

# Tight Bounds for Prefetching and Buffer Management Algorithms for Parallel I/O Systems

Peter J. Varman<sup>1</sup> \* and Rakesh M. Verma<sup>2</sup> \*\*

<sup>1</sup> ECE Department, Rice University, Houston TX 77251, USA  
E-mail: pjb@rice.edu

<sup>2</sup> Department of Computer Science, University of Houston, Houston TX 77204, USA  
E-mail: rmverma@cs.uh.edu

**Abstract.** The growing importance of multiple-disk parallel I/O systems requires the development of appropriate prefetching and buffer management algorithms. We answer several fundamental questions on prefetching and buffer management for such parallel I/O systems. Specifically, we find and prove the optimality of an algorithm, P-MIN, that minimizes the number of parallel I/Os. Secondly, we analyze P-CON, an algorithm which always matches its replacement decisions with those of the well-known demand-paged MIN algorithm. We show that P-CON can become fully sequential in the worst case. Finally, we define and analyze P-LRU, a semi-on-line version of the traditional LRU buffer-management algorithm. Unexpectedly, we find that the performance of P-LRU is independent of the number of disks.

## 1 Introduction

The increasing imbalance between the speeds of processors and I/O devices has resulted in the I/O subsystem becoming a bottleneck in many applications. The use of multiple disks to build a parallel I/O subsystem has been advocated to increase I/O performance and system availability [5], and most high-performance systems incorporate some form of I/O parallelism. Performance is improved by overlapping accesses at several disks using judicious prefetching and buffer management algorithms that ensure that the most useful blocks are accessed and retained in the buffer.

A parallel I/O system consists of  $D$  independent disks, each with its own disk buffer, that can be accessed in parallel. The data for the computation is spread out among the disks in units of *blocks*. A block is the unit of retrieval from a disk. The computation is characterized by a computation sequence, which is the ordered sequence of blocks that it references. In our model all accesses are read-only. Prefetching (reading a data block before it is needed by the computation)

---

\* Research partially supported by a grant from the Schlumberger Foundation

\*\* Research partially supported by NSF grant CCR-9303011

is a natural mechanism to increase I/O parallelism. When the computation demands a disk-resident block of data, concurrently a data block can be prefetched from each of the other disks in parallel, and held in buffer until needed. This requires discarding a block in the buffer to make space for the prefetched block. Some natural questions that arise are: under what conditions is it worthwhile to discard a buffer-resident block to make room for a prefetch block which will be used only some time later in the future? And, if we do decide to discard a block, what replacement policy should be used in choosing the block to be replaced.

In this paper we answer several fundamental questions on prefetching and buffer management for such parallel I/O systems. The questions we address are: what is an optimal prefetch and buffer management algorithm, and how good are the algorithms proposed earlier for sequential (single) disk systems in this context. We obtain several interesting results, which are informally stated below and more precisely stated in Section 2. We find and prove the optimality of an algorithm, P-MIN, that minimizes the number of parallel I/Os. This contrasts with the recent results on prefetching to obtain CPU-disk overlap [4], where no efficient algorithm to find the optimal policy is known. Secondly, we show that P-CON, an algorithm that attempts to optimize the number of I/Os on each disk, can have very poor parallel performance. Finally we investigate the behavior of semi-on-line algorithms using parallel I/O. The concept of semi-on-line algorithms that we consider in this paper captures the dual requirements of prefetching (which needs some future knowledge) and on-line behavior (no future knowledge). We define and analyze P-LRU, a semi-on-line version of the traditional Least Recently Used (LRU) buffer-management algorithm. We find the performance of P-LRU is independent of the number of disks, in contrast to P-CON where the performance can degrade in proportion to the number of disks.

In contrast to single-disk systems (sequential I/O) for which these issues have been studied extensively (e.g. [2, 6]), there has been no formal study of these issues in the parallel I/O context. In the sequential setting the number of block accesses (or I/Os) is a useful performance metric; scaling by the average block access time provides an estimate of the I/O time. In contrast, in the multiple-disk case there is no direct relationship between the number of I/Os and the I/O time, since this depends on the I/O parallelism that is attained. The goals of minimizing the number of I/Os done by each disk and minimizing the parallel I/O time can conflict. Traditional buffer management algorithms for single-disk systems have generally focused on minimizing the number of I/Os. In the parallel context it may be useful to perform a greater than the absolute minimal (if the disk were operated in isolation) number of I/Os from each disk, if this allows a large number of them to be overlapped.

The rest of the paper is organized as follows. Section 1.1 summarizes related work. Section 2 develops the formal model and summarizes the main results. In Section 3.1 we derive a tight upper bound for P-CON algorithm. In Section 3.2 we prove the optimality of P-MIN. Section 3.3 analyzes the performance of the semi-on-line algorithm P-LRU.

## 1.1 Related Work

In single-disk systems, buffer management (or paging problem) algorithms were studied [2, 6, 11], and several policies (LRU, FIFO, Longest Forward Distance, etc.) were proposed and analyzed. The Longest Forward Distance [2] policy minimizes the number of page faults, and is therefore called the MIN algorithm. All these policies use demand I/O and deterministic replacement, i.e. they fetch only when the block being referenced is not in the buffer, and the choice of the replaced block is deterministic. (Randomized replacement algorithms, e.g. see [8], are beyond the scope of this paper.) In the sequential case, it is well known [11] that prefetching does not reduce the number of I/Os required.

Sleator and Tarjan [11] analyzed the competitive ratio of on-line paging algorithms relative to the off-line optimal algorithm MIN. They showed that LRU's performance penalty can be proportional to the size of fast memory, but no other on-line algorithm can, in the worst case, do much better. These fundamental results have been extended in several ways, most often to include models that allow different forms of lookahead [3, 1, 9, 7]. All these works deal with the question of which buffer block to evict. In contrast, in our situation the additional question that arises is *when* to fetch a block and evict some other.

Cao et al. [4] examined prefetching from a single disk to overlap CPU and I/O operations. They defined two off-line policies called *aggressive* and *conservative*, and obtained bounds on the elapsed time relative to the optimal algorithm. We use prefetching to obtain I/O parallelism with multiple disks, and use the number of parallel I/Os (elapsed I/O time) as the cost function. The P-MIN and P-CON algorithms analyzed here generalize the aggressive and conservative policies respectively. However, while aggressive is suboptimal in the model of [4], P-MIN is proved to be the optimal algorithm in our model. The prefetching algorithm for multiple disks analyzed in [10] assumed a global buffer and read-once random data.

We also investigate semi-on-line algorithms using parallel I/O. Since prefetching involves reading blocks that are required in the future (relative to where the computation has progressed), this presents a natural situation where lookahead is necessary. This inspires us to define a lookahead version of LRU, P-LRU, in which the minimum possible lookahead of one block beyond those currently in the buffer is known for each disk.<sup>3</sup> We find the performance of P-LRU is independent of the number of disks, in contrast to P-CON whose performance can degrade in proportional to the number of disks.

## 2 Preliminaries

The computation references the blocks on the disks in an order specified by the *consumption sequence*,  $\Sigma$ . When a block is referenced the buffer for that disk is checked; if the block is present in the buffer it is consumed by the computation,

---

<sup>3</sup> Recently, Breslauer [7] arrived at this lookahead definition independently in a sequential demand context.

which then proceeds to reference the next block in  $\Sigma$ . If the referenced block is not present in the disk buffer, then an I/O (known as a *demand I/O*) for the missing block is initiated from that disk. If only demand I/Os were initiated, then the other disks in the system would idle while this block was being fetched. However, every demand I/O at a disk provides a *prefetch* opportunity at the other disks, which may be used to read blocks that will be referenced in the near future. For example, consider a 2-disk system holding blocks  $(a_1, a_2)$  and  $(b_1, b_2)$  on disks 1 and 2 respectively. If  $\Sigma = a_1, b_1, b_2, a_2$ , then strictly demand I/O would require four non-overlapped I/Os to fetch the blocks. A better strategy is to overlap reads using prefetching. During the demand I/O of block  $a_1$ , the second disk could concurrently prefetch  $b_1$ ; after  $a_1$  and  $b_1$  have been consumed, a demand I/O for block  $b_2$  will be made concurrently with a prefetch of block  $a_2$ . The number of parallel I/Os in this case is now two.

While prefetching can increase the I/O parallelism, the problem is complicated by the finite buffer sizes. For every block read from a disk some previously fetched block in the corresponding buffer must be replaced. For prefetch blocks the replacement decision is being made earlier than is absolutely necessary, since the computation can continue without the prefetched block. These early replacement choices can be much poorer than replacement choices made later, since as the computation proceeds, other, more useful replacement candidates may become available. Of course, once a block becomes a demand block then the replacement cannot be deferred. A poor replacement results in a greater number of I/Os as these prematurely discarded blocks may have to be fetched repeatedly into the buffer. Thus there is a tradeoff between the I/O parallelism that can be achieved (by using prefetching), and the increase in the number of I/Os required (due to poorer replacement choices).

## 2.1 Definitions

The consumption sequence  $\Sigma$  is the order in which blocks are requested by the computation. The subsequence of  $\Sigma$  consisting of blocks from *disk*  $i$  will be denoted by  $\Sigma^i$ . Computation occurs in *rounds* with each round consisting of an *I/O phase* followed by a *computation phase*. In the I/O phase a parallel I/O is initiated and some number of blocks, at most one from any disk, are selected to be read. For each selected disk, a block in the corresponding disk buffer is chosen for replacement. When all new blocks have been read from the disks, the computation phase begins. The CPU consumes zero or more blocks that are present in the buffer in the order specified by  $\Sigma$ . If at any point the next block of  $\Sigma$  is not present in a buffer, then the round ends and the next round begins. The block whose absence forced the I/O is known as a *demand block*; blocks that are fetched together with the demand block are known as *prefetch blocks*. An I/O phase may also be initiated before the computation requires a demand block. In this case all the blocks fetched in are prefetch blocks. We will often refer to the I/O phase of a round as an I/O time step.

**Definition 1.** An *I/O schedule* with *makespan*  $T$ , is a sequence  $\langle \mathcal{F}_1, \dots, \mathcal{F}_T \rangle$ , where  $\mathcal{F}_k$  is the set of blocks (at most one from each disk) fetched by the parallel

I/O at time step  $k$ . The makespan of a schedule is the number of I/O time steps required to complete the computation.

**Definition 2.** A *valid schedule* is one in which axioms A1 and A2 are satisfied.

- A1: A block must be present in the buffer before it can be consumed.
- A2: There are at most  $M$ , where  $M$  is the buffer size, blocks in any disk buffer at any time.

**Definition 3.**

- An *optimal schedule* is a valid schedule with minimal makespan among all valid schedules.
- A *normal schedule* is a valid schedule in which each  $\mathcal{F}_k$ ,  $1 \leq k \leq T$ , contains a demand block.
- A *sequential schedule* is a valid schedule in which the blocks from each disk  $i$  are fetched in the order of  $\Sigma^i$ .

At the start of a round, let  $U_i$  denote the next referenced block of  $\Sigma^i$  that is not currently in the buffer of disk  $i$ . Define a *min-block* of disk  $i$ , to be the block in disk  $i$ 's buffer with the longest forward distance to the next reference.

**Definition 4.** A P-MIN schedule  $\mathcal{S} = \langle \mathcal{F}_k, k = 1, \dots, T \rangle$ , is a normal, sequential schedule in which at I/O step  $k$ ,  $U_i \in \mathcal{F}_k$  unless all blocks in disk  $i$ 's buffer are referenced before  $U_i$ . If  $U_i \in \mathcal{F}_k$ , then replace the min-block of disk  $i$  with  $U_i$ .

**Definition 5.** A P-CON schedule  $\mathcal{S} = \langle \mathcal{F}_k, k = 1, \dots, T \rangle$ , is a normal, sequential schedule in which at every I/O step  $k$ ,  $U_i \in \mathcal{F}_k$  provided the min-block of disk  $i$  now is the same as the min-block if  $U_i$  were fetched on demand. If  $U_i \in \mathcal{F}_k$  then replace the min-block of disk  $i$  with  $U_i$ .

**Definition 6.** A P-LRU schedule  $\mathcal{S} = \langle \mathcal{F}_k, k = 1, \dots, T \rangle$ , is a normal, sequential schedule in which at every I/O step  $k$ ,  $U_i \in \mathcal{F}_k$  unless all blocks in disk  $i$ 's buffer are referenced before  $U_i$ . If  $U_i \in \mathcal{F}_k$  then from among the blocks in the buffer whose next reference is not before that of  $U_i$  choose the least recently used block and replace it with  $U_i$ .

Notice that all three schedules defined above are normal. That is in every I/O step, one disk is performing a demand fetch and the rest are either performing a prefetch or are idle. Of these P-MIN and P-LRU are greedy strategies, and will almost always attempt to prefetch the next unread block from a disk. The only situation under which a disk will idle is if every block in the buffer will be referenced before the block to be fetched. Note that this greedy prefetching may require making “suboptimal” replacement choices, that can result in an increase the number of I/Os done by that disk. We show, however, that P-MIN policy has the minimal I/O time, and is therefore optimal.

An example with  $D = 2$  and  $M = 3$  is presented below. Let the blocks on disk 1 (2) be  $a_i, i = 1, \dots, 4$  (respectively  $b_i, i = 1, \dots, 4$ ). Assume that:

$$\Sigma = a_1 a_2 a_3 b_1 a_4 a_1 a_2 b_2 b_3 b_4 a_3 b_1 b_2 b_3 a_1 a_2 a_3 b_1 a_4 a_1 a_2$$

Round	Disk 1	Disk 2	CPU
1	<b>a<sub>1</sub></b> /-	b <sub>1</sub> /-	a <sub>1</sub>
2	<b>a<sub>2</sub></b> /-	b <sub>2</sub> /-	a <sub>2</sub>
3	<b>a<sub>3</sub></b> /-	b <sub>3</sub> /-	a <sub>3</sub> , b <sub>1</sub>
4	<b>a<sub>4</sub></b> /a <sub>3</sub>	b <sub>4</sub> /b <sub>1</sub>	a <sub>4</sub> , a <sub>1</sub> , a <sub>2</sub> , b <sub>2</sub> , b <sub>3</sub> , b <sub>4</sub>
5	<b>a<sub>3</sub></b> /a <sub>4</sub>	b <sub>1</sub> /b <sub>4</sub>	a <sub>3</sub> , b <sub>1</sub> , b <sub>2</sub> , b <sub>3</sub> , a <sub>1</sub> , a <sub>2</sub> , a <sub>3</sub> , b <sub>1</sub>
6	<b>a<sub>4</sub></b> /a <sub>3</sub>	-/-	a <sub>4</sub> , a <sub>1</sub> , a <sub>2</sub>

P-MIN Schedule

Round	Disk 1	Disk 2	CPU
1	<b>a<sub>1</sub></b> /-	b <sub>1</sub> /-	a <sub>1</sub>
2	<b>a<sub>2</sub></b> /-	b <sub>2</sub> /-	a <sub>2</sub>
3	<b>a<sub>3</sub></b> /-	b <sub>3</sub> /-	a <sub>3</sub> , b <sub>1</sub>
4	<b>a<sub>4</sub></b> /a <sub>3</sub>	-/-	a <sub>4</sub> , a <sub>1</sub> , a <sub>2</sub> , b <sub>2</sub> , b <sub>3</sub>
5	a <sub>3</sub> /a <sub>4</sub>	<b>b<sub>4</sub></b> /b <sub>3</sub>	b <sub>4</sub> , a <sub>3</sub> , b <sub>1</sub> , b <sub>2</sub>
6	-/-	<b>b<sub>3</sub></b> /b <sub>4</sub>	b <sub>3</sub> , a <sub>1</sub> , a <sub>2</sub> , a <sub>3</sub> , b <sub>1</sub>
7	<b>a<sub>4</sub></b> /a <sub>3</sub>	-/-	a <sub>4</sub> , a <sub>1</sub> , a <sub>2</sub>

P-CON Schedule

Round	Disk 1	Disk 2	CPU
1	<b>a<sub>1</sub></b> /-	b <sub>1</sub> /-	a <sub>1</sub>
2	<b>a<sub>2</sub></b> /-	b <sub>2</sub> /-	a <sub>2</sub>
3	<b>a<sub>3</sub></b> /-	b <sub>3</sub> /-	a <sub>3</sub> , b <sub>1</sub>
4	<b>a<sub>3</sub></b> /-	b <sub>3</sub> /-	a <sub>3</sub> , b <sub>1</sub>
5	<b>a<sub>4</sub></b> /a <sub>1</sub>	<b>b<sub>4</sub></b> /b <sub>1</sub>	a <sub>4</sub>
6	<b>a<sub>1</sub></b> /a <sub>2</sub>	-/-	a <sub>1</sub>
7	<b>a<sub>2</sub></b> /a <sub>3</sub>	-/-	a <sub>2</sub> , b <sub>2</sub> , b <sub>3</sub> , b <sub>4</sub>
8	-/-	<b>b<sub>2</sub></b> /b <sub>3</sub>	b <sub>2</sub>
9	-/-	<b>b<sub>3</sub></b> /b <sub>4</sub>	b <sub>3</sub> , a <sub>1</sub> , a <sub>2</sub> , a <sub>3</sub> , b <sub>1</sub>
10	<b>a<sub>4</sub></b> /a <sub>1</sub>	-/-	a <sub>4</sub>
11	<b>a<sub>1</sub></b> /a <sub>2</sub>	-/-	a <sub>1</sub>
12	<b>a<sub>2</sub></b> /a <sub>3</sub>	-/-	a <sub>2</sub>

P-LRU Schedule

Fig. 1. Examples of I/O Schedules

Figure 1 shows the I/O schedule using different policies for the example sequence. The entries in the second and third columns indicate the blocks that are fetched and replaced from that disk at that round. Bold and italic faced blocks indicate a demand block, and a prefetch block respectively.

In contrast to P-MIN, the conservative strategy [4] P-CON, is pessimistic. It does not perform a prefetch unless it can replace the “best” block, so that the number of I/Os done by any disk is the smallest possible. However, while minimizing the number of I/Os done by a disk, it may result in serialization of these accesses, and perform significantly worse than the optimal algorithm. Note that in Figure 1 at step 4, no block is fetched from disk 2 by P-CON. This is because the only candidate for replacement at this time (the current min-block) is block  $b_1$ ; however, if  $b_4$  were a demand block, the min-block would be  $b_3$ .

To take advantage of a prefetch opportunity, any algorithm must know which is the next unread block in  $\Sigma^i$ . That is, it requires to have a lookahead upto at least one block beyond those currently in its buffer. The replacement decision made by P-LRU is based solely by examining the current blocks in the buffer, and tracking which of them will be referenced before the next unread block. From those whose next reference is not before that of the next unread block, the least recently consumed block is chosen as the replacement candidate. If P-LRU is applied to the sequence  $\Sigma$  used for P-MIN and P-CON, a schedule of 12 I/O steps will be obtained (see Fig. 1). In the next section, we quantify precisely the performance of these three algorithms.

## 2.2 Summary of Results

Let  $T_{P-MIN}$ ,  $T_{P-CON}$  and  $T_{P-LRU}$  be the number of I/O time steps required by P-MIN, P-CON and P-LRU respectively. Let  $T_{opt}$  be the number of I/O steps required by an optimal schedule. Let  $N$  denote the length of  $\Sigma$ ; also recall that

$D$  is the number of disks and  $M$  the size of each disk buffer in blocks. The technical results of the paper are as follows.

1. The worst-case ratio between the makespan of a P-CON schedule and the corresponding optimal schedule is bounded by  $D$ . That is, at worst P-CON can serialize all disk accesses without increasing the number of I/Os performed by any disk (see Theorem 7).
2. The worst-case bound of P-CON stated above is tight. That is, there are consumption sequences, for which P-CON completely serializes its accesses (see Theorem 8).
3. P-MIN is an optimal schedule. That is it minimizes the number of parallel I/O time steps over all other valid schedules (see Theorem 11.)
4. The worst-case ratio between the makespan of a P-LRU schedule and the corresponding optimal schedule is bounded by  $M$ . That is, at worst P-LRU can inflate the number of I/Os performed by any disk to that done by a serial LRU algorithm (see Theorem 13).
5. The worst-case bound of P-LRU stated above is tight. That is, there are consumption sequences, for which P-LRU does inflate the accesses for each disk by a factor of  $M$  (see Theorem 14).

### 3 Detailed Results

#### 3.1 Bounds for P-CON

We begin with a simple upper bound for  $T_{\text{P-CON}}$ . Let  $T_{\text{MIN}}$  denote the maximum number of I/Os done by the sequential MIN algorithm to a single disk.

**Theorem 7.** *For any consumption sequence,  $T_{\text{P-CON}} \leq D T_{\text{opt}}$ .*

*Proof.* We show that  $T_{\text{P-CON}} \leq DT_{\text{MIN}} \leq DT_{\text{opt}}$ . The I/Os made by P-CON to disk  $i$  for consumption sequence  $\Sigma$ , are exactly the I/Os done by the sequential MIN algorithm to disk  $i$  for sequence  $\Sigma^i$ . Hence the number of I/Os performed by any disk in P-CON is bounded by  $T_{\text{MIN}}$ . At worst, none of the accesses of any of the disks can be overlapped whence the first inequality follows. Finally, the second inequality follows since the optimal parallel time for  $D$  disks cannot be smaller than the minimal number of I/Os for a single disk.  $\square$

**Theorem 8.** *The bound of Theorem 7 is tight.*

*Proof (sketch).* Construct the following four length- $M$  sequences, where  $B_i(j)$  is the  $j^{\text{th}}$  block on disk  $i$ . Also define  $\Sigma$  as follows, where  $(u)^N$  means  $N$  repetitions of the parenthesized sequence.

$$\begin{aligned} \alpha_i &= B_i(2M)B_i(1)B_i(2) \cdots B_i(M-1) \\ \beta_i &= B_i(M), B_i(1)B_i(2) \cdots B_i(M-1) \\ \gamma_i &= B_i(M)B_i(M+1)B_i(M+2) \cdots B_i(2M-1) \\ \delta_i &= B_i(2M)B_i(M+1)B_i(M+2) \cdots B_i(2M-1) \\ \Sigma &= (\alpha_1\beta_1\alpha_2\beta_2 \cdots \alpha_D\beta_D\gamma_1\delta_1\gamma_2\delta_2 \cdots \gamma_D\delta_D)^N \end{aligned}$$

It can be argued that for  $\Sigma$ ,  $T_{\text{P-CON}} = M + D + (2N - 1)MD = 2NMD - (M - 1)(D - 1) + 1$ , and that the P-MIN schedule has length  $2NM + D - 1$ , for  $M \geq D$ ; hence  $T_{\text{opt}} \leq 2NM + D - 1$ . Thus,  $T_{\text{P-CON}}/T_{\text{opt}}$  is lower bounded by  $\frac{2NMD - (M-1)(D-1) + 1}{2NM + D - 1} \geq D(1 - 1/2N)$ .  $\square$

### 3.2 Optimality of P-MIN

In this section we show that P-MIN requires the minimal number of parallel I/O steps among all valid schedules. For the proof, we show how to transform an optimal schedule OPT with makespan  $L$ , into a P-MIN schedule with the same makespan.

**Definition 9.** Schedules  $\alpha$  and  $\beta$  are said to *match* for time steps  $[T]$ , if for every  $t$ ,  $t \in \langle 1 \dots T \rangle$ , the blocks fetched and replaced from each disk in the two sequences are the same.

**Lemma 10.** *Assume that  $\alpha$  is a valid schedule of length  $W$ . Let  $\gamma$  be another schedule which matches  $\alpha$  for  $[T - 1]$ . After the I/O at time step  $T$ , the buffers of  $\alpha$  and  $\gamma$  for some disk  $i$  differ in one block: specifically  $\alpha$  has block  $V$  but not block  $U$ , and  $\gamma$  has block  $U$  but not block  $V$ . Assume that  $V$  is referenced after  $U$  in the consumption sequence following the references at time  $T - 1$ . We can construct a valid schedule  $\beta$  of length  $W$ , such that  $\beta$  and  $\alpha$  match for  $[T - 1]$  and  $\beta$  and  $\gamma$  match at time step  $T$ .*

*Proof.* Let  $T + \delta$ ,  $\delta > 0$ , be the first time step after  $T$  that  $\alpha$  fetches or discards either block  $V$  or  $U$ . It can either discard block  $V$ , or fetch block  $U$ , or do both, at  $T + \delta$ . Construct schedule  $\beta$  as follows:  $\beta$  matches  $\alpha$  for time steps  $[W]$  *except* at time steps  $T$  and  $T + \delta$ .

At  $T$ ,  $\beta$  fetches and replaces the same blocks as  $\gamma$ . At  $T + \delta$ , one of the following must occur:

- $\alpha$  fetches a block  $Z \neq U$  and discards block  $V$ : then in the construction  $\beta$  will also fetch block  $Z$ , but will discard block  $U$ .
- $\alpha$  fetches block  $U$  and discards block  $Z$ ,  $Z \neq V$ : then  $\beta$  will fetch block  $V$  and will also discard block  $Z$ .
- $\alpha$  fetches block  $U$  and discards block  $V$ : then  $\beta$  does not fetch or discard any block.

In all three cases above, following the I/O at  $T + \delta$  both  $\beta$  and  $\alpha$  will have the same blocks in the buffer. Since  $\beta$  fetches and replaces the same blocks as  $\alpha$  for all time steps  $t \geq T + \delta + 1$ , the buffers of  $\alpha$  and  $\beta$  will be the same for all time steps after the I/O at  $T + \delta$ .

At each time step  $t$ ,  $1 \leq t \leq W$ ,  $\beta$  will consume the same blocks as done by  $\alpha$  at  $t$ . Clearly,  $\beta$  satisfies axiom A2. We show that  $\beta$  is a valid schedule by showing that axiom A1 is satisfied for all blocks consumed by  $\beta$ .

Since  $\alpha$  and  $\beta$  have the same buffer before the I/O at  $T$  and after the I/O at  $T + \delta$ , the blocks consumed by  $\alpha$  at any time step  $t$ ,  $t \leq T - 1$  or  $t \geq T + \delta$  can also be consumed by  $\beta$  at the same time step.

Let  $T', T' \geq T$  be the first time step after the I/O at  $T$  that either block  $U$  or  $V$  is consumed by  $\alpha$ . Since  $\alpha$  does not have  $U$  in buffer till at least after the I/O at  $T + \delta$ , and by the hypothesis,  $V$  is consumed *after*  $U$ ,  $T' \geq T + \delta$ . Hence only blocks,  $X \neq U, V$  can be consumed by  $\alpha$  at time steps  $t$ ,  $T \leq t \leq T + \delta - 1$ . Since the buffers of  $\alpha$  and  $\beta$  agree except on  $\{U, V\}$ ,  $X$  can also be consumed by  $\beta$  at the same time step. Since  $\alpha$  is a valid schedule, all consumptions of  $\beta$  also satisfy axiom A1. Hence,  $\beta$  is a valid schedule.  $\square$

**Theorem 11.** *P-MIN is an optimal schedule.*

*Proof.* Let  $\Delta$  and  $\Omega$  denote the schedules created by P-MIN and OPT algorithms respectively. We successively transform  $\Omega$  into another valid schedule that matches  $\Delta$  and has the same length as  $\Omega$ . This will show that the P-MIN schedule is optimal.

The proof is by induction. For the Induction Hypothesis assume that at time step  $t$ ,  $\Omega$  has been transformed to a valid schedule  $\Omega_t$  which matches  $\Delta$  at time steps  $[t]$ . We show how to transform  $\Omega_t$  to  $\Omega_{t+1}$  below.

We discuss the transformation for an arbitrary disk at time step  $t + 1$ . The same construction is applied to each disk independently. If  $\Delta$  and  $\Omega_t$  match at  $t + 1$ , then let  $\Omega_{t+1}$  be the same as  $\Omega_t$ . Suppose  $\Delta$  and  $\Omega_t$  differ at time step  $t + 1$ . Then one of the following three cases must occur at  $t + 1$ : We consider each case separately.

- **Case 1** -  $\Delta$  fetches a block but  $\Omega_t$  does not fetch any block: Let  $\Delta$  fetch block  $P$  and discard block  $Q$  at  $t + 1$ . Since  $\Delta$  always fetches blocks in the order in which they are referenced,  $P$  will be referenced before  $Q$ . From the Induction Hypothesis,  $\Delta$  and  $\Omega_t$  have the same buffer at the start of time step  $t + 1$ . Hence after the I/O at  $t + 1$ ,  $\Delta$  and  $\Omega_t$  differ in one block:  $\Delta$  has block  $P$  but not block  $Q$ , while  $\Omega_t$  has  $Q$  but not  $P$ .

Using Lemma 10 with  $\alpha = \Omega_t, \beta = \Omega_{t+1}, \gamma = \Delta, U = P, V = Q, T = t + 1$ , we can construct valid schedule  $\Omega_{t+1}$  that matches  $\Omega_t$  at time steps  $[t]$  and  $\Delta$  at time  $t + 1$ . Hence, the Induction Hypothesis is satisfied for  $t + 1$ .

- **Case 2** -  $\Omega_t$  fetches a block but  $\Delta$  does not fetch any block: Since  $\Delta$  does not fetch any block at time step  $t + 1$ , every block in the buffer at the start of time step  $t + 1$ , will be consumed before any block not currently in the buffer is referenced.

Since  $\Delta$  and  $\Omega_t$  have the same buffer at the start of time step  $t + 1$ , if  $\Omega_t$  brings in a fresh block ( $P$ ) at  $t + 1$ , it must discard some block ( $Q$ ). Since  $\Delta$  chose to retain block  $Q$  in preference to fetching block  $P$ , then either  $Q$  must be referenced before  $P$ , or neither  $P$  nor  $Q$  will be referenced again.

In the first case, using Lemma 10 with  $\alpha = \Omega_t, \beta = \Omega_{t+1}, \gamma = \Delta, U = Q, V = P, T = t + 1$ , we can construct  $\Omega_{t+1}$ , a schedule that satisfies the Induction Hypothesis for  $t + 1$ .

In the second case,  $\Omega_{t+1}$  is the same as  $\Omega_t$ , except that at time step  $t + 1$ ,  $\Omega_{t+1}$  does not fetch any block. Since, the buffers of  $\Omega_t$  and  $\Omega_{t+1}$  agree on all blocks except  $P$  and  $Q$ , and these two blocks are never referenced again,

all blocks consumed by  $\Omega_t$  at a time step can also be consumed by  $\Omega_{t+1}$  at that time.

- **Case 3** -  $\Delta$  and  $\Omega_t$  fetch different blocks: Suppose that  $\Delta$  fetches block  $P$  and discards block  $Q$  at  $t + 1$ , and  $\Omega_t$  fetches block  $Y$  and discards block  $Z$  at  $t + 1$ . Assume that  $Q \neq Z$ , since otherwise the buffers of  $\Omega_t$  and  $\Delta$  differ in just the pair of blocks  $\{P, Y\}$ , and we can easily construct  $\Omega_{t+1}$  as before by using Lemma 10 with  $\alpha = \Omega_t, \gamma = \Delta, \beta = \Omega_{t+1}, U = P, V = Y, T = t + 1$ . By the Induction Hypothesis,  $\Delta$  and  $\Omega_t$  have the same buffer at the start of time step  $t + 1$ . Hence after the I/O at  $t + 1$ ,  $\Delta$  and  $\Omega_t$  differ in two blocks; specifically,  $buffer(\Delta) = (buffer(\Omega_t) - \{Y, Q\}) \cup \{P, Z\}$ , where  $buffer(\Theta)$  is the set of blocks in the buffer of schedule  $\Theta$ .

Let  $t + \delta, \delta > 1$ , be the first time after  $t + 1$  that  $\Omega_t$  fetches or replaces a block  $W \in \{P, Q, Y, Z\}$ . It can either discard block  $Q$  or  $Y$ , or fetch block  $P$  or  $Z$ , or some appropriate combination of these (see cases below), at  $t + \delta$ . Construct schedule  $\Omega'_{t+1}$  as follows:  $\Omega'_{t+1}$  matches  $\Omega_t$  at all time steps except  $t + 1$  and  $t + \delta$ . At  $t + 1$ ,  $\Omega'_{t+1}$  fetches  $P$  and discards  $Q$ , following the actions of  $\Delta$  at this time step. Hence after the I/O at  $t + 1$ ,  $buffer(\Omega'_{t+1}) = (buffer(\Omega_t) - \{Y, Q\}) \cup \{P, Z\}$ .

At  $t + \delta$ , one of the following will occur:

- $\Omega_t$  fetches block  $S \notin \{P, Z\}$  and discards  $Q$ : then  $\Omega'_{t+1}$  also fetches  $S$ , but discards  $Z$ . After the I/O at  $t + \delta$ ,  $buffer(\Omega'_{t+1}) = (buffer(\Omega_t) - \{Y\}) \cup \{P\}$ .
- $\Omega_t$  fetches  $P$  and discards  $Q$ : then  $\Omega'_{t+1}$  fetches  $Y$  and discards  $Z$ . After the I/O at  $t + \delta$ ,  $buffer(\Omega'_{t+1}) = buffer(\Omega_t)$ .
- $\Omega_t$  fetches  $Z$  and discards  $Q$ : then  $\Omega'_{t+1}$  does nothing at this step. After the I/O at  $t + \delta$ ,  $buffer(\Omega'_{t+1}) = (buffer(\Omega_t) - \{Y\}) \cup \{P\}$ .
- $\Omega_t$  fetches  $S \notin \{P, Z\}$  and discards  $Y$ : then  $\Omega'_{t+1}$  also fetches  $S$ , but discards  $P$ . After the I/O at  $t + \delta$ ,  $buffer(\Omega'_{t+1}) = (buffer(\Omega_t) - \{Q\}) \cup \{Z\}$ .
- $\Omega_t$  fetches  $P$  and discards  $Y$ : then  $\Omega'_{t+1}$  does not fetch any block at this time step. After the I/O at  $t + \delta$ ,  $buffer(\Omega'_{t+1}) = (buffer(\Omega_t) - \{Q\}) \cup \{Z\}$ .
- $\Omega_t$  fetches  $Z$  and discards  $Y$ : then  $\Omega'_{t+1}$  fetches  $Q$  and discards  $P$ . After the I/O at  $t + \delta$ ,  $buffer(\Omega'_{t+1}) = buffer(\Omega_t)$ .
- $\Omega_t$  fetches  $P$  and discards block  $S \notin \{Q, Y\}$ : then  $\Omega'_{t+1}$  fetches  $Y$  and discards block  $S$ . After the I/O at  $t + \delta$ ,  $buffer(\Omega'_{t+1}) = (buffer(\Omega_t) - \{Q\}) \cup \{Z\}$ .
- $\Omega_t$  fetches  $Z$  and discards block  $S \notin \{Q, Y\}$ : then  $\Omega'_{t+1}$  fetches  $Q$  and discards block  $S$ . After the I/O at  $t + \delta$ ,  $buffer(\Omega'_{t+1}) = (buffer(\Omega_t) - \{Y\}) \cup \{P\}$ .

Consider the consumptions made by  $\Omega_t$  at time steps  $T, t + 1 \leq T \leq t + \delta - 1$ . Notice that in the consumption sequence  $P$  must precede both  $Q$  and  $Y$ , and  $Z$  must precede  $Q$ . The constraints on  $P$  follow since  $\Delta$  fetches  $P$  and discards  $Q$ , and fetches  $P$  in preference to  $Y$ . The constraint on  $Z$  follows since  $\Delta$  discards  $Q$  rather than  $Z$ .

We now show how to transform  $\Omega'_{t+1}$  to  $\Omega_{t+1}$ . If the buffers of  $\Omega'_{t+1}$  and  $\Omega_t$  are the same after the I/O at  $t + \delta$ , then let  $\Omega_{t+1} = \Omega'_{t+1}$ . Otherwise, the buffers must differ in either the pair of blocks  $\{Q, Z\}$  or  $\{Y, P\}$ .

We will construct  $\Omega_{t+1}$  by concatenating the prefix of  $\Omega'_{t+1}$  between time steps 1 and  $t + \delta - 1$  with a schedule  $\beta$  that will be constructed using Lemma 10, as described below.

Let  $\alpha$  and  $\gamma$  be, respectively, the schedules consisting of the suffixes of  $\Omega_t$  and  $\Omega'_{t+1}$  for time steps greater than or equal to  $t + \delta$ . If at the end of the I/O at  $t + \delta$ , the buffers of  $\Omega_t$  and  $\Omega'_{t+1}$  differ in  $\{Y, P\}$ , then let  $U = P$  and  $V = Y$ ; otherwise, if they differ in  $\{Q, Z\}$ , then let  $U = Z$  and  $V = Q$ . Applying Lemma 10 with  $T = 1$ , we can construct the desired sequence  $\beta$ .  $\Omega_{t+1}$  is obtained by concatenating the prefix of  $\Omega'_{t+1}$  between 1 and  $t + \delta - 1$  with  $\beta$ .

The consumptions of blocks in  $\Omega_{t+1}$  are as follows: for time steps  $T$ ,  $1 \leq T \leq t + \delta - 1$ , the consumptions are those of  $\Omega_t$ , and for  $T \geq t + \delta$  the consumptions are determined by  $\beta$ . All consumptions from 1 till  $t$  are valid since  $\Omega_t$  is a valid schedule, and  $\Omega_{t+1}$  and  $\Omega_t$  match for  $[t]$ . By construction blocks consumed after  $t + \delta$  onwards are valid. We need to show that  $\Omega_{t+1}$  can consume the same blocks as  $\Omega_t$  at time steps  $T$ ,  $t + 1 \leq T \leq t + \delta - 1$ . Since  $\Omega_t$  does not have  $P$  or  $Z$  in buffer at the end of the I/O at  $t + 1$ , it can consume  $P$  or  $Z$  only after the I/O at time  $t + \delta$ , or later. Also, since  $Q$  and  $Y$  must be consumed *after*  $P$ , none of the blocks  $P, Q, Y, Z$  can be consumed by  $\Omega_t$  before the I/O at  $t + \delta$ . Since after the I/O at  $t + 1$ , the buffers of  $\Omega_t$  and  $\Omega_{t+1}$  agree except on  $\{P, Q, Y, Z\}$ , all blocks consumed by  $\Omega_t$  between  $t + 1$  and  $t + \delta - 1$  can also be consumed by  $\Omega_{t+1}$  at that time step.

This concludes the proof. □

### 3.3 Bounds for P-LRU

We now obtain an upper bound on the worst-case performance of P-LRU, and show that this bound is tight. We use the following lemma whose proof is omitted for brevity.

**Lemma 12.** *Let  $\mathcal{S}$  be a contiguous subsequence of  $\Sigma$  which references  $M$  or less distinct blocks from some disk  $i$ . Then in consuming  $\mathcal{S}$ , none of these blocks will be fetched more than once by P-LRU.*

**Theorem 13.** *For all consumption sequences,  $T_{\text{P-LRU}} \leq MT_{\text{opt}}$ .*

*Proof.* Inductively assume that consumptions made in the first  $t$  steps by P-MIN can be done in  $Mt$  or less steps by P-LRU. (This holds for  $t = 1$ .) Let  $U_i$  be the set of references made by P-MIN at  $t + 1$  from disk  $i$ .  $|U_i| \leq M$ , since at most  $M$  distinct blocks can be consumed from any disk at a time step of P-MIN. Since P-LRU will fetch a block of  $U_i$  at most once (Lemma 12), all P-MIN's consumptions at  $t + 1$  can be done in at most an additional  $M$  steps. □

**Theorem 14.** *The worst-case bound of Theorem 13 is tight.*

*Proof (sketch).* We show the construction of  $\Sigma$  for two disks;  $a_i$  and  $b_i$  are blocks from disks 1 and 2 respectively. Note that after the first  $M$  accesses of  $\Sigma$  (common to both P-LRU and P-MIN), P-LRU makes  $M$  accesses for every access of P-MIN.

$$\Sigma = (a_1, \dots, a_M, b_1, a_{M+1}, a_1, \dots, a_{M-1}, b_2, \dots, b_{M+1}, a_M, b_1, \dots, b_M)^N \quad \square$$

## 4 Discussion

In this paper we defined a model for parallel I/O systems, and answered several fundamental questions on prefetching and buffer management for such systems. We found and proved the optimality of an algorithm, P-MIN, that minimizes the number of parallel I/Os (while possibly increasing the number of I/Os done by a single disk). In contrast, P-CON, an algorithm which always matches its replacement decisions with those of the well-known single-disk optimal algorithm, MIN, can become fully serialized in the worst case. The behavior of an on-line algorithm with lookahead, P-LRU was analyzed. The performance of P-LRU is independent of the number of disks. Similar results can be shown to hold for P-FIFO, a parallel version of FIFO with lookahead.

## References

1. S. Albers. The influence of lookahead in competitive paging algorithms. In *Proc. 1st European Symposium on Algorithms LNCS, Springer Verlag, Berlin, Germany 1993*, pages 1–12, 1993.
2. L. A. Belady. A Study of Replacement Algorithms for Virtual Storage. *IBM Systems Journal*, 5:78–101, 1966.
3. S. Ben-David and A. Borodin. A New Measure for the Study of On-Line Algorithms. *Algorithmica*, 11:73–91, 1994.
4. P. Cao, E. Felten, A. Karlin, and K. Li. A Study of Integrated Prefetching and Caching Strategies. In *Proc. ACM SIGMETRICS Conference*, 1995.
5. P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-Performance Reliable Secondary Storage. *ACM Computing Surveys*, 26(2):145–185, 1994.
6. E. G. Coffman and P. J. Denning. *Operating Systems Theory*. Addison-Wesley, Englewood Cliffs, N.J., 1973.
7. D. Breslauer. On competitive on-line paging with lookahead. In *Proc. Symposium Theor. Aspects of Computer Science*, pages 593–603, 1996.
8. A. Fiat, R. Karp, M. Luby, L. McGeoch, D. D. Sleator, and N. E. Young. Competitive Paging Algorithms. *J. Algorithms*, 12:685–699, 1991.
9. E. Koutsoupias and C. H. Papadimitriou. Beyond competitive analysis. In *Proc. 35th IEEE Symposium on Foundations of Computer Science*, pages 394–400, 1994.
10. V. Pai, A. Schäffer, and P. Varman. Markov Analysis of Multiple-Disk Prefetching Strategies for External Merging. *Theor. Comp. Sci.*, 12:211–239, 1994.
11. D. Sleator and R. E. Tarzan. Amortized Efficiency of List Update and Paging Rules. *Comm. ACM*, 28(2):202–208, 1985.

This article was processed using the L<sup>A</sup>T<sub>E</sub>X macro package with LNCS style