpha}{1+\ln 2-2/\sqrt{\alpha}}$. The randomized algorithm is nearly optimal. In [3] it has been showed that $O(\ln \alpha)$ is a lower bound on competitive ratios for the one-way trading with time-varying exchange-rate bounds corresponding to the freshness problem. This gives a lower bound $O(\ln \alpha)$ to competitive ratios of randomized freshness algorithms.

Our proofs worked with a logarithmic transformation that reveals the geometry of this online problem and enabled us to find optimal competitive ratios, subject to constants. This transformation was inspired by the randomized EXPO algorithm [4] for unrestricted online searching.

This paper provides a starting point to further research the freshness problem on concurrent data objects as an online problem. It has presented algorithms that can apply on general wait-free data objects without any restrictions. However, wait-free data objects are just one kind of concurrent data objects while freshness itself is an interesting problem for concurrent data objects in general.

References

[1] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[2] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 117–128, 2000.

[3] P. Damaschke, P. H. Ha, and P. Tsigas. One-way trading with time-varying exchange rate bounds. *Technical report CS:2005-17, Chalmers University of Technology*, 2005.

[4] R. El-Yaniv, A. Fiat, R. M. Karp, and G. Turpin. Optimal search and one-way trading online algorithms. *Algorithmica*, 30(1):101–139, 2001.

[5] M. Herlihy. Wait-free synchronization. *ACM Transaction on Programming and Systems*, 11(1):124–149, Jan. 1991.

[6] M. Herlihy and J. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, July 1990.

[7] H.R.Simpson. Correctness analysis for class of asynchronous communication mechanisms. *Computers and Digital Techniques, IEE Proceedings-*, 139(1):35– 49, Jan., 1992.

[8] H.R.Simpson. Freshness specification for a class of asynchronous communication mechanisms. *Computers and Digital Techniques, IEE Proceedings-*, 151(2):110–118, Mar., 2004.

[9] K.-D. Kang, S. H. Son, and J. A. Stankovic. Managing deadline miss ratio and sensor data freshness in real-time databases. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1200–1216, Oct., 2004.

[10] A. Labrinidis and N. Roussopoulos. Exploring the tradeoff between performance and data freshness in database-driven web servers. *The VLDB Journal*, 13(3):240–255, 2004.

[11] L. Lamport. On interprocess communication. part ii: Algorithms. *Distributed Computing*, 1(2):86–101, 1986.

[12] W.-S. Li, O. Po, W.-P. Hsiung, K. S. Candan, and D. Agrawal. Engineering and hosting adaptive freshness-sensitive web applications on data centers. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 587–598, 2003.

[13] Y. Ling and W. Chen. Measuring cache freshness by additive age. *SIGOPS Oper. Syst. Rev.*, 38(3):12–17, 2004.

[14] E. Pacitti and E. Simon. Update propagation strategies to improve freshness in lazy master replicated databases. *The VLDB Journal*, 8(3-4):305–318, 2000.

[15] H. Sundell and P. Tsigas. NOBLE: A non-blocking inter-process communication library. In *Proceedings of the 6th Workshop on Languages, Compilers and Run-time Systems for Scalable Computers*, LNCS. Springer Verlag, 2002.

[16] P. Tsigas and Y. Zhang. Evaluating the performance of non-blocking synchronization on shared-memory multiprocessors. In *Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 320–321, 2001.

[17] P. Tsigas and Y. Zhang. Integrating non-blocking synchronisation in parallel applications: Performance advantages and methodologies. In *Proceedings of the 3rd ACM Workshop on Software and Performance (WOSP'02)*, pages 55–67, July 2002.