



Available at

www.ElsevierComputerScience.com

POWERED BY SCIENCE @ DIRECT®

Information Processing Letters 90 (2004) 47–52

Information
Processing
Letters

www.elsevier.com/locate/ipl

Analysis of simple randomized buffer management for parallel I/O[☆]

Mahesh Kallahalla^{a,*}, Peter J. Varman^b

^a Hewlett-Packard Labs, 1501 Page Mill Rd., Palo Alto, CA 94304, USA

^b Department of ECE, Rice University, Houston, TX 77005, USA

Received 5 October 2000; received in revised form 6 June 2001

Communicated by S.E. Hambrusch

Abstract

Buffer management for a D -disk parallel I/O system is considered in the context of randomized placement of data on the disks. A simple prefetching and caching algorithm PHASE-LRU using bounded lookahead is described and analyzed. It is shown that PHASE-LRU performs an expected number of I/Os that is within a factor $\Theta(\log D / \log \log D)$ of the number performed by an optimal off-line algorithm. In contrast, any deterministic buffer management algorithm with the same amount of lookahead must do at least $\Omega(\sqrt{D})$ times the number of I/Os of the optimal.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Parallel I/O; Prefetching; Caching; Buffer management; Analysis of algorithms; Randomization; Competitive ratio

1. Introduction

Multiple-disk parallel storage systems [9] are being increasingly deployed in high-performance servers to meet the ever-increasing demands for I/O performance. However it remains a challenging problem to effectively use the increased disk bandwidth to reduce application I/O latency. Effective use of I/O par-

allelism requires careful coordination between data placement, prefetching and caching policies.

In this paper we describe a simple buffer management algorithm, PHASE-LRU, and analyze its performance using the Parallel Disk Model [19]. In this model, the I/O system consists of D independently-accessible disks and an M -block shared I/O buffer. The data for the computation is stored on the disks in blocks. In each parallel I/O up to D blocks, at most one from each disk, can be accessed. From the viewpoint of the I/O, the computation is characterized by a *reference string* consisting of the ordered sequence of block references that are made by the computation. Serving the reference string entails performing I/O operations needed to provide the computation with blocks in the order specified by the reference string. Performance is

[☆] Supported in part by the National Science Foundation under grants CCR-9704562 and CCR-0105565, and a grant from the Schlumberger Foundation.

* Corresponding author.

E-mail addresses: maheshk@hpl.hp.com (M. Kallahalla), pjv@rice.edu (P.J. Varman).

¹ This work was done while the author was at the Department of ECE, Rice University.

based on the number of parallel I/Os required to service a given reference string.

Prefetching and caching are fundamental techniques used to improve parallel I/O performance. Prefetching refers to the process of initiating a read from a disk before the computation demands the data. While a read progresses on one disk, reads can be started concurrently on the other disks to prefetch data that are required later in the computation. These prefetched blocks are held in the I/O buffer until needed. Caching refers to the process of retaining previously referenced blocks in the buffer in anticipation of later reuse.

In order to prefetch accurately, the prefetching mechanism must have access to a portion of the reference string beyond the current reference. This is embodied in the notion of lookahead. Considerable attention has been devoted to the problem of obtaining such lookahead information from applications using combinations of programmer hints and program analysis [15]. The lookahead information can also be used by the caching mechanism in deciding which blocks to evict from the buffer. PHASE-LRU uses M -block lookahead, defined precisely in Section 1.2. Intuitively, M -block lookahead provides the next bufferload of distinct requests to the buffer management algorithm.

For a uniformly random assignment of data blocks to disks, the expected number of I/Os performed by PHASE-LRU is shown to be within a factor $\Theta(\log D / \log \log D)$ of the number of I/Os done by the optimal off-line algorithm. In contrast *any* scheduling algorithm with the same lookahead, that uses only deterministic rather than randomized placement is known to have a competitive ratio of $\Omega(\sqrt{D})$ [5].

In a sequential I/O model, a winning caching strategy is to always evict the block whose next reference is the farthest. This is the policy used in the optimal single-disk buffer management algorithm MIN [6]. In a parallel I/O system however, this policy can be ineffectual since the performance depends both on the number of blocks accessed and the parallelism with which they are fetched. Consider an example with $D = 3$ and $M = 6$, where blocks a_i (respectively b_i, c_i) are placed on disk 1 (respectively 2, 3). Suppose that at some point the buffer contains $a_1, a_2, a_3, b_1, b_2, c_1$, and the remainder of the reference string consists of the subsequence:

Disk 1	a_4	a_1	a_2	a_3
Disk 2	b_3			
Disk 3	c_2			

(a)

Disk 1	a_4	a_1
Disk 2	b_3	b_1
Disk 3	c_2	c_1

(b)

Fig. 1. Schedules for reference string Σ^* .

$\Sigma^* = a_4 b_3 c_2 a_4 b_3 b_2 b_1 c_1 a_1 a_2 a_3$. Fig. 1(a) shows the I/O schedule obtained by using MIN's eviction policy. To fetch a_4 MIN evicts a_3 . Blocks b_3 and c_2 are prefetched along with a_4 , evicting blocks a_2 and a_1 . Three more I/Os, fetching a_1, a_2 and a_3 respectively, are required to complete the schedule. In contrast Fig. 1(b) shows a schedule that takes only 2 rather than 4 steps. This is obtained by evicting blocks a_1, b_1 and c_1 at the first step. When the computation references b_1 these blocks are read back in just one parallel I/O.

1.1. Previous work

Classical buffer management has been studied extensively in a sequential I/O model [2,6,7,10,14,17]. These works primarily deal with developing efficient buffer management algorithms for a single-disk system, by optimizing decisions regarding the blocks to be evicted from the buffer. The use of information regarding future accesses, *lookahead*, to improve the eviction decisions made by on-line algorithms for single-disk systems was studied in [2] and [7], using different models of lookahead. The notion of lookahead used in this paper closely resembles the notion of strong lookahead introduced in [2]. In L -block strong lookahead, the lookahead window is the largest subsequence of the reference string that includes L distinct references beyond the current reference. The model in [7] appears harder to realize in practice, requiring knowledge of the next L distinct references that are not currently in the buffer.

The overlap of processor and I/O operations in a stall model of computation for a single-disk system was addressed in [8]. Off-line approximation algorithms were presented and analyzed using the relative I/O and processing speeds as parameters. Later, in [3] optimal scheduling algorithms for this model based on integer programming were shown.

In [5] the question of designing on-line prefetching algorithms for read-once reference strings was

addressed, and fundamental bounds were presented. However, the problem of general reference strings in which blocks can be repeatedly accessed was not considered. For read-once reference strings and using M -block lookahead, matching bounds of $\Theta(\sqrt{D})$ on the competitive ratio of prefetching algorithms were derived. Prefetching algorithms for read-once reference strings using multiple disks and randomized data placements were analyzed in [1,4,5,12,16]. Note that caching is not useful in read-once reference strings since there is no reuse of blocks.

In [13] the single-disk stall model of [8] was generalized to incorporate parallel I/O, and an approximate off-line buffer-management algorithm was presented. An optimal off-line buffer management and scheduling algorithm was presented in [18] for a distributed-buffer parallel I/O model in which each disk has its own private buffer.

1.2. Definitions

We use the Parallel Disk Model [19] consisting of D independent disks and an M -block shared I/O buffer, $M \geq D$. The *reference string* $\Sigma = r_1 r_2 \cdots r_n$, is the ordered sequence of block references made by the computation. We use $\text{block}(r_i)$ to refer to the block referenced by r_i .

Definition 1. The buffer management algorithm has *M -block lookahead* if immediately after r_i is referenced, the lookahead consists of the smallest subsequence, $r_{i+1} r_{i+2} \cdots r_j$, containing references to exactly M distinct blocks.

Both M -block lookahead and M -block strong lookahead [2] use a lookahead window that includes the next M distinct blocks. However, while the latter uses the largest such window, the former uses the smallest such window.

Definition 2. An online parallel buffer management algorithm \mathcal{A} has a competitive ratio of C_A if for any reference string the number of I/Os that \mathcal{A} requires is within a factor C_A of the number of I/Os required by the optimal off-line algorithm serving the same reference string. If \mathcal{A} is a randomized algorithm then the expected number of I/Os done by \mathcal{A} is considered.

2. PHASE-LRU

In this section we describe PHASE-LRU, and analyze its performance for random data placement. We shall show that PHASE-LRU has a competitive ratio of $\Theta(\log D / \log \log D)$, with M -block lookahead. The algorithm is described in Fig. 2.

For deterministic and static placement of data on the disks, PHASE-LRU has a worst-case competitive ratio of $\Omega(\min\{\sqrt{M}, D\})$. This can be seen by constructing a reference string Σ from alternating sequences of M blocks as follows: every odd sequence consists of references to the same set of M blocks on disk 1; every even sequence consists of references to M different blocks striped among N , $N \leq D - 1$ disks different from disk 1. All odd sequences therefore access the same set of M blocks sequentially from disk 1, while every even sequence accesses a fresh set of M

Algorithm PHASE-LRU

On a reference r_i , algorithm PHASE-LRU takes the following actions.

If $\text{block}(r_i)$ is present in the buffer then no I/O is necessary.

If $\text{block}(r_i)$ is not present in the buffer then

Let H_i be the set of blocks in the lookahead consisting of the first block from each disk that is not present in the buffer.

From among the blocks in the buffer that do not occur in the lookahead select the $|H_i|$ *least recently referenced* blocks for eviction.

Evict the selected blocks and initiate an I/O to fetch the blocks in $|H_i|$.

Service reference r_i .

Fig. 2. Algorithm PHASE-LRU.

blocks evenly striped among the same set of N disks. If Σ consists of k repetitions of the odd and even sequences, then it may be seen that PHASE-LRU will perform $\Theta(Mk)$ I/Os, since it will evict all blocks of an odd phase (even though they occur again in the next odd phase) to fetch blocks of the following even phase. In contrast, a better algorithm would cache most of the blocks (all but N) of the odd phases, and use the free N buffer to service the striped even phases with full parallelism. For the odd phases at most N blocks missing from the buffer need to be fetched one-at-a-time; for the even phases the blocks can be fetched in M/N parallel I/Os. The total number of I/Os done is bounded by $O((N + M/N)k)$, from which we get the following lemma.

Lemma 1. *The competitive ratio of PHASE-LRU for a deterministic data placement is $\Omega(\min\{\sqrt{M}, D\})$.*

Motivated by the pessimistic nature of these results, we explore the performance of PHASE-LRU in a randomized setting, where the blocks are distributed among the disks randomly. That is each block independently has a probability $1/D$ of being on any disk.

2.1. Analysis

We analyze the performance of PHASE in sequences that consist of references to M distinct blocks called *phases*. Blocks in a phase are colored either *red*, *blue* or *green* depending on where they were referenced previously.

Definition 3.

- The reference string is partitioned into subsequences of maximal length called phases, such that the number of distinct blocks referenced in any phase is exactly M . The first reference is in phase(1).
- Let \mathcal{S}_i denote the set of M blocks referenced in phase(i). Block $b \in \mathcal{S}_i$ is assigned a unique color as follows: b is colored *red* if it is not referenced in any phase(j), $j < i$; is colored *blue* if its last reference outside this phase was in phase($i - 1$); and is colored *green* if its last reference outside this phase was in some phase(j), $j \leq i - 2$.

Note that by the definition above, the same block may have different colors in different phases. However its color in any give phase is uniquely defined. In the following development we assume that phase(i) is an arbitrary but fixed phase of the reference string.

Let R_i , B_i and G_i denote the number of blocks of \mathcal{S}_i that are colored red, blue and green respectively. Note that by definition of phase, $R_i + G_i \geq 1$. For each color, let the number of blocks of \mathcal{S}_i that are present in the buffer, just prior to referencing the first block of phase(i) be r_i , b_i and g_i , respectively. Hence, at the start of phase(i) at most $M - (r_i + b_i + g_i)$ blocks of \mathcal{S}_i need to be fetched to service the phase; and the buffer contains $M - (r_i + b_i + g_i)$ blocks that are not referenced in phase(i). Denote these blocks by the set $\text{waste}(i)$. We show in Lemma 2 that all blocks of $\text{waste}(i)$ must have had their last reference in phase($i - 1$).

Lemma 2. *All blocks of $\text{waste}(i)$, found in PHASE-LRU's buffer at the start of phase(i), must have been referenced during phase($i - 1$).*

Proof. A block in $\text{waste}(i)$ that is not referenced in phase($i - 1$) must be referenced in phase($i - 2$) or earlier. Assume for purposes of contradiction that among all blocks in $\text{waste}(i)$ whose last reference was prior to phase($i - 1$), b is the block with the earliest last reference.

Since M blocks different from b are referenced during phase($i - 1$), PHASE-LRU must evict b unless it occurs in the lookahead. If b is evicted it will not be refetched into the buffer unless it reappears in the lookahead. In either case b cannot be in $\text{waste}(i)$: a contradiction. \square

Lemma 3. *The number of blue blocks of \mathcal{S}_i not present in the buffer at the start of phase(i) is at most $r_i + g_i$.*

Proof. By Lemma 2 all blocks of $\text{waste}(i)$ are referenced in phase($i - 1$). Therefore, at least $|\text{waste}(i)|$ blocks referenced in phase($i - 1$) are not referenced in phase(i), and hence there are at most $M - |\text{waste}(i)| = (r_i + b_i + g_i)$ blue blocks in \mathcal{S}_i . Now b_i blue blocks of \mathcal{S}_i are present in the buffer at the start of phase(i), so at most $(r_i + g_i)$ blue blocks are not present in the buffer. \square

Lemma 4. *The number of blocks of \mathcal{S}_i fetched by PHASE-LRU during phase(i) is at most $(R_i + G_i)$.*

Proof. Since no block referenced in phase(i) will be evicted if it is later referenced in the same phase, we need only count the number of blocks of \mathcal{S}_i that are not present in the buffer at the start of phase(i). Summing the number of each color and using Lemma 3, the number of blocks not in the buffer at the start of phase(i) is at most $(R_i - r_i) + (r_i + g_i) + (G_i - g_i)$. \square

To bound the number of I/Os done by PHASE-LRU in any phase, we relate it to the classical occupancy problem [11]. *Suppose that B balls are randomly (uniform distribution) thrown into U urns, what is the expected maximum number of balls in any urn? Let $\mathcal{N}_{B,U}$ denote the expected maximal occupancy when B balls are thrown into U urns.*

Lemma 5. *The expected value of the number of I/Os done by PHASE-LRU during phase(i) is at most $\lceil (R_i + G_i)/D \rceil \mathcal{N}_{D,D}$.*

Proof. Since PHASE-LRU has M -block lookahead, at the start of phase(i) it knows all the blocks in \mathcal{S}_i . Now PHASE-LRU greedily fetches a block in the lookahead from each disk, and never evicts a block in the lookahead. Hence, the number of I/Os made by PHASE-LRU in phase(i) is upper bounded by the maximum number of uncached blocks on any disk in that phase. Using Lemma 4, the expected maximal number of uncached blocks on any disk is bounded by $\mathcal{N}_{(R_i+G_i),D}$, which is at most $\lceil (R_i + G_i)/D \rceil \mathcal{N}_{D,D}$. \square

Lemma 6. *Let T_{OPT} denote the number of I/Os done by the optimal offline algorithm OPT in servicing the reference string. $T_{\text{OPT}} \geq \frac{1}{5} \sum_i 1 + \frac{R_i+G_i}{D}$, where the sum is taken over all phases in the reference string.*

Proof. The total number of red blocks in the reference string is $\sum_i R_i$; hence

$$T_{\text{OPT}} \geq \left\lceil \frac{\sum_i R_i}{D} \right\rceil.$$

Consider two adjacent phases, phase($i - 1$) and phase(i). By definition, there are $M + G_i$ distinct

blocks referenced in these two phases, and hence OPT must fetch at least G_i blocks in these two phases combined. Summing over all the phases we have

$$T_{\text{OPT}} \geq \left\lceil \frac{\sum_i G_i}{D} \right\rceil / 2.$$

Finally, by definition of a phase there are at least $M + 1$ blocks referenced in any two phases, and so OPT must fetch at least 1 block, doing at least 1 I/O, in each pair of phases. Hence, $T_{\text{OPT}} \geq \sum_i 1/2$. Combining the individual inequalities the lemma follows. \square

Theorem 1. *The competitive ratio of PHASE-LRU is $\Theta(\log D / \log \log D)$.*

Proof. Using Lemma 5, the expected number of I/Os done by PHASE-LRU is at most $\mathcal{N}_{D,D} (\sum_i 1 + \frac{R_i+G_i}{D})$, where R_i and G_i are the number of red and green blocks respectively in \mathcal{S}_i , and the sum is taken over all phases in the reference string. From Lemma 6 and using the fact that $\mathcal{N}_{D,D} = O(\log D / \log \log D)$, the result follows. \square

3. Discussion

We presented a simple buffer management algorithm for parallel I/O and analyzed its competitive ratio in the Parallel Disk Model. For deterministic placement the lower bound on competitive ratio of *any* algorithm is known [5] to be $\Omega(\sqrt{D})$. For PHASE-LRU a lower bound of $\Omega(\min\{\sqrt{M}, D\})$ was obtained. With random block placement on disks, PHASE-LRU performs significantly better, and was shown to have a competitive ratio of $\Theta(\log D / \log \log D)$.

References

- [1] J. Aerts, J. Korst, S. Egner, Random duplicate storage strategies for load balancing in multimedia servers, *Inform. Process. Lett.* 76 (1) (2000) 51–59.
- [2] S. Albers, On the influence of lookahead in competitive paging algorithms, *Algorithmica* 18 (3) (1997) 283–305.
- [3] S. Albers, N. Garg, S. Leonardi, Minimizing stall time in single and parallel disk systems, in: *Proc. of 30th Annual ACM Symp. on Theory of Computer Science*, ACM Press, New York, 1998, pp. 454–462.
- [4] R.D. Barve, E.F. Grove, J.S. Vitter, Simple randomized merge-sort on parallel disks, *Parallel Comput.* 23 (4) (1996) 601–631.

- [5] R.D. Barve, M. Kallahalla, P.J. Varman, J.S. Vitter, Competitive parallel disk prefetching and buffer management, *J. Algorithms* 36 (2000) 152–181.
- [6] L.A. Belady, A study of replacement algorithms for a virtual storage computer, *IBM Syst. J.* 5 (2) (1966) 78–101.
- [7] D. Breslauer, On competitive on-line paging with lookahead, in: Proc. of the 13th Annual Symp. on Theoretical Aspects of Computer Science, in: *Lecture Notes in Comput. Sci.*, vol. 1046, Springer-Verlag, Berlin, 1996, pp. 593–603.
- [8] P. Cao, E.W. Felten, A.R. Karlin, K. Li, A study of integrated prefetching and caching strategies, in: Proc. of the Joint Internat. Conf. on Measurement and Modeling of Computer Systems, ACM Press, New York, 1995, pp. 188–197.
- [9] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, D.A. Patterson, RAID: High performance reliable secondary storage, *ACM Comput. Surv.* 26 (2) (1994) 145–185.
- [10] A. Fiat, R. Karp, M. Luby, L. McGeoch, D.D. Sleator, N.E. Young, Competitive paging algorithms, *J. Algorithms* 12 (4) (1991) 685–699.
- [11] N.L. Johnson, S. Kotz, *Urn Models and Their Application: An Approach to Modern Discrete Probability Theory*, Wiley, New York, 1977.
- [12] M. Kallahalla, P.J. Varman, Randomized Parallel Prefetching and Buffer Management, in: P.M. Pardalos, S. Rajasekaran (Eds.), *Advances in Randomized Parallel Computing*, Kluwer Academic, Dordrecht, 1999, pp. 183–208, Chapter 9.
- [13] T. Kimbrel, A.R. Karlin, Near-optimal parallel prefetching and caching, *SIAM J. Comput.* 29 (4) (2000) 1051–1082.
- [14] L.A. McGeoch, D.D. Sleator, A strongly competitive randomized paging algorithm, *Algorithmica* 6 (1991) 816–825.
- [15] R.H. Patterson, G. Gibson, E. Ginting, D. Stodolsky, J. Zelenka, Informed prefetching and caching, in: Proc. of the 15th ACM Symp. on Operating Systems Principles, 1995, pp. 79–95.
- [16] P. Sanders, S. Egner, J. Korst, Fast concurrent access to parallel disks, in: Proc. of SIAM Symposium on Discrete Algorithms, 2000, pp. 849–858.
- [17] D.D. Sleator, R.E. Tarjan, Amortized efficiency of list update and paging rules, *Comm. ACM* 28 (2) (1985) 202–208.
- [18] P.J. Varman, R.M. Verma, Tight bounds for prefetching and buffer management algorithms for parallel I/O systems, in: *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, 1999, pp. 1262–1275.
- [19] J.S. Vitter, E.A.M. Shriver, Optimal algorithms for parallel memory, I: Two-level memories, *Algorithmica* 12 (2–3) (1994) 110–147.