Algorithmica (1994) 11: 73–91

Algorithmica
© 1994 Springer-Verlag New York Inc.

A New Measure for the Study of On-Line Algorithms

S. Ben-David[1] and A. Borodin[2]

Abstract. An accepted measure for the performance of an on-line algorithm is the "competitive ratio" introduced by Sleator and Tarjan. This measure is well motivated and has led to the development of a mathematical theory for on-line algorithms.

We investigate the behavior of this measure with respect to memory needs and benefits of lookahead and find some counterintuitive features. We present lower bounds on the size of memory devoted to recording the past. It is also observed that the competitive ratio reflects no improvement in the performance of an on-line algorithm due to any (finite) amount of lookahead.

We offer an alternative measure that exhibits a different and, in some respects, more intuitive behavior. In particular, we demonstrate the use of our new measure by analyzing the tradeoff between the amortized cost of on-line algorithms for the paging problem and the amount of lookahead available to them. We also derive on-line algorithms for the K-server problem on any bounded metric space, which, relative to the new measure, are optimal among all on-line algorithms (up to a factor of 2) and are within a factor of $2K$ from the optimal off-line performance.

Key Words. On-line algorithms, Competitive analysis.

1. Introduction.

We consider the problem of carrying out a sequence of tasks subject to some cost function where at any given moment a current task has to be processed without knowledge of future requests.

In a seminal paper, Sleator and Tarjan [ST] introduce a measure for analyzing the performance of on-line algorithms. In particular, they study the move to front strategy for list accessing and various strategies for paging. Rather than study the performance of such on-line algorithms relative to probabilistic assumptions about the input distribution, they relate the on-line cost to that of an optimal off-line algorithm. An algorithm is said to be *M-competitive* if the cost incurred by using it on any request sequence is never more than M times the minimal possible (off-line) cost of serving that sequence (up to an additive constant). The resulting measure, called the competitive ratio in [KMRS], has now been investigated in a number of specific and abstract settings. Although the competitive ratio is in general a pessimistic bound (in that it applies even when the future is completely unpredictable), there are often algorithms yielding surprisingly good bounds on this ratio.

On the other hand, since on-line versus off-line considerations are so pervasive, it should not be expected that consideration of the competitive ratio (or any one measure) will always lead to efficient (or even reasonable) algorithms.

[1] Department of Computer Science, Technion, Haifa, Israel.
[2] Department of Computer Science, University of Toronto, Toronto, Ontario, Canada.

Received August 7, 1990; revised April 16, 1992. Communicated by Prabhakar Raghavan.

The following example describes a fairly common situation in which we choose to abandon all competitive strategies in favor of a noncompetitive one. Consider the question of buying an insurance policy. Paying \$5 a month to insure one's car against theft is a noncompetitive strategy! The mere existence of a scenario where one will never present a claim to the insurance agent suffices to make it so. On such a scenario the all-knowing off-line strategy would have never bought the policy.

We introduce another measure, which we call the Max/Max ratio. Simply stated, the Max/Max ratio compares the worst-case amortized behavior of an algorithm with that of an optimal off-line algorithm. An algorithm is said to have Max/Max ratio M if it is guaranteed that on no request sequence will it ever have to pay more than M times the maximal cost that an optimal off-line algorithm pays on a sequence of the same length (up to an additive constant). Thus our "new" measure is just a revival of the traditional worst-case amortized analysis except that we now normalize by the best that can be done using an optimal off-line algorithm. We discuss the basic features of this new measure and investigate the types of on-line algorithms it suggests.

We consider several aspects of the on-line versus off-line problem, such as the effects of available lookahead and the dependence of the on-line algorithm upon memory. We investigate both the competitive and the Max/Max measure with respect to these aspects and show that, as far as these questions are concerned, the Max/Max gives rise to a perhaps more intuitive behavior.

Using a simple argument, we show that the definition of the competitive ratio measure implies that it does not reflect any improvement in an on-line agent's behavior due to any finite amount of lookahead. This is a rather disappointing aspect of that measure. With respect to the Max/Max ratio, we will see that the potential benefit of finite lookahead becomes a central issue.

One of the more enticing aspects of the perspective provided by the competitive ratio is the role of memory. In fact, at first it appears to be a paradox that the known competitive algorithms make essential use of memory even though the past is uncorrelated with the future. The paradox is explained by observing that the goal of a competitive algorithm is not to be optimal on any sequence but rather to make sure that the optimal (off-line) algorithm cannot be doing too much better. That is, it is sometimes beneficial to increase one's own cost, if in doing so it can be guaranteed that the optimal off-line cost grows accordingly.

In Section 3 we examine this question. We show that memory dependence is inherent in the competitive ratio by giving some lower bounds on the amount of memory needed for running competitive on-line algorithms. Our results imply that, for certain on-line tasks, any competitive algorithm should devote unbounded (i.e., must grow with certain parameters of the metric space) memory space to recording past behavior. We limit our discussion to deterministic algorithms. Randomization can compensate for memory constraints, but our arguments can be applied to derive similar lower bounds on the product of the number of memory bits and the number of random bits used by any competitive algorithm.

This paradoxical behavior seems to disappear when considering the Max/Max measure. We show, in Section 5, that once efficiency is measured via the Max/Max ratio, then, for every metric space, there exists a $2K$-competitive memoryless on-line algorithm for the K-server problem on that space.

For definiteness we formulate our considerations in terms of the K-server model of Manasse *et al.* [MMS2] although the issues transcend this particular abstract model. In the next section we introduce the required definitions and notation. Section 3 investigates the issues of memory and the benefits of lookahead for the competitive measure. Starting in Section 4 we shift our focus to the Max/Max measure. We first introduce the new measure and its basic properties. In Section 5 we proceed to show that for every bounded K-server system there is a natural, simple, and memoryless on-line algorithm achieving a Max/Max ratio of $2K$. This should be contrasted with the still open and intriguing K-server conjecture of [MMS2], of whether or not for every K-server system the competitive ratio is equal to K. (Fiat *et al.* [FRR] and Grove [G] have made substantial progress on this conjecture by showing that the competitive ratio for every K-server system is bounded by an (exponential) function of K.) An additional benefit of our Max/Max measure is that on-line algorithms can be directly compared (i.e., without reference to the performance of an optimal off-line algorithm whose analysis might be quite complex). Our natural and memoryless on-line algorithm is shown to be within a factor of 2 of *any* on-line memoryless algorithm. In Section 6 we begin the study of lookahead, starting with a complete analysis of "paging." In particular, we show that with lookahead $l = n - 1$, an on-line player can perform optimally in the K-server paging problem; that is, the Max/Max ratio equals one! We then consider arbitrary K-server problems and conclude with our own version of a K-server conjecture.

2. Definitions and Notation.

A K-server system consists of a metric space $(V; d)$ where V is a set of nodes (possibly infinite) and d is a metric defined on the nodes. That is, d: $V \times V \to \mathbf{R}^{>0}$ is symmetric and satisfies the triangle inequality.[3] At any time, K-servers are located on the nodes, where $K < |V|$. Following a *request* for a node v, if no server is on v, at least one server must be moved from its present location u to v at a cost of $d(u, v)$. Given a request sequence $\bar{\sigma} = v_1, v_2, \ldots, v_t$, an algorithm $\mathscr{A}$ responds by an appropriate sequence of moves.

Given a K-server system and an initial configuration S_0 of servers, an algorithm $\mathscr{A}$, when applied to a request sequence $\bar{\sigma} = v_1, \ldots, v_t$, produces a sequence of configurations $S_1, S_2, \ldots, S_t$ such that $v_t \in S_t \subseteq V$.[4] The cost of $\mathscr{A}$ on $\bar{\sigma}$, denoted $C_A^{(V;d)}(\bar{\sigma})$, is the sum of move costs, $\sum_{i=i}^{t} d(S_i, S_{i+1})$ where $d(S_i, S_{i+1})$ is the cost induced by the metric d to change from configuration S_i to configuration S_{i+1} (i.e., $d(S_i, S_{i+1}) = \min\{\sum_{u \in S_i} d(u, f(u))$: f is a function from S_i onto $S_{i+1}\}$).

[3] In the [MMS2] formulation, d need not be symmetric but since all the known results concern symmetric systems we always make this assumption.

[4] The original definition of the K-server problem also implies that $|S_i \cap S_{i+1}| \geq K - 1$ (i.e., at most one server moves). As observed in [MMS2], because of the triangle inequality, the competitive ratio does not change when we allow more than one server to move. However, when memory requirements are also considered, this greater generality is useful and we allow this generality in the definition.

Define the competitive ratio of an algorithm $\mathscr{A}$ as

$$w_C^{(V;d)}(\mathscr{A}) = \lim_{t\to\infty} \sup_{C_{\mathrm{OPT}}(\bar{\sigma})\geq t} \frac{C_{\mathscr{A}}(\bar{\sigma})}{C_{\mathrm{OPT}}(\bar{\sigma})},$$

where $C_{\mathrm{OPT}}(\bar{\sigma})$ is the optimal cost to satisfy the request sequence $\bar{\sigma}$.[5] That is, we compare the on-line and off-line performance on the same request sequence and consider the limiting behavior of the worst sequences in terms of this ratio. An algorithm $\mathscr{A}$ is called competitive if its competitive ratio is bounded by some constant. It should be noted that the optimal cost is well defined since $\bar{\sigma}$ is a finite sequence. Moreover, dynamic programming affords an obvious algorithm (see [BLS] and [MMS2]) to realize the optimal bound. However, dynamic programming is an "off-line" algorithm in that the entire sequence must be seen in order to produce the configuration (or server move) sequence. (A much more efficient but still off-line algorithm can be found in [CKPV].) An algorithm $\mathscr{A}$ is called on-line if, for all i, S_{i+1} is a function of $v_1, \ldots, v_{i+1}$. We can also define algorithms with limited lookahead and we do so by saying that $\mathscr{A}$ has lookahead l if, for all i, S_{i+1} is a function of $v_1, \ldots, v_{i+l}$. In particular then, on-line algorithms are algorithms with lookahead $l = 1$.

The concept of memory in server systems is introduced by Raghavan and Snir [RS]. They define a K-server on-line algorithm with Q memory states (i.e., $\lceil \log_2 Q \rceil$ bits of memory) as a function $u: Q \times V^K \times V \mapsto Q \times V^K$ satisfying the server property that if $u(s, \langle x_1, \ldots, x_K\rangle, z) = (s', \langle y_1, \ldots, y_K\rangle)$, then $z \in \{y_1, \ldots, y_K\}$.

This definition of memory is permissive in the sense that it allows the algorithm to store for free any information that does not vary with request sequences. Furthermore, the algorithm may use unlimited working space, as long as it resets it afresh after each move of a server.

3. The Role of memory and Lookahead in the Competitive Ratio.

In this section we wish to show that competitive K-server on-line algorithms have to use memory growing (unboundedly) with the distances of the graphs they are designed for.

As mentioned at the end of the previous section, our definition of memory is very permissive. It does not charge for any computation or data size as long as the computation can be done (and the data be written down) before seeing the request sequence. It follows that the memory we are charging for can be thought of as "learning from past experience." On the other hand, such learning should be irrelevant to the on-line/off-line problem as we are assuming the worst-case scenario where the future can be independent of the past.

This state of affairs differs from most real-life situations. In our personal lives

[5] We henceforth drop the superscript $(V; d)$ whenever it is clear from the context. We should note that an equivalent definition of the competitive ratio as defined in [BLS] and [MMS2] is given as

$$\inf\{\rho \mid \text{there exists a constant } \beta \text{ such that, for all finite request sequences } \bar{\sigma},\ C_A(\bar{\sigma}) - \rho \cdot C_{\mathrm{OPT}}(\bar{\sigma}) \leq \beta\}.$$

we often do find that "experience is the best teacher." What is the reason for this discrepancy between daily practice and what should be expected from a measure of success for on-line algorithms? The answer is twofold. First, in real-life situations there is usually a correlation between past and future. Second, in practice there is always a limitation on computational resources so that not all possible precomputation is available. We therefore view the results of this section as reflecting a counterintuitive aspect of the competitive ratio in spite of their consistency with daily practice.

We begin by considering the restricted model of *reasonable algorithms* (as defined in the conference version of Manasse *et al.* [MMS1])[6] in which a server is only allowed to move if it serves the current request. Then, in Theorem 3.2, we show how a memory bound for the general model can be easily derived from bounds in the restricted scenario. The main tool for our proof of a memory lower bound is the following simple lemma.

LEMMA 3.1. *For every competitive on-line algorithm $\mathscr{A}$ for the K-server problem on a metric space T of size $(K + 1)$, if $\mathscr{A}$ is reasonable and uses at most M memory states, then, in some sequence of $(KM + 1)$ requests, $\mathscr{A}$ will leave every node of T unoccupied for at least one point of time.*

PROOF. Let us denote by $\{t_0, \ldots, t_K\}$ the vertices of T and examine the response of the algorithm $\mathscr{A}$ to a long sequence of requests that keeps hitting the unoccupied node of T (as T has $(K + 1)$ many nodes and $\mathscr{A}$ operates only K many servers, such a node always exists).

If $\mathscr{A}$ keeps a server on some node, say t_0, for more than MK many responses to consecutive requests, then, due to its memory bound, there must be two requests—$\sigma(i)$, $\sigma(j)$—to the same node in $\{t_1, \ldots, t_K\}$, such that $\mathscr{A}$ sees both requests in the same configuration (of servers and memory state). It follows that $\mathscr{A}$ will then keep a server immobile on t_0 for any request sequence that, from that moment on, will just repeat the requests $\sigma(i), \sigma(i + 1), \ldots, \sigma(j)$. As all these requested nodes are among $\{t_1, \ldots, t_K\}$ the off-line player can fix his servers, pay nothing, and defeat any competitive ratio. This contradicts our assumption that $\mathscr{A}$ is competitive. $\square$

The following definition extracts the metric space parameters that we use to bound (from below) the necessary memory size.

DEFINITION 3.1. Let $(G; d)$ be a metric space and let T be a finite subset of G.

- Let $L(T)$ be $\max\{d(t, (T - \{t\})): t \in T\}$ (where, for a point t and a set S, $d(t, S)$ denotes the distance between t and the member of S closest to it).
- Let $l(T)$ denote the smallest distance between two (different) members of T.

[6] The journal version of Manasse *et al.* [MMS2] refers to such algorithms as lazy algorithms which should not be confused with the concept of a "lazy adversary" as defined in [RS].

- Let $S_K(G)$ denote the supremum of the ratio $L(T)/l(T)$ over all K-size subsets $T \subseteq G$.

THEOREM 3.1. *Let $(G; d)$ be a metric space over n vertices, $K < n$ and M a natural number. Every reasonable on-line algorithm for the K-server problem on G that has at most M memory states achieves a competitive ratio no better than (i.e., at least) $(2MK + 2S_{K+1}(G))/(2MK + 1)$.*

PROOF. Assume $\mathscr{A}$ is a competitive on-line K-server algorithm for G. Let $T = \{t_0, t_1, \ldots, t_K\}$ be a $(K + 1)$-size subset of G such that

$$S_{K+1}(G) = d(t_0, \{t_1, \ldots, t_K\})/d(t_1, t_2).$$

Without loss of generality we assume that in the initial configuration all the servers are occupying nodes on T (we can always force a competitive algorithm to set his servers in a given configuration by a long enough sequence of requests to these nodes only).

By Lemma 3.1, $\mathscr{A}$ is bound to vacate t_0 at least every $(MK + 1)$ many requests. Every such move costs $\mathscr{A}$ at least $2L(T)$, while the off-line player is being charged at most $l(t)$ per request (on any request sequence confined to T). Our claim now follows by a straightforward calculation (taking into account that, on a request sequence that always hits an unoccupied node, $\mathscr{A}$ is charged at least $l(T)$ on every request). □

COROLLARY 3.1. *If $(G; d)$ is such that $S_{K+1}(G)$ is infinite, a finite memory reasonable algorithm cannot be competitive for the K-server problem on G. Similarly, no finite memory on-line algorithm can achieve a bounded competitive ratio on a family of spaces with unbounded S_{K+1}.*

COROLLARY 3.2. *Any K competitive reasonable on-line algorithm for the K-server problem on a space $(G; d)$ should have at least $(2S_{K+1}(G) - K)/(K^2 - K)$ many memory states.*

THEOREM 3.2. *Let $(G; d)$ be a finite metric space of size n. Let $K < n$ and M be natural numbers. If $\mathscr{A}$ is an on-line algorithm for the K-server problem on G and M is an upper bound on the number of $\mathscr{A}$'s memory states, then the competitive ratio $w_C(\mathscr{A})$ is no better than (i.e., at least) $(2n^{K-1}MK + 2S_{K+1}(G))/(2n^{K-1}MK + 1)$.*

PROOF OF THE THEOREM. We wish to apply Theorem 3.1 but now we should consider the possibility that many servers can move on a given request. A simple way of doing this is to simulate any algorithm that moves servers at will, by an algorithm that moves a server only to service a request. We do this by using memory to recall the "virtual locations" of the original algorithm's servers. For such a simulation of a K-server algorithm with M memory states on an n size space, $\leq n^{K-1}M$ memory states are needed. The theorem now follows by substituting this memory formula in the bound of Theorem 3.1. □

How well do our bounds for memory size fit in with the memory use of known on-line algorithms? Well, for the two-server problem on a triangle our bound meets precisely the needs of the BALANCE algorithm (known also as the "Ski Rental" algorithm). This algorithm serves with the closest server to a requested node until it has incurred a cost that would have sufficed for bringing over the remote server, and at such points it does choose to serve with the remote server. To implement such an algorithm, we have to compare costs accumulated by portions of size $l(T)$ with $L(T)$, thus using at least $L(T)/l(T) = S_3(G)$ many memory states (here $G = T$ is the underlying triangle).

Chrobak *et al.* [CKPV] present a K-competitive on-line algorithm for subsets of the real line. On any request to an uncovered point, their algorithm moves the two servers adjacent to that point. (If the request is to one side of all the servers, then only the one adjacent server moves.) Both servers "move toward the request with equal speed" and stop when the first of them reaches it. Such an algorithm can be viewed as being "memoryless." The "memoryless" aspect of the Chrobak *et al.* algorithm is achieved by allowing more than one server to move and by having the ability to move servers to arbitrary points on the line.

To bridge the gap between our model and theirs, consider again the reasonable algorithm model. It is clearly possible to simulate the Chrobak *et al.* algorithm in such a model by using memory to recall "virtual locations" of servers. It is interesting to note that the upper bounds, for K-servers on the line that follows from such a simulation, match the behavior of the lower bounds on memory provided by the above results (e.g., when K is kept constant, both the bound of Corollary 3.2 and the memory needs of the simulated algorithm grow linearly in the ratio between the smallest and largest distances among the requested nodes).

On the other hand, as $S_{K+1}(G)$ can be arbitrarily high for finite subsets on the real line, our theorems can be applied to conclude that, for every $n > 1$ and $1 < K < n$, there can be no finite upper bound for the competitive ratio of a memoryless on-line algorithm that must work for every K-server system on a finite set of nodes on the line.

3.1. The Competitive Ratio of Memoryless Algorithms. As we have mentioned in the introduction the exact value of the best possible competitive algorithms is a major open question. Applying the above results we can give an almost complete answer to this question in the realm of memoryless algorithms.

DEFINITION 3.2. Let $(G; d)$ be a metric space.

- The *basic K competitive ratio* of $(G; d)$ is the best possible competitive ratio of a memoryless K-server on-line algorithm for $(G; d)$. That is,

$$E_C^K(G) = \inf\{w_C(\mathscr{A}) \colon \mathscr{A} \text{ is a reasonable memoryless } K\text{-server on-line algorithm for } G\}$$

- Let $S(G)$ denote $\sup\{(d(x, y)/d(s, t)) \colon x, y, s \neq t \in G\}$.

THEOREM 3.3. *For every metric space $(G; d)$ and every K, the basic K-competitive ratio of G satisfies $2S_{K+1}(G) \leq E_C^K(G) \leq K \cdot S(G)$.*

PROOF. The lower bound is just the bound derived by substituting $M = 0$ in the bound of Theorem 3.1. For the upper bound the algorithm ROTATE presented by Chrobak *et al.* [CKPV] can be applied. This algorithm is reasonable and memoryless but just the same, it is shown in that paper, that this algorithm is K-competitive for every K-server on any uniform space (i.e., the distance between any pair of different points equals 1). It is easy to see that, for every space $(G; d)$, if an algorithm is W-competitive for $(G; u)$, where u is the uniform metric on G, then it is $\leq W \cdot S(G; d)$ competitive for $(G; d)$. □

We have limited our discussion to deterministic algorithms. Randomization can compensate for memory constraints, but our arguments can be applied to derive similar lower bounds on the product of the number of memory bits and the number of random bits used by any competitive algorithm.

It is an interesting open problem as to whether or not for every K-server system the optimal competitive ratio can be obtained by memoryless algorithms whose servers are allowed to move a fractional distance on an edge (in the spirit of the [CKPV] line algorithm). In order to avoid trivializing this question (say by encoding arbitrarily many memory states in the least-significant digits of one server's location), we can restrict the algorithm's function $u: V^K \times V \to V^K$ so that $u(v_1, v_2, \ldots, v_k, v)$ is determined by an algebraic computation on the costs $d(u, w)$ for $u, w \in \{v_1, \ldots, v_k, v\}$.

3.2. The Role of Lookahead. The only advantage an off-line algorithm has over an on-line one is its ability to see the future. It might be expected that allowing an on-line agent access to some finite lookahead would result in improving its performance relative to that of the off-line algorithm. Contrary to this expectation, as far as the competitive ratio is concerned, no finite lookahead is sufficient for any improvement in the performance of an on-line algorithm. As a measure of efficiency of an algorithm, the competitive ratio fails to reflect an important aspect of one's everyday experience—the benefits of (finite) lookahead for decision-making tasks.

THEOREM 3.4. *For every l-lookahead on-line algorithm $\mathscr{A}$ (for any K-server system) there exists a fully on-line (i.e., with lookahead $l = 1$) algorithm $\mathscr{B}$ achieving exactly the same competitive ratio.*

PROOF. Let $\mathscr{B}$ react to any sequence of requests $\sigma = (x_1, x_2, \ldots, x_t)$ by simulating $\mathscr{A}$'s behavior on $\sigma^l = (\underbrace{x_1 \cdots x_1}_{l}\, \underbrace{x_2 \cdots x_2}_{l} \cdots \underbrace{x_t \cdots x_t}_{l})$. It is straightforward to note that $C_{\mathscr{B}}(\sigma)/C_{\mathrm{OPT}}(\sigma) = C_{\mathscr{A}}(\sigma^l)/C_{\mathrm{OPT}}(\sigma^l)$. It now follows from the definition of the competitive ratio that $w_C(\mathscr{A}) \geq w_C(\mathscr{B})$. (The reverse inequality is trivially true.) □

It should be noted that this simple argument remains valid when the on-line algorithm is allowed to chose its responses with the help of a random source (and the cost of serving a sequence is defined as the average cost over the bits generated by that source).

4. Definition and Basic Properties of the Max/Max Ratio.

We now introduce a new ratio which, in some sense, seems closer to the spirit of traditional worst-case complexity analysis. The new measure we offer here gives rise to a different priority relation between on-line algorithms. The Max/Max ratio supplies a mathematical model for preferring to buy an insurance policy, a model in which the benefits of lookahead are evident and there exist (almost) optimal memoryless algorithms. The new measure clearly fails to capture locality of reference; in particular, for the paging problem all K-server on-line algorithms have the same Max/Max ratio. It should be noted though that, contrary to common belief, the competitive ratio suffers from a similar problem—it does not necessarily prefer algorithms employing locality of reference. Chrobak *et al.* [CKPV] present an on-line algorithm for paging—ROTATE—that is absolutely memoryless (and therefore cannot take into account any such consideration), but still achieves the best possible competitive ratio—K.

Thus we are not trying to argue in favor of a particular measure. Rather we are simply claiming that different measures reflect different concerns.

We first define the amortized cost of an algorithm $\mathscr{A}$ as:

DEFINITION 4.1. $M_t(\mathscr{A}) = \max_{|\bar{\sigma}|=t} \mathscr{C}_{\mathscr{A}}(\bar{\sigma})/t$. The amortized cost of an algorithm $\mathscr{A}$ is defined as $M(\mathscr{A}) = \limsup_{t\to\infty} M_t(\mathscr{A})$.

This is a natural adaptation of the notion of amortized complexity to our context. However, in order to make this concept well defined, we hereafter assume that our server systems are *bounded* in the sense that, for some $\mathscr{B} < \infty$, $d(u, v) \le \mathscr{B}$ for all nodes u, v. (Alternatively, we could normalize $M_t(\mathscr{A})$ by the diameter covered by the requested nodes.) Also for definiteness we assume that both the on-line and off-line algorithms start in the same specified initial configuration.

The following technical lemma helps to simplify the presentation.

LEMMA 4.1. *For every bounded K-server system and any optimal off-line algorithm OPT for serving it, the sequence $M_t(OPT)$ converges to a (finite) limit as t goes to infinity (thus $M(OPT) = \lim_{t\to\infty} M_t(OPT)$).*

We are now in a position to introduce the main definition of this paper—the Max/Max ratio of an algorithm:

DEFINITION 4.2. Let $\mathscr{A}$ be an algorithm for serving requests on some K-server system, the Max/Max ratio of $\mathscr{A}$ denoted $w_M(\mathscr{A})$ is $\limsup_{t\to\infty}(M_t(\mathscr{A})/M_t(OPT))$.

It immediately follows from Lemma 1 that:

LEMMA 4.2. $w_M(\mathscr{A}) = \limsup_{t\to\infty}(M_t(\mathscr{A})/M_t(OPT)) = M(\mathscr{A})/M(OPT)$.

That is, we compare the worst-case behavior of A and OPT rather than their performance on the same sequence. Our Max/Max ratio is a normalized amortized complexity measure, where here we normalize by the best that can be done using the optimal off-line algorithm.

The entity abstracted by the competitive ratio is quite different; in many cases minimizing the amortized cost is in sharp conflict with minimzing the competitive ratio, as illustrated by the following problematic concern:

Let us consider a two-server, three-node system as a very simplified abstraction for studying the purchase of insurance (say on a car). Let $d(1, 2) = p$ and $d(1, 3) = d(2, 3) = c$ for some c much larger than p. We can think of p as an annual premium and c as the cost of replacement. The presence (resp. absence) of a server on node 3 will correspond to the state of being insured (resp. not being insured). To obtain a bounded competitive ratio (say by using the BAL algorithm of Manasse *et al.* [MMS1]) we must vacate node 3 whenever the premium costs paid thus far equals the replacement cost. This decision takes place no matter how many times node 3 has been requested (i.e., how many replacements have taken place)! On the other hand, minimizing the Max/Max ratio forces us to buy insurance every year as long as $p < c$. Clearly, the different measures determine very different strategies! *In practice the measure (or combination of measures) has to be chosen that best reflects the consideration(s) being emphasized.*

We would like to point out a few more properties of the Max/Max measure.

COROLLARY 4.1. *Let $\mathscr{A}$, $\mathscr{B}$ be algorithms for servicing the same finite K-server system, then*

$$\frac{M(\mathscr{A})}{M(\mathscr{B})} = \frac{w_M(\mathscr{A})}{w_M(\mathscr{B})}.$$

PROOF. This is a straightforward consequence of Lemma 2. □

Corollary 4.1 indicates that for a given space $(V; d)$ the Max/Max measure orders all servicing algorithms. This property enables a complete analysis of the relative (Max/Max) efficiency of two on-line algorithms without referring to an optimal off-line algorithm (and consequently without having to analyze it). We make use of this convenience in Section 4 where we present an on-line algorithm which is optimal up to a factor of 2 amongst all on-line algorithms for any bounded K-server system.

One more possible use of this comparability property is that it gives an exact measure for the improvement in performance gained by increasing the number of servers; it is not clear how best to address this issue relative to the competitive ratio. (For one interesting approach, see Young [Y].)

4.1. The Max/Max Ratio Is Graph-Dependent. Given the [MMS1] competitive ratio lower bound of K for every K-server system, and their K-server conjecture which states that this is optimal for every K-server system, it may very well be that the competitive ratio is independent of the underlying system. The situation is quite different when we consider the Max/Max ratio. In Section 6 we see that for uniform $K + 1$ node space the Max/Max ratio for K-servers is exactly K. We now present a class of metric spaces for which this ratio is 1.

DEFINITION 4.3. A space $(G; d)$ is a K-cluster if it can be partitioned into K subsets $G_1, \dots, G_K$ so that:

(i) For all i, there are at least two nodes in G_i.
(ii) For all $i \neq j$, if $x \in G_i$, $y \in G_i$, $z \in G_j$, then $d(x, z)/d(x, y) > K$.
(iii) For all i, j, diameter (G_i) = diameter (G_j).

FACT 1. *If G is a K-cluster, then an on-line K-server algorithm exist that achieves a Max/Max ratio of 1 on G.*

PROOF. Let the on-line algorithm devote a fixed server to every cluster. The amortized cost per request of such an algorithm is at most the diameter of the G_i's. On the other hand, if $v_1 \cdots v_K$, $u_1 \cdots u_K$ are nodes such that, $\forall i$, $1 \leq i \leq K$, diameter $(G_i) = d(v_i, u_i)$, then the request sequence $(v_1 \cdots v_K u_1 \cdots u_K)^*$ forces any off-line algorithm to pay $d(v_i u_i)$ per request. □

"Cost Graphs" are a simple tool for visualizing the behavior of servicing algorithms and for comparing different measures. A cost graph displays the cost that an algorithm incurs as a function of the task sequence it has to serve as follows: Given a K-server system we fix a natural number t and consider the set of all request sequences of length t. We present the request sequences in an arbitrary order along the x-axis and use the y-axis for the cost.

FACT 2. *Given any task system and any algorithm $\mathscr{A}$ for servicing it, $w_M(\mathscr{A}) \leq w_C(\mathscr{A})$.*

PROOF. This fact becomes self-evident when cost graphs are viewed. Given a task system, for every s and every positive ρ the cost graphs of all algorithms that are within competitive ratio ρ of the optimal off-line lie in the strip between the graph of $C_{\mathrm{OPT}}(\bar{\sigma})$ and $\rho C_{\mathrm{OPT}}(\bar{\sigma})$, whereas any algorithm whose graph is bounded by $\rho \cdot \max_{|\bar{\sigma}|=t} C_{\mathrm{OPT}}(\bar{\sigma})$ is within a Max/Max factor $\leq \rho$. □

4.2. Memoryless Max/Max Optimal Algorithms. The first issue on which we wish to compare the Max/Max ratio with the competitive measure is the dependence of good algorithms upon memory. We have seen in Section 3 that competitive on-line algorithms necessarily depend upon memory. We wish to show that, once the Max/Max ratio is used for measuring the efficiency of on-line algorithms, memory use becomes irrelevant. Unfortunately we are unable, at this stage, to prove the ultimate result along this line—i.e., that, for any K and any metric space G, a memoryless on-line algorithm exists that achieves the optimal Max/Max ratio. In Section 5 we do prove a result that approximates the above statement. That is, we provide a general purpose memoryless on-line algorithm that, for any K and any bounded metric space G, achieves a Max/Max ratio that is within a factor of 2 of the best possible on-line algorithm (for this K and G).

5. A Good Algorithm.

Considering the previous discussion, we can expect that a relatively simple algorithm based on "staying in one's territory" will yield a good bound on the Max/Max ratio. This is indeed the case.

DEFINITION 5.1. Given a bounded metric space $G = (V; d)$ and $K \in \mathbf{N}$, the *K-covering radius* of G is defined as

$$R(K, G) = \inf\{r \mid \text{there exist a "set of centers" } X = \{v_1, \ldots, v_K\} \subseteq V \text{ such that } d(u, X) \leq r \text{ for all } u \in V\}.$$

(Recall that $d(u, X)$ denotes $\inf_{v \in X} d(u, v)$.)

LEMMA 5.1. *Let $R(K, G) = r$. Then for every $\varepsilon > 0$ there exist $u_1, \ldots, u_{K+1}$ in V such that $d(u_i, u_j) > r - \varepsilon$ for all $i \neq j$.*

PROOF. Suppose that for every set of $K + 1$ distinct points $u_1, \ldots, u_{K+1}$ there is a pair of points u_i, u_j with $d(u_i, u_j) \leq r - \varepsilon$. Then we can define a K-set $X = \{v_1, \ldots, v_K\}$ such that the covering radius (by using these points) is $\leq r - \varepsilon$ as follows: Choose v_1 arbitrarily. Given $X = \{v_1, \ldots, v_l\}$ with $d(v_i, v_j) > r - \varepsilon$ for all $i \neq j$, if a $v \in V$ exists such that $d(v, X) \geq r - \varepsilon$, then set $X = X \cup \{v\}$. Otherwise the process ends. By assumption the process must end with $\# X \leq K$ and X is an appropriate set of centers. $\square$

Consider the following *K-Center* algorithm for any K-server problem on a bounded metric space $(V; d)$. Let the covering radius be R and let $X = \{v_1, \ldots, v_K\}$ be a set of centers such that all nodes in V are within R of X. Place the ith server on v_i. To serve a request, choose any server subject to the constraint that the ith server remains within distance R of v_i.

THEOREM 5.1. *The above K-Center algorithm achieves a Max/Max ratio $\leq 2K$ for the K-server problem on any bounded metric space.*

PROOF. Clearly the algorithm never pays more than $2R$ to serve a request. For every $\varepsilon > 0$ let $u_1, \ldots, u_{k+1}$ be a set of pairwise $(R - \varepsilon)$ remote nodes whose existence is established by Lemma 5.1. Consider any (off-line) algorithm on the repeating request sequence $(u_1, \ldots, u_{K+1})^*$. Clearly, for every $\varepsilon > 0$, even the optimal algorithm must pay $R - \varepsilon$ every K requests (see the discussion concerning the uniform system in the next section), so that, for all $\varepsilon > 0$, $w_M(K\text{-}Center) \leq 2R/((R - \varepsilon)/K)$. As we can choose ε arbitrarily small, we can conclude $w_M(K\text{-}Center) \leq 2K$. □

COROLLARY 5.1. *The K-Center algorithm is optimal up to a factor of* 2 *amongst all on-line algorithms.*

PROOF. For every $\varepsilon > 0$, let $u_1, \dots, u_{k+1}$ be a set of pairwise $(R - \varepsilon)$ remote nodes as in the theorem. By always requesting an unoccupied node from this set every on-line algorithm can clearly be forced to pay at least $R - \varepsilon$ in every request. □

Hochbaum and Shmoys [HS] show that the problem of computing an optimal K-set of centers for a finite metric space is an NP-hard problem. In [HS] it is shown that an efficient approximation algorithm does exist for obtaining a K-set with radius $r' \leq 2r$ where r is the optimal radius, but obtaining an approximation within a factor of $(2 - \varepsilon)$ remains an NP-hard problem. If the on-line algorithm uses this approximation, then clearly w_M(approximate *K-Center*) $\leq 4K$.

Finally, one might ask about the performance of perhaps the "most basic" algorithm, namely, serving with the closest server. It is easy to see that if the K-servers are configured so as to make the covering radius of the remaining nodes large (e.g., by having one server for two remote nodes), then "serving with the closest server" will not correct itself. Even if the servers are initially placed on an optimal set of centers, it is still possible to drag serves away from their territory.

CLAIM. *There is a finite metric space such that, starting from any initial configuration, the Max/Max ratio of the Serve-with-Closest on-line algorithm on this metric space is exponential in K.*

PROOF. Let b_i be the ith element in the Fibonacci sequence and let $a'_1 = 0$ and $a'_{i+1} = a'_i + b_i$. Let our space be a set of reals $\{a_1, \dots, a_{2K}\}$ where, for all i satisfying $3 \leq i \leq 2K - 1$, we have $a_{i+1} - a_i < a_i - a_{i-2}$ and the a_i's are arbitrarily close to the a'_i's (i.e., given any ε choose the a_i's so as to make sure that $\sum_{i=1}^{2K} |a'_i - a_i| \leq \varepsilon$). The metric on the space is naturally defined by the distances between the points on the real line.

We wish to show that, starting from any initial configuration, we can force a *Serve-with-Closest* player to set its K-servers on the odd indexed points.

We do this in K steps. At the beginning of stage i $(i \geq 1)$, we have the property that for each $j < i$ there is exactly one server on node a_{2j-1} and that all of the remaining $K - i + 1$ servers are on the nodes $\{a_j | j \geq 2i - 1\}$. Stage i proceeds as follows: Until there is at most one server on the two nodes $\{a_{2i-1}, a_{2i}\}$, we continue to move servers toward a_{2K} by requesting any node a_j $(j \geq 2i)$ such that a_j is unoccupied but a_{j-1} is occupied by a server. If the total number of servers on $\{a_{2i-1}, a_{2i}\}$ is greater than one, then such an a_j must exist. If there is exactly one server on the nodes $\{a_{2i-1}, a_{2i}\}$, then request a_{2i-1} to complete stage i.

Otherwise we bring one server toward a_{2i-1} by requesting the lowest a_j such that a_j is unoccupied but a_{j+1} is occupied by a server. The stage ends when a_{2i-1} gets requested.

Having so forced the servers to the odd indexed points, the adversary requests a_{2K-1} and a_{2K} alternatively, so that the *Serve-with-Closest* algorithm is forced to pay $b_{2K-1} \approx \varphi^{2K-2}$ per request, where $\varphi = (1 + \sqrt{5})/2$.

On the other hand, as $\sum_{i=1}^{K-1} b_i < b_{K-1} + b_K$, the *K-Center* algorithm will station its servers on the upper half of the set of vertices and serve all requests to vacant nodes by the leftmost stationed server. It is easy to check that the cost of serving any request is thus $O(\varphi^K)$. □

6. The Influence of Lookahead. Unlike the situation for the competitive ratio, we now show that, when considering the Max/Max ratio, lookahead can improve on-line performance. In particular, we can completely analyze the power of lookahead with respect to any uniform server system (i.e., paging).

LEMMA 6.1. *Let U_n denote the uniform n-node space (i.e., $d(i,j) = 1$ for all $i \neq j$). Then, for all $K \leq n - 1$,*

$$M(OPT) = \frac{n-K}{n-1}.$$

PROOF. If we consider the uniform server system U_n, then the off-line player can play a simple greedy strategy, namely, to postpone paging as long as possible. That is, it evicts the page whose next request is farthest in the future. (This strategy was proposed by Belady [B] and proven optimal by Mattison *et al.* [MGST]. An elegant proof can be found in [MS].) We assume the initial configuration has servers on nodes $x_{n-K+1}, \ldots, x_n$ and consider the request sequence $x_1, \ldots, x_n$, $x_1, \ldots, x_n, \ldots$. In trying to postpone moving as much as possible, at time t the off-line player will serve an uncovered node with that server whose present location will be the last to be requested. In doing so, it is easy to see that on the given input sequence the off-line player then moves $n - K$ times for every $n - 1$ requests. Conversely, by looking ahead at the next $n - 1$ requests in any request sequence, OPT can guarantee that at most $n - 1 - (k - 1) = n - k$ nodes will be unoccupied when requested. □

We now show that an on-line player with lookahead $l = n - 1$ can achieve the same max cost by essentially following the same greedy strategy. Indeed, for any l satisfying $n - K < l \leq n - 1$, the on-line player can use the greedy strategy to achieve a max ratio of $(n - 1)/l$.

THEOREM 6.1. *Consider the K-server problem on U_n, with lookahead l, $1 \leq l \leq n - 1$.*

(i) *If $l \leq n - K$, then the greedy strategy achieves a Max/Max ratio* $= (n - 1)/(n - K)$ *and this is optimal. (That is, any $l \leq n - K$ yields the same Max/Max rato as for* $l = 1$.)
(ii) *If $n - K < l \leq n - 1$, then the greedy strategy achieves a Max/Max ratio of $(n - 1)/l$ and this is optimal. (Thus, in particular, the Max/Max ratio* $= 1$ *for* $l = n - 1$.)

PROOF. We first consider the case $l \leq n - K$ so without loss of generality let $l = n - K$. At every point in time, there are $n - K$ uncovered nodes. The adversary constructs the request sequence by maintaining the property that the next $n - K$ requests are distinct and different from the K covered nodes. To do so, the adversary makes the $(n - K)$th next request to be a node that is neither covered now nor is it requested in the next $l - 1$ requests foreseen by the on-line algorithm.

Clearly, such a strategy forces the on-line player to move a server on every request.

Now we consider the case $n - K < l \leq n - 1$. The on-line algorithm l-GREEDY is as follows:

> Lookahead to the next l requests and use the greedy strategy described above (in Lemma 6.1's proof), relative to the lookahead available to you. That is, if there are any presently occupied nodes which are not requested in the next l requests, then use any server sitting on such a node. Otherwise use that server whose location is the last to be requested for the first time amonst the next l requests.

We will show $M(l\text{-GREEDY}) = (n - K)/l$. Combining this with Lemma 6.1 we have

$$w_M^{U_n}(l\text{-GREEDY}) = \frac{n - K/l}{n - K/n - 1} = \frac{n - 1}{l}.$$

Let us prove that, in any sequence of l requests, the on-line algorithm pays at most $n - K$.

Let $\sigma_1\sigma_2 \cdots \sigma_m$ be a sequence of requests and let C be a configuration of servers. We say *σ_i is a repeat point in $\sigma_1\sigma_2 \cdots \sigma_m$ relative to C* if either σ_i is a location occupied in C or there exists $j < i$ such that $\sigma_j = \sigma_i$.

CLAIM 1. *If $t < t'$, and C_t and $C_{t'}$ represent the configurations (before satisfying the request) at times t and t' respectively, then any request at time $t'' > t'$ to a node v in $C_{t'}$ is a repeat point with respect to C_t.*

PROOF. If $v = \sigma_{\bar{t}}$ is in C_t we are done. Else, v must have been requested before time t' (else $v \notin C_{t'}$). Thus $\sigma_{\bar{t}}$ is a repeat point relative to C_t. □

CLAIM 2. *For every C and every $\sigma_1\sigma_2 \cdots \sigma_l$, at least $l - (n - K)$ repeat points relative to C exist.*

PROOF. Claim 2 is immediate since there are only $n - K$ uncovered nodes. □

CLAIM 3. *Suppose l-GREEDY vacates a node v in configuration C_t. Then either v is not requested in the next l steps or before the first request to v there are at least $K - 1$ requests to members of C_t.*

PROOF. If v is vacated even though it is to be requested within the next l requests, then, by definition of l-GREEDY, there must have been at least $K - 1$ requests, one to each of the nodes in $C_t - \{v\}$.

CLAIM 4. *Relative to any C_1 the first $K - 1$ repeat points (or as many that exist) in $\sigma_1\sigma_2\cdots\sigma_l$ do not cost l-GREEDY.*

PROOF. Let σ_s be the ith repeat point for any $1 \leq i \leq K - 1$; i.e., σ_s occurs with the algorithm in configuration C_s.

If σ_s is in C_1 and is not vacated in the processing of $\sigma_1, \ldots, \sigma_{s-1}$, then clearly σ_s does not cost l-GREEDY. If σ_s is not in C_1, then it must have been requested at some time t': $1 \leq t' < s$. Let t' be the last such request. Clearly, $\sigma_s \in C_{t'+1}$ and if it is not vacated in the processing of $\sigma_{t'+1}, \ldots, \sigma_s$, then again σ_s does not cost the algorithm. □

Returning to the proof of the theorem, suppose that $C_{\bar{t}}$ is the last configuration from which σ_s was vacated. Then by Claim 3 there must be at least $K - 1$ requests in $\sigma_{\bar{t}}, \ldots, \sigma_{s-1}$ to members of $C_{\bar{t}}$ and, by Claim 1, each of these requests is a repeat point with respect to C_1. Hence there are $K - 1$ repeat points relative to C_1 in $\sigma_{\bar{t}}\sigma_2\cdots\sigma_{s-1}$ contradicting the assumption that σ_s is the $i \leq K - 1$ repeat point relative to C_1.

The on-line player then pays at most $n - K$ in any l steps since the first $l - (n - K)$ repeat points are free. Thus the

$$\text{Max/Max ratio} \leq \frac{(n - K)/l}{(n - K)/(n - 1)} = \frac{n - 1}{l}.$$

Finally let us show a lower bound of $(n - 1)/l$ on the Max/Max ratio of any on-line algorithm with lookahead l in the range $[n - K, n - 1]$.

CLAIM 5. *For any $n - K \leq l \leq n - 1$ and for any on-line algorithm $\mathscr{A}$ with lookahead l, a request sequence on n nodes exists that forces the on-line player to move a server at least $n - K$ many times on every subsequence of l many consecutive requests.*

PROOF. The claim follows once we can demonstrate the existence of a request sequence $\sigma_{\mathscr{A}}$ such that, for every i, the subsequence $\sigma(i + 1), \sigma(i + 2), \ldots, \sigma(i + l)$

contains at least $n - K$ nodes that are not occupied by servers after $\mathscr{A}$'s response to the ith request of σ. Such a sequence can be easily generated by letting $\sigma(1), \ldots, \sigma(l)$ contain $n - K$ nodes not occupied by the servers in their initial configuration (this is possible whenever $n - K \leq l$), and then, at each stage $i > l$, chose as $\sigma(i + 1)$ the last node vacated by $\mathscr{A}$ before it serves $\sigma(i - l + 1)$. □

This completes the proof of Theorem 6.1. □

The previous theorem raises an important question. Namely, is it the case that, for every bounded K-server problem, indeed for every bounded task system problem, an l exists such that there is an l-lookahead algorithm achieving a Max/Max ratio *equal* to 1? That is, can amortized optimality always be achieved with finite lookahead? The following example shows that finite lookahead cannot always guarantee optimality.

CLAIM. *There is a K-server system for which no amount of finite lookahead can guarantee Max/Max optimality. Consider three servers with lookahead l on a "barbell" graph with nodes $\{a, b, c, d\}$ and edge costs $d(a, b) = d(c, d) = 1$ and $d(a, c) = d(a, d) = d(b, c) = d(b, d) = 4$. The initial configuration is $\{a, b, c\}$. The sequence of requests is generated in segments. The prefix $(abcd)^l$ begins each segment. If, before the first request of the segment, a or b is vacant, ab is added to end the segment, otherwise c or d is vacant and cd is added to end the segment. Segments are repeatedly generated this way. The on-line algorithm incurs a cost of at least 2 per (abcd) piece and an additional 2 for the two requests at the end of each segment, for an average cost of $(2l + 2)/(4l + 2)$ per request. For any sequence of n requests, OPT can incur an average cost per request bounded by $\frac{1}{2} + 4/n$ by immediately moving a server so that both nodes on the side of the barbell that will be requested most are covered, and then only moving the server on the other side of the barbell back and forth.*

Although finite lookahead cannot always guarantee optimality, it is the case that increasing the lookahead can improve the Max/Max ratio. As observed before, the usual optimal (off-line) algorithm is dynamic programming which needs to see the entire request sequence. The obvious approach then would be to use an l-lookahead approximation to dynamic programming, which indeed as l grows gives a better approximation to the optimal dynamic programming algorithm. We have the following:

THEOREM 6.2. *For every bounded K-server system, $\forall \varepsilon > 0$, $\exists l$ and an l-lookahead algorithm $DP(l)$ such that the Max/Max ratio of $DP(l) \leq (1 + \varepsilon)$.*

PROOF. Let $R = R(K, G)$ be the K-covering radius of G, so that by the argument of Theorem 5 the off-line player pays at least R/K per request on a worst-case sequence. The on-line algorithm $DP(l)$ simply looks ahead l requests and performs, as does the dynamic programming solution OPT for the sequence of l requests $\sigma_1 \sigma_2 \cdots \sigma_l$, say, ending in configuration C. Then $DP(l)$ looks ahead at the next l

requests $\sigma_{l+1}\sigma_{l+2}\sigma_{2l}$ and now simulates the behavior of OPT on the entire sequence of $2l$ requests $\sigma_1\sigma_2\cdots\sigma_{2l}$. If OPT would be in configuration C' after $\sigma_1\sigma_2\cdots\sigma_l$, then, in processing $\sigma_{l+1}\cdots\sigma_{2l}$, the cost to $DP(l)$ will be at most $d(C, C') + OPT$'s cost on $\sigma_{l+1}\cdots\sigma_{2l}$. Continuing in this manner, it follows that if $l = m\cdot\max_{(C,C')} d(C, C')/(R(K, G)/K)$, then $C_{DP(l)}(\bar{\sigma}) \leq (1 + 1/m)C_{\mathrm{OPT}}(\bar{\sigma})$. □

The argument above produces an l that depends upon both K and the metric space. We have already seen that in the paging scenario the amount of lookahead needed for approximating an off-line performance grows unboundedly with n (and therefore with K).

Claim. *For a two-server problem on a triangle G, any lookahead less than $2S_3(G)$ does not improve the Max/Max ratio of an on-line algorithm. (It should be noted, though, that such lookahead can help an on-line algorithm reduce its cost on* some *request sequences; e.g., sequences with blocks of length $\leq l$ on which the requests are constrained to a subset of size $\leq K$ of the nodes). The claim can be easily extended to any K-server problem.*

Proof. The idea is similar to that of Theorem 3.1.

Let (v_1v_2) be the smallest edge of G. Let d denote $d(v_1v_2)$ and let D be the length of the medium-sized edge of G (so $S_3(G) = D/d$). An on-line algorithm can choose to leave a server at v_3 and thus bound its amortized cost by d per request.

Let A be any l-lookahead on-line algorithm. If $l < 2S_3(G)$, then, being faced with a sequence of l many requests alternating between v_1 and v_2, A should keep a server unmoved on v_3. Having only l-lookahead there is the risk that the $(l + 1)$st next request may be to v_3, and as $d\cdot l < 2D$ this is the best way to serve $\underbrace{v_1v_2v_1\cdots v_2}_{l}v_3v_1$. It follows that A's amortized cost $\geq d$, just as could be achieved without lookahead. □

Conjecture. *We conclude with our own version of a K-Server Conjecture:*

For every (bounded) K-server system, the Max/Max ratio $\leq K$; furthermore, this ratio can be achieved by a memoryless algorithm.

Acknowledgments. We are greatly indebted to the referees who not only provided many constructive suggestions for improving the presentation but also found errors in some of our original claims and conjectures. In particular, we thank one of the referees for the example contained in the Claim preceding Theorem 6.2.

References

[B] L. A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems J.*, 5:78–101, 1966.

[BLS] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. *Proc. 19th Annual ACM Symposium on Theory of Computing*, New York, May 1987, pp. 373–382. Also in *J. Assoc. Comput. Mach.*, 39:745–763, 1992.

[CKPV] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *Proc. 1st Annual ACM–SIAM Symposium on Discrete Algorithms*, San Francisco, CA, 1990.

[FRR] A. Fiat, Y. Rabani, and Y. Ravid. Competitive K server algorithms. *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science*, St. Louis, MO, Oct. 1990, pp. 454–463.

[G] E. Grove. The harmonic k-server algorithm is competive. *Proc. 23rd Annual ACM Symposium on Theory of Computing*, May 1991.

[HS] D. S. Hochbaum and D. B. Shmoys. Powers of graphs: a powerful approximation technique for bottleneck problems. *Proc. ACM Symposium on Theory of Computing*, 1984, pp. 324–333.

[KMRS] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[MGST] R. L. Mattison, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems J.*, 9(2):78–117, 1970.

[MMS1] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. *Proc. 20th Annual ACM Symposium on Theory of Computing*, Chicago, ILl, May 1988, pp. 322–333.

[MMS2] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. *Journal of Algorithms*, 11:208–230, 1990.

[MS] L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. Technical Report CMU-CS89–122, School of Computer Science, Carnegie Mellon University, 1989. Also in *Algorithmica*, 6(6):816–825, 1991.

[RS] P. Raghavan and M. Snir. Memory vs. randomization in on-line algorithms. *Proc. 16th ICALP*, Italy, July 1989, pp. 687–703. LNCS 372, Springer-Verlag, Berlin, 1990.

[ST] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, 1985.

[Y] N. Young. The k-server dual and loose competitiveness for paging. To appear in *Algorithmica.*