Online Algorithms for Locating Checkpoints

Marshall Bern * Daniel H. Greene * Arvind Raghunathan † Madhu Sudan ‡

Abstract

Motivated by applications in data compression, debugging, and physical simulation, we consider the problem of adaptively choosing locations in a long computation at which to save intermediate results. Such checkpoints allow faster recomputation of arbitrary requested points within the computation. We abstract the problem to a server problem in which k servers move along a line in a single direction, modeling the fact that most computations are not reversible. Since checkpoints may be arbitrarily copied, we allow a server to jump to any location currently occupied by another server. We present online algorithms and analyze their competitiveness. We give lower bounds on the competitiveness of any online algorithm and show that our algorithms achieve these bounds within relatively small factors.

1. Introduction

Suppose you are building software for accessing an encyclopedia. To save space, you store the encyclopedia in compressed form using an adaptive data compressor [8, 14]. Your software must handle requests from users wishing to read arbitrarily-located articles within the encyclopedia. Here a problem arises: in order to decompress a specific article, you must recreate the compression statistics as they were at the time that article was compressed.

There are several possible approaches to this problem. One could save all compression statistics, but this defeats the purpose of compression. One could break the encyclopedia into smaller files– or equivalently restart the compressor occasionally– but this, too, compromises compression. A similar solution is to occasionally save– or checkpoint– the compression statistics while compressing; then a request is handled by finding the closest previous checkpoint and recomputing statistics from that point up to the requested article. The most flexible solution allows the locations of the checkpoints to move and adapt to the pattern of requests. In this paper we investigate adaptive solutions to the problem of locating checkpoints.

Besides data compression our work applies to a number of other contexts.

- In debugging a long program, one typically probes an irreversible computation at various points in order to check intermediate values [10].
- In studying an irreversible physical system, one would like to interactively probe a computer simulation.
- In testing a VLSI design, different members of a design team may work on different parts of a critical path simultaneously. Thus a useful feature of a waveform simulator such as *Spice* would be the capability to answer probes at arbitrary points along a path. As above, the computation of a waveform is typically irreversible.

We model the problem as follows. (Here we use the terminology of the data compression application.) We can afford k "permanent" checkpoints; in addition, we set aside scratch space for one temporary checkpoint. We think of the temporary checkpoint as residing in fast memory so that it can be rapidly updated as we read through the encyclopedia; the other k checkpoints may reside in slow memory.

We are presented with a sequence of n requests, each at a real number in the half-open interval $[0, m)$. A permanent checkpoint may (1) move forward (towards larger numbers) along positions in $[0, m)$, incurring cost equal to the difference between starting and ending positions; (2) *fork*, that is, immediately move at no cost,

*Xerox PARC, 3333 Coyote Hill Rd., Palo Alto, CA, 94304
†Courant Institute, New York University
‡Computer Science Division, UC - Berkeley, supported in part by NSF grant CCR-8947792

to a position currently occupied by another checkpoint; or (3) *restart*, at no cost, at position 0. Any number of these moves may be made in response to a request. After these moves, the request at $r \in [0, m)$ is *serviced* by the temporary checkpoint, incurring fixed cost 1 plus the distance to the request from the closest checkpoint at a position no greater than r. In the terminology of Manasse et al [11], each request is serviced by an *excursion* from the nearest permanent checkpoint. The temporary checkpoint does not persist between requests.

We would like to minimize the total cost of a sequence of requests; that is, we are interested in maximizing throughput rather than minimizing worst-case latency. In our model, the only costs are computation; copying one block of memory to another, as in a fork move, is free.

For simplicity, we carry out our analysis assuming that position m coincides with position 0, that is, the encyclopedia is circular. This assumption eliminates move (3) and clarifies our arguments. We then show how to transfer our results back to the linear case.

We analyze the *competitiveness* of our algorithms [3, 9, 11, 13]. That is, we compare the performance of an online algorithm against the performance of an optimal offline algorithm that sees all requests in advance. An algorithm is called c-competitive if its cost on any sequence of n requests is at most $O(1)$ greater than c times the offline algorithm's cost. This style of analysis refines traditional worst-case analysis. Competitive analysis is worst-case in that no assumptions about the distribution or correlation of requests are made; however, it measures performance relative to what is achievable by an omniscient algorithm, rather than in absolute terms.

A discretized version of our problem is an example of a *task system* as defined by Borodin et al [3]. Borodin et al, however, study a more general model in which the costs of serving requests– rather than just request locations– may be chosen by an adversary, so their bounds have no nontrivial implications for our problem. Other related work includes a number of recent papers on *server problems* [2, 4, 5, 6, 7, 11, 12]. The paper that addresses the problem most similar to our work is by Chrobak et al [5]. This paper presents an optimally (i.e., k-) competitive algorithm for k servers moving on a line. Our problem differs from this problem in several ways: our servers move in only one direction, we include the additive cost of 1 for each request, we allow excursions, and, most crucially, we allow the fork move. To our knowledge, our work introduces the fork move to the server literature. This move is very natural for applications in which servers represent information rather than physical objects.

We obtain the following results, comparing online algorithms against offline algorithms with an equal number of checkpoints:

- For the case of $k = 1$, that is, only one permanent checkpoint, a lower bound of $(m^{1/3}/2)$-competitiveness that applies to any deterministic online algorithm.
- For $k = 1$, a memoryless $4m^{1/3}$-competitive algorithm.
- For any k, a $2(k+1)m^{1/3}$-competitive algorithm when the fork move is disallowed for both online and offline algorithms. (This result generalizes the upper bound for $k = 1$ and separates the difficulties introduced by one-way motion and forking.)
- For $k \geq 3$, a lower bound of $\Omega(m^{1/2})$-competitiveness. (Here Ω notation implies a constant independent of k as well as m.)
- For $k \geq 2$, a memoryless $3(km)^{1/2}$-competitive algorithm.

We also give lower bounds on the competitiveness of an online algorithm compared with an offline algorithm with fewer servers. Finally, we give algorithms for the offline problem. Some proofs are omitted or merely sketched in this preliminary version.

The variable m is the ratio between the total amount of computation and the minimum amount of work to answer a request. In some sense it represents the number of smallest units: articles in an encyclopedia or lines of code in a program to be debugged. For the location of checkpoints to matter, m must be much greater than k; moreover, since the size of intermediate results typically increases as the overall length of the computation increases, k may have to decrease with increasing m. So except for the case of $k = 2$ we have obtained algorithms that match our lower bounds up to relatively small factors.

From a practical point of view, our results are mixed. Although it is encouraging that there exist algorithms more competitive than the (m/k)-competitiveness of static checkpoints, our performance guarantees are quite weak for interesting values of m (say 100,000). Our lower bounds show that no online algorithm can always be satisfying when viewed retrospectively.

Finally, we believe our work introduces an interesting test case for competitive analysis. We are extending this type of analysis to a problem more difficult than those previously considered (caching, list access, online scheduling of elevators and disk drives), as indicated by our strong lower bounds. Can competitive analysis nevertheless lead us to algorithms for locating checkpoints that perform well in practice?

2. Preliminaries

In subsequent sections, we refer to the permanent checkpoints as *servers* and the temporary checkpoint as the *temporary server*. In analyzing competitiveness, we usually refer to the online algorithm under consideration as the *player* and the optimal offline algorithm as the *adversary*.

Player servers and adversary servers (sometimes called simply *players* and *adversaries* when no confusion is possible) both move on a *directed circle* of circumference m, with $m^{1/3} \geq \max\{3, k\}$ where k is the number of player servers. On the directed circle, all servers move only clockwise.

We use interval notation to denote arcs of the directed circle. For example $[x, y)$ with $x < y \leq m$ denotes the clockwise half-open arc from x to y, i.e., the one that does not contain m, while $[y, x)$ denotes the complementary arc. The *distance* $d(s, x)$ is the length of $[s, x]$.

The sequence of request locations is denoted $r_1, r_2, \ldots, r_n$, where each r_i is a real number in $[0, m)$. R_t denotes $r_1, \ldots, r_t$ and $R[s, t]$ denotes $r_s, r_{s+1}, \ldots, r_t$.

By the term *online*, we mean that the player chooses the positions of his servers at time t deterministically as a function of the request history R_t. If in addition, the player uses only the current request r_t and the current positions of his servers to choose his next move, then the algorithm is called *memoryless* [11, 12]. The *offline* algorithm (i.e., the adversary) may use all of R_n to choose server locations at any time t. For an online algorithm P, let $\mathit{Ratio}(P, R_n)$ denote the ratio of the cost incurred by P to the cost incurred by the adversary on request sequence R_n. The *competitiveness* of online algorithm P is then defined to be the "worst ratio" $\limsup_{n\to\infty} \sup_{R_n} \mathit{Ratio}(P, R_n)$.

3. 1-Server Lower Bound

This section gives the first in a series of lower bounds on the competitiveness of deterministic online algorithms as compared to optimal offline algorithms. There appears to be no advantage in allowing the online algorithms to make probabilistic choices based on R_t, but for clarity the proofs are restricted to deterministic algorithms. (See [1] for more on randomization.)

Theorem 1. *An online 1-server algorithm can be no better than $(m^{1/3}/2)$-competitive.*

Proof: We describe an adversary strategy that can be repeated on *epochs* of $n = \lceil m^{1/3} \rceil$ requests. Let the adversary's server be at location z and the history of requests be R_t. The adversary considers extending the history by $n-1$ requests at location $z + m^{1/3}$ followed by one request at either (1) z or (2) $z+m^{1/3}$. Case (2) is used if the player algorithm would keep its server in $(z+m^{1/3}, z]$ for the entire subsequence $R[t+1, t+n]$ (most likely by holding back at z). In this case the adversary moves up to $z+m^{1/3}$, incurring a cost of $m^{1/3}$ and then has n requests at cost 1. By contrast the player has n requests at cost at least $m^{1/3}+1$ each, for a total of at least $n(m^{1/3}+1)$. The ratio $n(m^{1/3}+1)/(n+m^{1/3})$ is at least $m^{1/3}/2$. If the player ever leaves the arc $(z+m^{1/3}, z]$, then case (1) is used; the adversary holds back at z, spending $n(m^{1/3}+1) \leq 2m^{2/3}$ to process the requests, and the player must spend $m - m^{1/3} + n$ because of the empty arc. In either case the player incurs cost at least $m^{1/3}/2$ times the adversary's cost for the epoch. ■

Essentially the same argument works for the case of servers moving on the line rather than the circle. The adversary first moves its server to $m/2$. There follow at least $m^{2/3}/2$ epochs of the form above, as the adversary's server progresses from $m/2$ to m. In each epoch the ratio of the player's cost to the adversary's cost is at least $m^{1/3}/2$. The player's total work before the adversary must reset at $m/2$ is at least $m^{4/3}$, so over an entire "superepoch" the player's ratio can be no better than $m^{1/3}/2 - 1$.

4. 1-Server Upper Bound

In this section we give an algorithm for the movement of a single server. This algorithm will lead us to the more general algorithm for an arbitrary number of servers when both player and adversary are not allowed to fork. We show that the following simple, memoryless algorithm, called Two-Phase, is $4m^{1/3}$-competitive.

for each request r **do**
  (Phase 1) Move the server (at s) towards r by $\max\{0, d(s, r) - m^{2/3}\}$
  (Phase 2) Move the server (now at s') further towards r by $\max\{0, \frac{d(s', r) - m^{1/3}}{m^{1/3}+1}\}$
  Serve r using the temporary server **od**

In order to analyze the performance of this algorithm and later online algorithms, we think of player and adversary working in parallel on $r_1, r_2, \ldots, r_n$.

We define a potential function Φ that is used to "smooth" the induction. The player's work typically reduces Φ by at least the work done, while the adversary's work increases Φ by at most the competitive factor times the amount of work done. These bounds, along with bounds on the initial and final values of Φ, suffice to bound the player's work on sequence R_n by a multiple of the adversary's.

In this section, we define Φ as follows, where s denotes the position of the player's server and x denotes the position of the adversary's server.

$$\Phi_1 = d(s,x) \qquad \Phi_2 = \min\{m, m^{1/3} \cdot d(s,x)\}$$
$$\Phi_3 = \max\{0, 2m^{2/3}(m^{1/3} - d(s,x))\}$$
$$\Phi = \Phi_1 + \Phi_2 + \Phi_3$$

This Φ is a piecewise-linear function of the distance between player and adversary. When adversary and player are further than $m^{2/3}$ apart, Φ decreases with slope 1 as the player approaches the adversary, reflecting the rapid motion of Phase 1. When adversary and player are between $m^{2/3}$ and $m^{1/3}$ apart, Φ decreases with slope about $m^{1/3}$ to compensate for the slower motion of Phase 2. Closer than $m^{1/3}$ apart, Φ increases with slope about $2m^{2/3}$ so that it does not jump discontinuously when the player crosses the adversary or vice versa.

Let s denote (as above) the player's position before Phase 1, s' the player's position after Phase 1 and before Phase 2, and s'' the player's position after Phase 2. Similarly let x and x' be the adversary's starting and final positions. Let W_P denote the total cost incurred by the player in serving r, that is, 1 plus the total permanent and temporary checkpoint motion. Similarly let W_A denote the total cost incurred by the adversary.

For ease of analysis, the actions following the receipt of a request r ($= r_i$ for some i) are conceptually divided into the following steps: (1) the adversary moves arbitrarily; (2) the player executes an iteration of Two-Phase; and, (3) the adversary services the request without moving a permanent checkpoint. The first two lemmas analyze the changes in Φ during steps (1) and (2).

Lemma 1. *When the adversary moves, Φ increases by at most $(m^{1/3}+1) \cdot d(x,x')$.*

Proof: First note that Φ varies continuously when the adversary crosses over the player or when the player crosses over the adversary (i.e., when $d(s,x)$ changes from m to 0 or 0 to m). Next observe that when the adversary moves distance d without crossing the player, Φ_1 increases by at most d, Φ_2 increases by at most $m^{1/3}d$, and Φ_3 does not increase. Thus Φ increases by at most $(m^{1/3}+1) \cdot d(x,x')$ when the adversary moves from x to x'. ■

Lemma 2. *Assume $d(x',r) \le m^{2/3} - m^{1/3}$. Then in Phase 1, Φ decreases by at least $d(s,s')$, and in Phase 2, Φ decreases by at least $(m^{1/3}+1) \cdot d(s',s'') - 2m^{1/3} \cdot d(x',r)$.*

Proof: If $d(s,s') = 0$, then the player does not move in Phase 1, so Φ cannot increase. On the other hand, if $d(s,s') > 0$ then $d(s',r) = m^{2/3}$, and the adversary at x' is at least $m^{1/3}$ ahead of the player. Hence Φ_1 decreases by at least $d(s,s')$ and neither Φ_2 nor Φ_3 increase.

We now analyze the change in Φ during Phase 2. As above, the case that $d(s',s'') = 0$ is trivial. So assume $d(s',s'') > 0$ and consider the case that $x' \in (r,s']$, that is, the adversary lies behind the player at the end of Phase 1. Then in Phase 2, Φ_1 decreases by $d(s',s'')$ and Φ_2 and Φ_3 do not increase. Thus Φ– their total– decreases by at least $0 \ge (m^{1/3}+1)d(s',r) - 2m^{1/3}d(s',r)$ $\ge (m^{1/3}+1)d(s',s'') - 2m^{1/3}d(x',r)$.

So assume $x' \in (s',r]$. To handle this difficult case we further subdivide the player's Phase 2 motion.

Let z be such that $m^{1/3}d(s',z) = d(x',r)$, and let s^* be the first of s'', z, and x' after s'. As the player moves from s' to s^*, Φ_1 decreases by $d(s',s^*)$ and Φ_2 decreases by $m^{1/3}d(s',s^*)$. Now let r^* be the point such that $d(r^*,r) = m^{1/3}d(s',s^*)$. Note that $r^* = x'$ exactly when $s^* = z$; otherwise, r^* is in $(x',r]$. Also note that $r^* \in [s'',r]$ since

$$\begin{aligned} d(r^*,r) &= m^{1/3}d(s',s^*) \\ &\le m^{1/3}d(s',s'') \\ &\le m^{1/3}\frac{d(s',r) - m^{1/3}}{m^{1/3}+1} \\ &\le d(s',r) - m^{1/3} \\ &\le d(s'',r). \end{aligned}$$

The increase in Φ_3 as the player moves from s' to s^* is at most $2m^{2/3}d(s',s^*) = 2m^{1/3}d(r^*,r)$. Altogether, Φ decreases by at least $(m^{1/3}+1)d(s',s^*) - 2m^{1/3}d(r^*,r)$ in this first "subphase".

For the second subphase– from s^* to s''– we separately consider the three cases: (1) $s^* = s''$, (2) $s^* = z$ (and hence $x' = r^*$), and (3) $s^* = x'$. We assert that in each case, Φ decreases by at least $(m^{1/3}+1) \cdot d(s^*,s'') - 2m^{1/3}d(x',r^*)$. Summing this assertion with the bound above for the first subphase will complete the proof.

In case (1), $d(s^*,s'') = 0$ and the assertion is trivially true. In case (2),

$$\begin{aligned} d(s'',x') &= d(s',r) - d(s',s'') - d(x',r) \\ &\ge d(s',r) - d(s',s'') - m^{1/3}d(s',s^*) \\ &\ge d(s',r) - (m^{1/3}+1) \cdot d(s',s'') \\ &\ge d(s',r) - (d(s',r) - m^{1/3}) \\ &\ge m^{1/3} \end{aligned}$$

follows from the definition of Two-Phase. Thus Φ_3 does not increase during the motion from s^* to s''. $\Phi_1 + \Phi_2$ decreases by at least $(m^{1/3}+1)d(s^*,s'')$, so the total decrease in Φ satisfies the assertion.

In case (3) the player crosses over the adversary at s^*. Recall that Φ is continuous at the crossover. In the subsequent motion from s^* to s'', Φ_1 decreases by $d(s^*, s'')$, while Φ_2 and Φ_3 remain fixed at m and 0. So Φ decreases by $d(s^*, s'') = (m^{1/3} + 1)d(s^*, s'') - m^{1/3}d(s^*, s'')$, which is at least $(m^{1/3} + 1)d(s^*, s'') - 2m^{1/3}d(x', r^*)$ since $d(x', r^*) = d(s^*, r^*) \geq d(s^*, s'')$. ■

The next lemma relates $\Delta\Phi$, the total change in Φ during steps (1) and (2), to the total player and adversary costs W_P and W_A (in all three steps).

Lemma 3. *For each request r, $W_P + \Delta\Phi \leq 4m^{1/3} \cdot W_A$.*

Proof: The adversary's work $W_A = 1 + d(x, x') + d(x', r)$, since an optimal offline algorithm would not waste motion by going all the way around the circle.

Assume first that $d(x', r) > m^{2/3} - m^{1/3}$. In this case the lemma follows from $W_P \leq m$ and the fact that at all times $0 \leq \Phi \leq 2m$.

So assume that $d(x', r) \leq m^{2/3} - m^{1/3}$. By Lemma 1, the increase in Φ in step (1) is bounded by

$$(m^{1/3} + 1)d(x, x').$$

The player's work W_P in step (2) is $1 + d(s, s'') + d(s'', r)$, which is no greater than

$$d(s, s') + (m^{1/3} + 1)d(s', s'') + m^{1/3} + 1.$$

Finally, by Lemma 2, Φ decreases in step (2) by at least

$$d(s, s') + (m^{1/3} + 1)d(s', s'') - 2m^{1/3}d(x', r).$$

Subtracting this expression from the sum of the first two and then dividing by $1 + d(x, x') + d(x', r)$, we obtain $(W_P + \Delta\Phi)/W_A \leq 2m^{1/3}$, which implies the lemma. ■

Theorem 2. Two-Phase *is $4m^{1/3}$-competitive.*

Proof: Let Φ_i be the initial value of Φ before request sequence R_n, and let Φ_f be the final value of Φ after R_n. The theorem then follows from Lemma 3– summed over all requests– and the fact that $\Phi_f - \Phi_i \leq 2m$. ■

The 1-server problem on the line is quite different. The restart move, in which a server jumps to location 0, introduces many of the complexities of forking. We do not have an $O(m^{1/3})$ upper bound for this problem.

5. Forking Disallowed

In this section we generalize Theorem 2 to give a $2(k+1)m^{1/3}$-competitive algorithm for locating checkpoints in the case that neither the online nor the offline algorithm is allowed to fork. The 1-server lower bound argument can be extended to give a matching $\Omega(m^{1/3})$ lower bound by adding requests that "freeze" $k-1$ of the player's servers, as in the proof of Theorem 4 below.

In response to each request r, the algorithm applies Two-Phase to the server nearest to r. We analyze this algorithm's performance as in Section 4, only this time the potential function is somewhat more elaborate. For a player server at s and adversary server at x we define the following functions:

$$l(s, x) = d(s, x) + \max\{0, m^{2/3}(m^{1/3} - d(s, x))\},$$

$$f(s, x) = m - d(s, x) + \min\{m, m^{1/3} \cdot d(s, x)\}.$$

Let the positions of player (adversary) servers $1, 2, \ldots, k$ be denoted $s_1, s_2, \ldots, s_k$ (respectively, $x_1, x_2, \ldots, x_k$). Next we define M to be the minimum weight of a matching of player servers to adversary servers where the weight of matching player server i to adversary server j is $l(s_i, x_j)$. We now define our potential function,

$$\Phi = (k+1)M + \sum_{i,j} f(s_i, x_j).$$

Roughly speaking, we match players to adversaries in Φ so that a player's motion is paid for by increased proximity to its matched adversary. As in the previous section, weights are piecewise-linear functions in order to compensate for the varying "speed" of Two-Phase.

As above we divide the response to request r into steps: (1) the adversary moves arbitrarily; (2) the player executes Two-Phase; and (3) the adversary services r without moving a permanent checkpoint.

Lemma 4. *When an adversary server moves distance d, Φ increases by at most $km^{1/3}d + (k+1)d$.*

Proof: Observe first that $l(s, x)$ and $f(s, x)$ are both continuous at crossovers. Since "nested" matched pairs have the same total distance as "crossed" matched pairs, M is also continuous at crossovers, and hence Φ is as well. When adversary server x_j moves distance d without crossing a player, M increases by at most d and for each i, $f(s_i, x_j)$ increases by at most $m^{1/3}d$. ■

As above let s denote the initial position of the player that services r, and s' and s'' its positions after Phase 1 and Phase 2, respectively. Denote by x' the position of the adversary closest to r after step (1).

Lemma 5. *If $x' \in [s, r]$ then there exists a minimum weight matching in which the distance from the player at s to its matched adversary is at least $d(s, x')$.*

Proof: Assume that s is matched to some adversary at x^* with $x^* \in [s, x')$, while the adversary at x' is matched to some other player at s^*, necessarily behind s (i.e., in (r, s)). We show that switching these

two matched pairs does not increase the weight of the matching, that is,

$$l(s, x^*) + l(s^*, x') \geq l(s, x') + l(s^*, x^*).$$

In backwards order from r, we have x', x^*, s, and s^*. Our first observation is $d(s, x^*) + d(s^*, x') = d(s, x') + d(s^*, x^*)$. Thus we have only to show that

$$\max\{0, m - m^{2/3} d(s, x^*)\} + \max\{0, m - m^{2/3} d(s^*, x')\} \geq$$

$$\max\{0, m - m^{2/3} d(s, x')\} + \max\{0, m - m^{2/3} d(s^*, x^*)\}.$$

This follows from our first observation if all quantities within the brackets are nonnegative. Also if $d(s^*, x') > m^{1/3}$ and all other distances are no greater than $m^{1/3}$, the inequality holds strictly. If one of the distances on the righthand side is greater than $m^{1/3}$, then so is $d(s^*, x')$, and then the inequality holds since $d(s, x')$ and $d(s^*, x^*)$ are both at least $d(s, x^*)$. ■

Lemma 6. *Assume $d(x', r) \leq m^{2/3} - m^{1/3}$. Then in Phase 1, Φ decreases by at least $d(s, s')$, and in Phase 2, Φ decreases by at least $(m^{1/3} + 1) \cdot (d, s', s'') - (k + 1)m^{1/3} \cdot d(x', r)$.*

Proof: We consider Phase 1 first and assume $d(s, s') > 0$ (so $d(s, r) > m^{2/3}$). If the player at s is not matched to the adversary at x', then by Lemma 5, we may assume that it is matched to an adversary at x^* with $x^* \in (r, s)$. In either case the matched adversary is at least $m^{1/3}$ past s', so the weight of the current matching decreases by $d(s, s')$ as player moves from s to s'. Hence M decreases by at least this amount. The k functions $f(s, x_j)$ each increase by at most $d(s, s')$. So overall Φ decreases by at least $(k+1)d(s, s') - kd(s, s')$, which is $d(s, s')$.

For Phase 2, first assume that the player at s' is not matched to an adversary in $[s', r]$ by any minimum weight matching. Then the adversary matched to s' lies at least $m^{1/3}$ beyond s''. Thus the matching part of the potential function decreases by at least $(k + 1)d(s', s'')$ as the player moves from s' to s''. Since $d(s', x') \leq m^{2/3}$, the function that is initially $f(s', x')$ decreases by $(m^{1/3} - 1)d(s', s'')$ during this motion. Each of the other $f(s', x_j)$ terms increases by at most $d(s', s'')$, and thus overall Φ decreases by at least $(m^{1/3} + 1)d(s', s'')$.

So assume that the player at s' is matched to an adversary in $[s', r]$. By Lemma 5, we may assume that s' is matched to x'. The analysis of this case closely follows that of Lemma 2 and shows that Φ decreases by at least $(m^{1/3} + 1) \cdot (d, s', s'') - (k+1)m^{1/3} \cdot d(x', r)$. ■

Lemma 7. *For each r, $W_P + \Delta\Phi \leq 2(k + 1)m^{1/3} W_A$.*

Proof: W_A is at least $1 + d(x', r)$ plus the motion in step (1). By Lemma 4, the change in Φ in step (1) is bounded by the distance moved times $km^{1/3} + k + 1$; this factor is less than $2(k + 1)m^{1/3}$.

First assume $d(x', r) \leq m^{2/3} - m^{1/3}$. The player's cost $W_S = d(s, s') + d(s', s'') + d(s'', r) + 1$, which is no greater than

$$d(s, s') + (m^{1/3} + 1)d(s', s'') + m^{1/3} + 1.$$

By Lemma 6, the decrease in Φ in step (2) is at least

$$d(s, s') + (m^{1/3} + 1)d(s', s'') - (k + 1)m^{1/3} d(x', r).$$

Subtracting this expression from the previous one and dividing by $1 + d(x', r)$ now gives the result.

If $d(x', r) > m^{2/3} - m^{1/3}$, the lemma follows from $W_P \leq m$, $W_A \geq m^{2/3} - m^{1/3} + 1$, and the fact that the increase in Φ due to the movement of any one player is at most $(2k + 1)m - km^{2/3}$. ■

Theorem 3. Two-Phase *is $2(k+1)m^{1/3}$-competitive for locating k checkpoints in the case that forks are disallowed.* ■

6. Lower Bounds for 3 or More Servers

In this section forking is allowed. We prove a lower bound of $\Omega(m^{1/2})$, which is much larger than the upper bound proved in the last section for the case of forking disallowed. Thus the fork move is major contributor to the difficulty of competitiveness.

Theorem 4. *For $k \geq 3$, a k-server algorithm can be no better than $\Omega(m^{1/2})$-competitive.*

Proof: We first give the proof in the case that $k = 3$, followed by the modifications for the general case.

The adversary incurs a one-time cost of $m/2$ to position two servers on opposite sides of the circle, and then the proof proceeds as before in epochs, where each epoch begins with two of the adversary's servers at z_1 and $z_2 = z_1 + m/2$ and with history R_t. The adversary considers extending R_t by $\lceil m^{1/2} \rceil$ alternating requests at $z_1 + m^{1/2}$ and $z_2 + m^{1/2}$, followed by a single request at either z_1 or z_2, and the adversary processes these requests by placing servers at $z_1 + m^{1/2}$, $z_2 + m^{1/2}$, and at one of z_1 or z_2 for the last request. Thus the adversary's solution will have cost $2m^{1/2} + \lceil m^{1/2} \rceil$ for the epoch. If the player fails to leave a server in both $(z_1 + m^{1/2}, z_2]$ and $(z_2 + m^{1/2}, z_1]$ then the last request is at the forward endpoint, z_2 or z_1, of the arc that was empty at some time during the epoch. This last request implies at least $m/2 - m^{1/2} + \lceil m^{1/2} \rceil$ player cost for the epoch. On the other hand, if the player maintains servers in both $(z_1 + m^{1/2}, z_2]$ and $(z_2 + m^{1/2}, z_1]$, then the alternating requests must be processed with only

one additional server, so one of the arcs $(z_1, z_1 + m^{1/2}]$ or $(z_2, z_2 + m^{1/2}]$ is left empty after each alternation, and yet both forward endpoints must be processed during each alternation. The player obtains a total cost of at least $\lceil m^{1/2} \rceil m^{1/2}/2$. In either case the player is no better than $((m^{1/2} - 1)/6)$-competitive.

After this epoch, the adversary can repeat the pattern with $z_1 + m^{1/2}$ and $z_2 + m^{1/2}$ serving as z_1 and z_2. By using the fork move, the adversary moves its third server to whichever of these two positions will be the endpoint of an arc of length $m/2 - m^{1/2}$ not occupied by a player server at some point in the next epoch.

We now consider the case of $k > 3$. As above the adversary places servers at 0 and $m/2$. It also places servers at $k-3$ other locations $z_1, z_2, \ldots, z_{k-3}$ that are $m^{1/2}$ apart in the interval $[m/8, 3m/8]$. The adversary incurs total set-up cost $O(km)$ and has one server left over.

Now we have epochs of size $3\lceil m^{1/2} \rceil$. Each epoch is divided into *cycles* of 3 requests. Each cycle consists of a request at $m^{1/2}$, another at $m/2 + m^{1/2}$, and a third request at one of the locations z_i.

The adversary considers all possible ways to extend the current history R_t with an epoch of this form. If the player leaves either $(m - m/8, 0]$ or $(3m/8, m/2]$ empty at any time in one of these extensions, the adversary chooses that extension and follows the epoch with a request at whichever of 0 and $m/2$ is at the endpoint of a vacated arc. In this case the player's cost exceeds $m/8$. In answering this epoch, the adversary does not move its servers at $z_1, z_2, \ldots, z_{k-3}$. It advances its servers at 0 and $m/2$ to $m^{1/2}$ and $m/2 + m^{1/2}$, leaving its leftover server at the location of the final request. Thus the adversary incurs total cost about $5m^{1/2}$.

So assume that the player leaves arcs $(m - m/8, 0]$ and $(3m/8, m/2]$ occupied at all times. In this case, either the player pays at least $m^{1/2}$ per request for one of the locations $m^{1/2}$ or $m/2 + m^{1/2}$, or the player leaves an arc of the form $(z_i - m^{1/2}, z_i]$ empty at some point during each cycle. If the latter is true, then the adversary "hits 'em where they ain't" by choosing the third request of each cycle to be at a location z_i that was– at some time during that cycle– the endpoint of an empty arc. Either way the player's cost per cycle is at least $m^{1/2}$ and cost for the entire epoch at least $m/3$. As above the adversary's cost is about $5m^{1/2}$.

The pattern can now repeat with $m^{1/2}$ and $m/2 + m^{1/2}$ serving as 0 and m. Locations $z_1, z_2, \ldots, z_{k-3}$ need not change until the adversary server that occupies successive positions 0, $m^{1/2}$, $2m^{1/2}$, ..., comes too close to the first z_i. This happens only after $\Omega(m^{1/2})$ epochs, by which time the player has incurred $\Omega(m^{3/2})$ total cost. The adversary can now reset incurring an asymptotically negligible cost of $O(km)$. ■

It is straightforward to show that Theorem 4 also holds for the case of servers moving on a line.

7. An Upper Bound for the Problem with Forking

In this section the number of servers is at least 2 and as above both adversary and player may fork (though in our algorithm the player never does). We show that the following algorithm, called HOLDBACK, is $3(km)^{1/2}$-competitive.

for each request r **do**
  Let s be the position of the server nearest r
  Move a closest server $\max\{0, d(s, r) - (km)^{1/2}\}$
  Serve r using the temporary server **od**

For B a subset of the set of adversary servers A, let $M(B)$ be the minimum cost of a matching of members of B to player servers, where the cost of matching adversary server j with player server i behind j is $d(s_i, x_j)$. For adversary server j, let $prev(j) = \min_{l \neq j}\{d(x_l, x_j)\}$ if $d(x_l, x_j) > 0$ for all $l \neq j$. If several adversary servers occupy a single position, let all but one of them have *prev* value 0 and an arbitrarily chosen one, say j, have *prev* value equal to the minimum nonzero value of $d(x_l, x_j)$. Now define

$$\Phi(B) = M(B) + 2(km)^{1/2} \sum_{j \notin B} prev(j)$$

and our potential function

$$\Phi = \min_{B \subseteq A} \{\Phi(B)\}.$$

Here the new ingredient in Φ is that we choose a subset of the adversaries to match. This added level of optimization keeps Φ from jumping up when the adversary forks.

We think of the actions following request r as consisting of the following steps: (1) the player moves according to HOLDBACK above, and (2) the adversary moves arbitrarily (and serves request r). First observe that at all times $0 \leq \Phi \leq km$ since $M(A) \leq km$. Let $s = s_i$ be the position of player server i, a closest player server to request r, and let s' be the position of server i after step (1).

Lemma 8. *Assume $d(s, r) \geq (km)^{1/2}$ and that there is an adversary server at position x such that $d(x, r) \leq (km)^{1/2}/2$. Then in step (1), Φ decreases by at least $d(s, s')$.*

Proof: The assumptions above imply that there is an adversary server between s and r. Thus in a matching

$M(B)$ for which $\Phi(B)$ is minimum, some adversary l is paired with player server i. If x_l lies in $[s', s)$ then $M(B)$ decreases by $d(s, s')$ in step (1), so Φ must decrease by at least $d(s, s')$.

So assume x_l lies in nonempty $[s, s')$. In this case $d(s', r) = (km)^{1/2}$, so there must be another adversary server h in $[s', r]$. Let us now assume $h \in B$ for a choice of B that minimized $\Phi(B)$ before step (1). Then since player server i was the closest player to r, adversary h must have been paired with a server between r and s. After step (1) we can pair adversary h with player i and adversary l with h's old partner. This new pairing reduces $M(B)$ by $d(s, s')$; hence Φ must decrease by at least $d(s, s')$.

The remaining case is that all adversary servers in $[s', r]$ are in $A \setminus B$ (before step (1)) for each B that minimizes $\Phi(B)$. Then Φ before step (1) is at least $2(km)^{1/2} \sum prev(j)$, where the sum is over all adversaries between s' and r. Either this sum is greater than $d(s', r)/2$, which is at least $(km)^{1/2}/2$, or there is no adversary server h such that $d(x_h, r) \leq (km)^{1/2}/2$. The latter case contradicts one of our assumptions; the former case contradicts $\Phi \leq km$. ■

Lemma 9. *(1) When the adversary forks, Φ cannot increase. (2) When an adversary server j moves distance d, Φ increases by at most $2d(km)^{1/2}$.*

Proof: When adversary j is forked to be coincident with adversary l, we may assume that $prev(j)$ becomes 0 and $prev(l)$ retains its former value. So j's contribution to Φ decreases to 0. The contribution of the first adversary ahead of j's old location increases by no more than j's old contribution to Φ.

In order to prove (2), let B be a set of adversary servers that minimizes $\Phi(B)$ before the adversary motion. After j moves, $\Phi(B)$ has increased by at most d in the case $j \in B$ and by at most $2d(km)^{1/2}$ in the case that $j \notin B$ and $2(km)^{1/2} \cdot prev(j)$ appears in $\Phi(B)$. Hence Φ increases by at most $2d(km)^{1/2}$. ■

Theorem 5. HOLDBACK *is $3(km)^{1/2}$-competitive.*

Proof: We show that for each request r, $W_P + \Delta\Phi \leq 3(km)^{1/2} W_A$, where $\Delta\Phi$ is the total change in the potential function in both steps. This inequality and the invariant $0 \leq \Phi \leq km$ suffice to prove the theorem.

First assume that there is no adversary server j such that $d(x_j, r) \leq (km)^{1/2}/2$ when request r arrives. Then $W_A > (km)^{1/2}/2$ and the right-hand side above is greater than $(3/2)km$. The work done by the player is at most m, and the increase in Φ can be at most km, so the left-hand side is smaller than the right-hand side for $k \geq 2$.

Now assume that there is an adversary server j such that $d(x_j, r) \leq (km)^{1/2}/2$. If $W_P < (km)^{1/2}/2$, then the player does not move in HOLDBACK, and hence does not increase Φ. By Lemma 9, $\Delta\Phi \leq 2(km)^{1/2} W_A$, and the fact that $W_A \geq 1$ implies the inequality above. So we now assume that $W_P \geq (km)^{1/2}$. Then by Lemma 8, in step (1) Φ decreases by at least the distance that the player server moves before sending a temporary server, that is, at least $W_P - (km)^{1/2}$. By Lemma 9, the increase in Φ during step (2) is at most $2(km)^{1/2} W_A$. Thus $W_P + \Delta\Phi \leq (km)^{1/2} + 2(km)^{1/2} W_A \leq 3(km)^{1/2} W_A$. ■

HOLDBACK and other checkpoint algorithms for the circle can be extended to the line by thinking of the line as a circle with $k + 1$ servers, one of which is fixed at 0. (This restriction applies to both the player and the adversary.) Whenever HOLDBACK tries to move the server at 0, instead the closest server behind 0 should be moved to the same spot. The analysis can be carried out in almost the same manner as above to prove that this version of HOLDBACK is $O((km)^{1/2})$-competitive for the checkpoint problem on the line.

8. Lower Bounds for Unequal Numbers of Servers

So far we have only compared online algorithms against offline algorithms with the same number of servers. Time was the resource used to measure competitiveness. It is natural to explore the space resource as well by allowing the online algorithms more servers than the offline algorithms. (Here we are following the lead of Manasse et al [11].) The next theorem also shows that our lower bounds are robust; strong (i.e., m^ϵ) lower bounds still hold even when the player is allowed k servers and the adversary only 1.

Theorem 6. *On-line algorithms with k servers can be no better than $\Omega(m^{\frac{1}{2^{k+1}-1}})$ competitive when compared to the 1-server optimal algorithm.*

Proof: The proof depends on a hierarchy of epoch sizes $\lceil m^{\alpha_1} \rceil, \lceil m^{\alpha_2} \rceil, \ldots, \lceil m^{\alpha_k} \rceil$, (given largest to smallest) defined by the recurrence:

$$\alpha_i = 2\alpha_{i+1} + \alpha_k, \qquad \alpha_k = \frac{1}{2^{k+1} - 1}$$

or the closed form:

$$\alpha_i = \frac{2^{k+1-i} - 1}{2^{k+1} - 1}.$$

At the beginning of an epoch, the adversary is at an arbitrary location z_1. (Thus the adversary's strategy can be repeated for any number of epochs.) The

adversary considers all possible ways of extending the current request sequence R_t by $\lceil m^{\alpha_1} \rceil$ requests in the arc $[z_1 + m^{\alpha_1}, z_1 + km^{\alpha_1}]$. If one of these (infinite number of) extensions causes the player to vacate the arc $(z_1 + km^{\alpha_1}, z_1]$, then the adversary chooses this extension, followed by a request at z_1, and processes the sequence by leaving its server at z_1. This results in player cost $\Omega(m)$ while the adversary cost is only $O(m^{2\alpha_1})$, giving the ratio claimed in the theorem.

On the other hand, if the player would maintain a server in $(z_1 + km^{\alpha_1}, z_1]$ for all possible extensions, then the adversary divides the epoch into $\Omega(m^{\alpha_1 - \alpha_2})$ subepochs of duration $\lceil m^{\alpha_2} \rceil$ each. The adversary starts the jth subepoch with its server at location $z_1 + m^{\alpha_1} + (j-1)(k-1)m^{\alpha_2}$. In each subepoch, the adversary considers all extensions of R_t by $\lceil m^{\alpha_2} \rceil$ requests in an arc of length $(k-2)m^{\alpha_2}$ starting m^{α_2} ahead of its server. If the player fails to maintain a server behind the adversary's location, a final request at that location forces the player's cost to $\Omega(m^{\alpha_1})$ while the adversary's cost is $O(m^{\alpha_2})$, thus giving the ratio claimed in the theorem. If the player would maintain a server, then that subepoch is further divided into subsubepochs of duration $\lceil m^{\alpha_3} \rceil$ comprising requests in subarcs of the subepoch's arc. In the ith level of this recursion a subarc has the form $[z_i + m^{\alpha_i}, z_i + (k-i+1)m^{\alpha_i}]$, where the choices of z_i are such that these subarcs are evenly spaced within a subarc of level $i-1$. The recursion terminates with a case identical to the single server proof of Section 3, since the player has $k-1$ confined servers and hence only one server free to process requests in the most deeply nested, zero-length subarc. ■

9. The Offline Problem

In this section we give algorithms for the problem of computing offline an optimal sequence of responses to a sequence of requests $r_1, r_2, \ldots, r_n$. We first give an algorithm for the case $k = 1$. Let $C_i(s)$ be the minimum cost of serving requests $r_1, r_2, \ldots, r_i$ and leaving the server at position $s \in [0, m)$. We have the recurrence

$$C_i(s) = \min\{C_{i-1}(s) + d(s, r_i),\ C_{i-1}(r_i) + d(r_i, s)\},$$

where the first term corresponds to the option of leaving the server at s and serving the ith request with the temporary server and the second term corresponds to the option of leaving the server at r_i after the previous request and then moving to position s after serving the ith request. Other possibilities, such as leaving a server at a position s' and then moving from s' to r_i to s, are dominated by these two options. The initial condition is $C_0(s) = s$.

Lemma 10. *$C_i(s)$ is a continuous, piecewise-linear function with at most $i+1$ pieces and maximum slope 1.*

Let p denote the number of distinct locations among $r_1, r_2, \ldots, r_n$. Obviously $p \le n$; we state our running times in terms of p rather than n as in many applications $p \ll n$. The function $C_i(s)$ can be stored implicitly in a complete binary tree with $p+1$ leaves in which each node stores a linear function of s. The value of this function at a specific s is computed by summing values from a leaf to the root. This data structure can be updated from $C_{i-1}(s)$ to $C_i(s)$ in amortized time $O(\log p)$. We omit the details.

Theorem 7. *For $k = 1$, the offline problem can be solved in time $O(n \log p)$, where $p \le n$ is the number of distinct request locations.*

The general offline problem can also be solved by dynamic programming, though in this case we did not speed up the algorithm using special data structures (as it looks rather messy). Note that although standard k-server problems can be reduced to minimum-cost flow [5], the checkpointing problem is quite nonlinear due to forking and excursions.

Theorem 8. *For fixed $k \ge 2$, the offline problem can be solved in time $O(np^k)$.*

10. Conclusions

We have explored adaptive online schemes for locating checkpoints. To do so, we introduced a server problem that includes several nonstandard features: a fixed cost per request, one-way motion, excursions, and forking. Including the fixed cost enables us to differentiate algorithms that would otherwise have simply been declared noncompetitive. One-way motion and excursions taken together raise the optimal competitiveness from k (the number of servers [5, 11]) to about $m^{1/3}$, where m is in effect the size of the playing field. Forking further raises this bound to about $m^{1/2}$. A number of interesting open problems remain:

- Does there exist an $O(m^{1/3})$-competitive algorithm for locating checkpoints in the case $k = 2$? (This question raises the important issue of whether forking is of any use to the player. For $k = 2$, we can prove a lower bound of $\Omega(m^{1/2})$ on the competitiveness of any online algorithm that does not fork.)

- Except for $k = 2$, our upper and lower bounds match in their dependence on the dominant factor

m. There remain, however, constant gaps and gaps depending on k. Can these be closed?

- What happens to our bounds if we disallow excursions? That is, memory is now assumed homogeneous.
- What is the effect of allowing forking on other server problems?

For some problems, most notably accessing a linear list, competitive analysis seems to give "the right answer"– that is, it leads to an algorithm that arguably dominates all others. For the checkpointing problem, the situation is less clear. We believe that competitive analysis has demonstrated the utility of an initially rapid, then increasingly slow, approach to a repeated request location. We also think that it has invalidated some initially attractive algorithms, such as one that always moves halfway towards a request. On the other hand, due to its emphasis on worst-case sequences, competitive analysis may have led us to overly conservative algorithms. In practice one would probably want to move a checkpoint closer than the distance $(km)^{1/2}$ prescribed by HOLDBACK.

In fact the choice of a practical algorithm, whether HOLDBACK, TWO-PHASE (which is slightly more aggressive), or something else, should depend on how "adversarial" are the expected request sequences. In applications such as debugging or physical simulation, there may be a small number of "hot spots" and users may often step backwards in time. In such situations request sequences may indeed appear quite adversarial.

It would be interesting to investigate the checkpoint location problem using other styles of analysis, such as probabilistic analysis assuming random (possibly correlated) requests.

Acknowledgements

We would like to thank Mike Paterson and Howard Karloff for some valuable discussions.

References

[1] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson, On the Power of Randomization in Online Algorithms, these proceedings.

[2] P. Berman, H. Karloff, and G. Tardos, A Competitive Three-Server Algorithm, *1st ACM-SIAM Symp. on Discrete Algorithms*, 1989.

[3] A. Borodin, N. Linial, and M. Saks, An Optimal Online Algorithm for Metrical Task Systems, *19th ACM Symp. on Theory of Computing*, 1987.

[4] A. Calderbank, E. Coffman, and L. Flatto, Sequencing Problems in Two-server Systems, *Math. Oper. Research* 10, 1985, 585-598.

[5] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan, New Results on Server Problems, *1st ACM-SIAM Symp. on Discrete Algorithms*, 1989.

[6] M. Chrobak and L. Larmore, An Optimal On-Line Algorithm for k Servers on Trees, manuscript, UC - Riverside, 1989.

[7] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir, Random Walks on Weighted Graphs, and Applications to On-line Algorithms, these proceedings.

[8] E. Fiala and D. Greene, Data Compression with Finite Windows, *CACM 32*, April 1989, 490-505.

[9] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator, Competitive Snoopy Caching, *Algorithmica* 3, 1988, 79-119.

[10] R. Korf, Complexity of Reverse Execution, manuscript, 1981.

[11] M. Manasse, L. McGeoch, and D. Sleator, Competitive Algorithms for On-line Problems, *20th ACM Symp. on Theory of Computing*, 1988.

[12] P. Raghavan and M. Snir, Memory vs. Randomization in Online Algorithms, *ICALP*, 1989.

[13] D. Sleator and R. Tarjan, Amortized Efficiency of List Update and Paging Rules, *CACM* 28, February 1985, 202-208.

[14] J. Storer, *Data Compression*, Computer Science Press, 1988.