Scheduling Multiple Flows on Parallel Disks *

Ajay Gulati[1] and Peter Varman[2]

[1] Department of Computer Science
[2] Department of Electrical Engineering and Computer Science
Rice University, Houston TX 77005, USA,
{gulati,pjv}@rice.edu

Abstract. We examine the problem of scheduling concurrent independent flows on multiple-disk I/O storage systems. Two models are considered: in the shared buffer model the memory buffer is shared among all the disks, while in the partitioned buffer model each flow has a private buffer. For the parallel disk model with $d > 1$ disks it is shown that the problem of minimizing the schedule length of $n > 2$ concurrent flows is NP-*complete* for both buffer models. A randomized scheduling algorithm for the partitioned buffer model is analyzed and probabilistic bounds on the schedule length are presented. Finally a heuristic based on static buffer allocation for the shared buffer model is discussed.

1 Introduction

Advances in disk drive and networking technologies and a sharp increase in data-intensive applications have revolutionized the architecture and usage paradigms of modern storage systems. Resource management issues have become increasingly important in data centers that must coordinate the operation of large numbers of concurrent devices and serve hundreds of gigabytes of data per second. Both commercial and scientific workloads require high-bandwidth access to large data sets that reside on shared storage facilities and are accessed by multiple applications. Shared storage servers are being increasingly proposed as a cost-effective solution for maintaining data repositories, which can take advantage of economies of scale and consolidated management. Sharing a storage server raises two issues: effective use of server resources and fair scheduling of individual clients. We examined the issue of performance isolation and providing QoS guarantees to individual clients in [1]. In this paper we address the problem of efficiently utilizing the resources of a shared storage system when servicing multiple concurrent flows.

Previous work on reducing latency in parallel I/O systems has dealt with the scheduling of a single flow to effectively exploit prefetching and caching from multiple disks; efficient algorithms are now known that maximize disk system throughput for a single flow (see [2–8] for example). Our work in this paper is a generalization of the parallel disk model [9] to the case when we have more than one concurrent flow simultaneously sharing the I/O system. The problems of sharing the parallel disks among multiple concurrent flows have only been recently considered [10–14, 1]. All of these works are

* Support by the National Science Foundation under Grant CCR-0105565 and the IR/D program is gratefully acknowledged.

mostly concerned with different models of storage virtualization and QoS-based resource allocation.

We consider two models of shared servers. In both models the disk subsystem is shared by all the flows. The models differ in how the memory buffer is allocated to the flows: in the shared buffer model all flows share a common buffer, while in the partitioned buffer each flow has its own private buffer. We show that in both models the problem of scheduling a set of requests from each flow to minimize the number of parallel I/O steps is NP-*Complete*. In contrast the case of a single flow is known to have efficient polynomial-time scheduling algorithms [2, 4–7]. We also show that the congestion-removal techniques of Leighton, Maggs and Rao [15] can be extended to obtain a randomized scheduling algorithm for the partitioned buffer model with probabilistically bounded schedule length. Finally for the shared buffer model, we present a heuristic that combines a novel static buffer allocation strategy with the randomized approach used in the partitioned buffer model.

The off-line scheduling of a single flow to obtain the minimum-length schedule has been extensively studied [16, 2, 7, 6, 17, 4, 5]. Polynomial time algorithms in the parallel disk model [9] and stall model [3] have been obtained in [6, 17, 4, 5] and [7, 2] respectively. However, the problem of scheduling multiple flows has not been formally considered previously in either parallel I/O model.

2 I/O models

When multiple applications share the disk system we look at two related models, one where the buffer is shared among all the flows, and the other in which the buffer is statically partitioned among them. In the Shared Disk Partitioned Buffer (SDPB) model, the disks are shared by all the flows but the buffer is partitioned among them. The most common scenario for such an organization is where the buffering is done in individual client nodes and the disks are accessed over a SAN. In the Shared Disk Shared Buffer (SDSB) model, the buffer is logically shared among all workloads. This may be either in the form of a centralized storage buffer or made up of distributed shared memory.

There are n independent *flows* (applications) that are simultaneously accessing the storage system. Each flow is abstracted by a *reference string* consisting of the ordered sequence of *blocks* that it accesses. Blocks need to be delivered to the application in the order in which they appear in the reference string. We focus on read-once reference strings where each block is unique, which model workloads like multimedia streaming. There is a fixed amount of buffer memory that the system uses for *prefetching*. A buffered block is consumed as soon as it becomes the first unconsumed block in its reference string. Each flow knows a subsequence of the reference string beyond its last access; this subsequence is the *lookahead window* for the flow. If the lookahead window includes the entire reference string then the schedule is said to be off-line. We assume that requests in one lookahead window are serviced before newly arriving requests are scheduled.

A model of the SDSB configuration is shown in Figure 1. Flow queues hold the requests in the currently visible portion of the reference strings. The high-level scheduler dispatches requests from the flow queues to the disk queues in accordance with the schedule that it constructs. In every parallel I/O step a number of requests, at most one

request for each disk, are dispatched by to the disk queues. Individual disks service the requests in their disk queues one at-a-time. The blocks read from the disks are placed in the memory buffer. In the SDPB model a block can only be placed in the partition belonging to the flow; in the SDSB model a common buffer of capacity M blocks is shared by all the flows. A flow will consume blocks in the order of the reference string from this buffer. Blocks that are fetched out-of-order of the reference string will be buffered until required by the flow. Each disk is free to reorder the requests in its disk queue to optimize physical access times by exploiting spatial locality in the data placement. The high-level scheduler must ensure that the requests it has dispatched to the disk queues at any time do not overflow the capacity of the buffer.

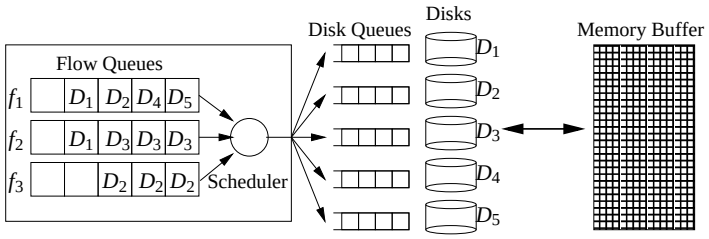

Fig. 1. Shared disk and Shared buffer configuration

If the memory buffer can hold data from all outstanding requests in a lookahead window, then the scheduling is trivial. The interesting case arises when the number of requests exceeds the buffer capacity. The scheduler must then determine which of the requests to dispatch at each step, so as to minimize the total number of I/O steps required to service all the requests.

We illustrate the scheduling problem by considering a single flow. Even for this special case the optimal schedule is not straightforward to construct. As an example we consider a single flow with reference string $\mathscr{R} = A_1\, A_2\, A_3\, A_4\, A_5\, B_1\, B_2\, A_6\, B_3\, B_4\, B_5\, A_7\, C_1\, C_2\, B_6\, B_7$. Here, A_i (B_i, C_i) stand for the i^{th} block from disk A (respectively disk B and disk C). Figure 2(a) shows the schedule created by an intuitive, greedy in-order prefetching strategy. A greedy prefetching algorithm tries to keep as many disks as possible busy at each I/O step; in-order fetching means it always chooses a block earlier in the reference string in preference to one that occurs later. The schedule assumes a shared buffer of size $M = 6$ blocks. In step 1 of Figure 2(a), blocks A_1, B_1 and C_1 are fetched from the three disks respectively. A_1 is consumed, and a request for the next block A_2 is made. Since there are four free blocks in the buffer at this time, the system will prefetch the next block from disks B and C along with A_2 in the next I/O step. On making the reference to A_3, the buffer holds 4 blocks; hence it cannot prefetch blocks from both B and C, but must choose to prefetch from one or the other of the disks. A greedy in-order prefetching scheduling algorithm will fetch B_3 in preference to C_3 since it occurs earlier in the reference string. Continuing in this manner, we obtain the schedule of length 9 to service the entire reference string. In contrast, Figure 2(b) shows the optimal-length schedule for this reference string consisting of 7 parallel I/O steps. When the input includes multiple flows there is an additional dimension to the problem. Not only does the scheduler need to determine which disks to fetch from in an I/O step, it also needs to decide which flow should be allocated a disk at each I/O step.

IOs→	1	2	3	4	5	6	7	8	9
Disk A	A_1	A_2	A_3	A_4	A_5	A_6	A_7	-	-
Disk B	B_1	B_2	B_3	-	-	B_4	B_5	B_6	B_7
Disk C	C_1	C_2	-	-	-	-	-	-	-

(a)Greedy Inorder IO schedule

IOs→	1	2	3	4	5	6	7
Disk A	A_1	A_2	A_3	A_4	A_5	A_6	A_7
Disk B	B_1	B_2	B_3	B_4	B_5	B_6	B_7
Disk C	-	-	-	-	-	C_1	C_2

(b)Optimal IO schedule

Fig. 2.

2.1 Scheduling Multiple Flows

The problem of constructing a minimum-length schedule for two or more reference strings in the SDPB and SDSB models is formally defined below. In the following, the number of fbws is denoted by n, the number of disks is d and the set of disks is denoted by $\mathscr{D} = \{D_1, D_2, \cdots D_d\}$.

Partitioned Buffer: The problem will be denoted by SDPB(n, d, Σ, L). $\Sigma = [m_1, m_2, \cdots m_n]$ is a vector specifying buffer allocations where m_i is the buffer size of fbw f_i, and L is a desired bound on the schedule length. The decision problem is to determine if there exists a schedule that requires less than or equal to L I/O steps. Note that in a valid schedule, each reference string consumes blocks in the order specified, but there is no restriction on the ordering across strings. Furthermore, fbw f_i can hold at most m_i blocks in the buffer at any time.

Shared Buffer: The problem will be denoted by SDSB(n, d, M, L), where the memory buffer of size M blocks is shared among all fbws, and L is a desired bound on the schedule length. The decision problem is to determine if there exists a schedule that requires less than or equal to L I/O steps. Note that in a valid schedule the buffer can hold at most M blocks at any time.

3 NP-completeness

In this section, we present the proofs for NP-completeness of the scheduling problems for the two models. The input length is the sum of lengths of all input reference strings. Each reference string will be assumed to be fully enumerated by the sequence of blocks that it accesses, so that a reference string with l requests is assumed to require $\Omega(l)$ bits to represent. That is we do not assume any form of compression of the input reference string. This assumption does not limit the generality of our model, and reflects the natural encoding in the application domain.

We will use the following known NP-complete problem, 3-Partition, to show NP-completeness of our problems.

Definition 1. 3-Partition*: Given a multi-set* $\mathscr{A} = \{a_1, a_2, ..., a_{3w}\}$ *and a positive integer* B*, such that* $\forall i, 1 \leq i \leq 3w, B/4 < a_i < B/2$*, and* $\sum_{i=1}^{3w} a_i = wB$*. Does there exists a partition of* $\mathscr{A}$ *into* w *subsets* $\{A_1, A_2, ..., A_w\}$*, such that each* A_s *has exactly 3 elements and* $\sum_{a_i \in A_s} a_i = B$*,* $\forall i, 1 \leq s \leq w$*.*

Lemma 1. *3-Partition is NP-Complete even when the input is assumed to be of size* wB *[18].*

Our first result is that the SDPB problem is NP-complete. This result actually follows by showing the correspondence between the special case of SDPB when each

$m_i = 1$ and the job shop scheduling problem that has been extensively studied in the literature (see [15, 19, 20]). The proof is omitted due to lack of space. The complexity of the SDSB problem however does not follow directly from job-shop scheduling or its variants, and we prove its NP-*Completeness* from first principles. To specify a reference string we will use the following notation: $r \times D_j$ will mean r distinct consecutive requests to disk D_j. The concatenation of two reference strings α and β will be denoted by $\alpha * \beta$, and the concatenation of α to itself s times will be denoted by α^s.

Theorem 1. SDPB*(n, d, Σ, L), n arbitrary, is NP-complete.*

Theorem 2. *SDSB(n, d, M, L), n arbitrary, is NP-complete.*

Proof. It is easy to verify that the problem is in NP.

We reduce 3-Partition to SDSB(n, d, M, L). Given w, B and a multi-set $A = \{a_1, a_2, ..., a_{3w}\}$, we construct an instance of SDSB(n, d, M, L) with $n = 3w+1$, $M = cB$ for some constant c, and $L = 2w\alpha$, where $\alpha = B + M - 1$. The reference strings $R_1, R_2, \cdots, R_{3w}$, R_{3w+1} are defined as follows.

$$R_i = (a_i \times D_1) * (a_i \times D_2),\ 1 \le i \le 3w$$
$$R_{3w+1} = (\alpha \times D_2) * [((\alpha + M - 1) \times D_1) * ((\alpha + M - 1) \times D_2)]^{(w-1)} * (\alpha + M - 1 \times D_1)$$

For all i, $1 \le i \le 3w$, the schedule for R_i consists of a_i fetches from D_1 followed by a_i fetches from D_2 with some overlap possible between the fetches from the two disks. The amount of overlap can be no more than $M-1$. R_{3w+1} consists of α blocks from D_2 followed by a repeating pattern consisting of $\alpha + M - 1$ blocks from D_1 followed by $\alpha + M - 1$ blocks from D_2; this pattern is repeated $w-1$ times, and is finally followed by $\alpha + M - 1$ blocks from D_1. Individual schedules for R_i are shown in Figure 3. There can be at most $M-1$ blocks prefetched at any time. Hence, in order for R_{3w+1} to complete within the schedule $L = 2\alpha w$, the following is necessary: in the interval $[(k-1)\alpha,\ k\alpha]$, $k = 1, 2, \cdots 2w$, α blocks of R_{3w+1} must be fetched from D_2 if k is odd, and from D_1 if k is even. In order to meet this schedule $M-1$ blocks of R_{3w+1} must be prefetched from the other disk in every interval $[(k-1)\alpha,\ k\alpha]$; these prefetched blocks are shown by the vertical stripes in the unshaded regions. This implies that there are no free buffers available for prefetching blocks of R_i, $1 \le i \le 3w$, which must be fetched one block at a time.

Now we will show that there exists a 3-Partition if and only if a schedule of length L is possible.

Case 1: Suppose a partition of A into w subsets $\{A_1, A_2, A_3, \cdots, A_w\}$ exists, such that $A_k = \{A_{k_1},\ A_{k_2},\ A_{k_3}\}$, $\sum_{i=1}^{3} a_{k_i} = B$, then the following schedule of length L is certainly possible.

In Figure 4, for each unshaded region $[2(k-1)\alpha,\ (2k-1)\alpha]$, $1 \le k \le w$, requests corresponding to A_k, in particular reference strings R_{k_1}, R_{k_2} and R_{k_3}, are scheduled on D_1. The corresponding requests from D_2 are scheduled in the unshaded interval $[(2k-1)\alpha,\ 2k\alpha]$. In the unshaded interval $[2(k-1)\alpha,\ (2k-1)\alpha]$, we fetch the B blocks belonging to R_{k_1}, R_{k_2}, and R_{k_3} on D_1, and prefetch the next set of $M-1$ blocks from R_{3w+1} on D_2; their relative ordering within the region is unimportant. Similarly

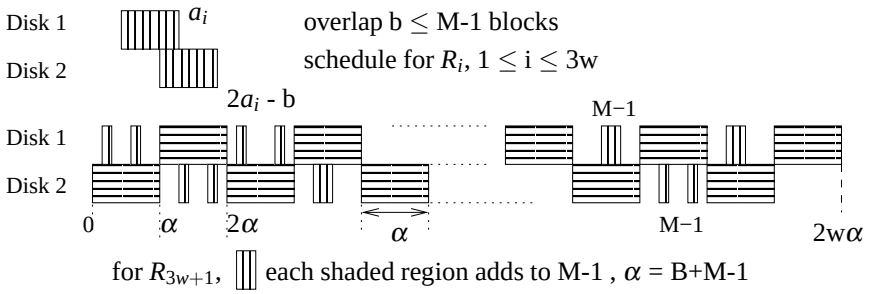

Fig. 3. Individual schedules for SDSB model

the unshaded region, $[(2k-1)\alpha, 2k\alpha]$ is used to schedule the B blocks on D_1 belonging R_{k_1}, R_{k_2} and R_{k_3}, and on D_2 (except for the last unshaded region) prefetch the next $M-1$ blocks from R_{3w+1}, as shown in figure 4. Hence if a 3-Partition exists, so does a schedule with length L.

Case 2: If a schedule of length $L = 2w\alpha$ exists, then we show that set $\mathscr{A}$ can be partitioned into w desired subsets $A_1, A_2, \cdots, A_w$. First observe that in a valid schedule of length $2w\alpha$, R_{3w+1} needs all the shaded regions in Figure 4, and $M-1$ of each unshaded interval (except the last one of D_2) for fetching its blocks. Consequently the remaining reference strings, R_i, $1 \leq i \leq 3w$, must be scheduled within the unshaded regions; since in any of these intervals, $M-1$ of the time is used to prefetch blocks of R_{3w+1}, the blocks of R_i, $1 \leq i \leq 3w$, must be fetched in time $\alpha - (M-1) = B$. The conditions are exactly the same as in Theorem 1, where the short reference strings needed to be scheduled in intervals of length B on each disk. Using identical reasoning, we can conclude that the scheduling of the R_i, $1 \leq i \leq 3w$, in the unshaded intervals induces a 3-Partition of $\mathscr{A}$.

Note that the sum of lengths of all reference strings is $2wB + 2w\alpha + (2w-2)(M-1) = 4wB + (4w-2)(M-1)$ which is polynomially related to wB when $M = \Theta(B)$.

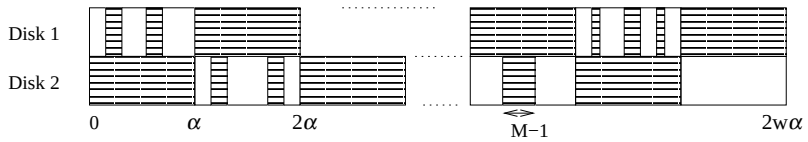

Fig. 4. A feasible schedule with length 2wα

4 Randomized Scheduling

In the previous section we showed that the general problem of scheduling multiple reference strings with the minimum number of I/O's is NP-*complete* for both the SDPB and SDSB models. In this section we look at the complexity of the optimal scheduling problem in more favorable situations. In the first situation we show how the fundamental result of Leighton, Maggs and Rao [15] for congestion removal in networks using randomization can be applied to the SDPB scheduling problem. This results in an algorithm with probabilistically bounded schedule length. For the SDSB problem we describe a heuristic scheme based on quasi-static memory allocation followed by congestion removal via randomization.

4.1 SDPB Model

We first consider the SDPB model where each fbw has its own independent buffer. For each fbw f_i with buffer size m_i we use the optimal scheduling algorithm [4, 5] to find a schedule that minimizes the number of I/O steps required to fetch all the blocks of f_i, when executed in isolation. Let T_i be the number of steps required by the optimal schedule of f_i using buffer m_i. Let $T = max_{1 \leq i \leq n}\ T_i$ be the maximum schedule length of any of the fbws. Note that T is a *lower bound* on the length of any schedule for the SDPB problem instance.

At any time step t between 1 and T_i, the schedule for f_i indicates which disks will be active fetching blocks of f_i at that time step. If we overlay the schedules of each f_i, $1 \leq i \leq n$, then *disk conflicts* may occur. That is at time step t, the schedules of two or more fbws may access the same disk D_k. A simple way to handle the congestion is to simulate each step of the overlay schedule by a number of substeps, where only one block is fetched from any disk in each substep. Let $c(i,t)$ indicate the congestion (number of contending fbws) for disk D_i at time step t. Let $c^*(t) = max_{1 \leq i \leq d}\ c(i,t)$ be the maximum congestion on any disk at time step t. Then the accesses at step t can be simulated in a conflict-free manner in $c^*(t)$ steps. Hence the entire schedule can be simulated in $\Sigma_{1 \leq t \leq T}\ c^*(t)$ steps. Since in the worst-case $c^*(t)$ can equal d, this gives a worst-case bound of Td for the length of the schedule using this strategy.

We now present a randomized scheduling method for SDPB with bounded performance. The strategy uses the fundamental idea of Leighton, Maggs and Rao [15] who showed how to remove congestion in a routing network using randomization. The relation of the network routing problem to *shop scheduling* problems was shown in [20]. In the *job shop scheduling* problem there are d machines and n jobs; each job is made up of an ordered sequence of operations which must be serviced in order. Each operation must be assigned to a particular machine depending on the operation. A machine may work on only one operation at any time step, and each job may be processed by only one machine at a time. A special case of job shop where each job is processed exactly once on each machine is known as the *flow shop* problem.

The routing problem in [15] is a special case of fbw-shop scheduling where each operation requires unit time. In [20] the technique of [15] was generalized to allow operations to have different and arbitrary lengths, and to allow a job to use the same machine several times. However, each job could only be processed by one machine at any time. It is possible to visualize the SDPB problem considered in this paper within this framework. Associate fbws with jobs, disks with the machines, and each block of a fbw as a unit-time operation of the job. Then the special case when the memory assigned to each fbw is one ($m_i = 1$), corresponds exactly to the job shop scheduling problem. When $m_i > 1$, SDPB differs from the standard job shop problem in that several machines (disks) may service the operations (fetch blocks) of the same job (fbw) at the same time step. That is, there is parallelism among the operations of a single job which is disallowed in the standard job shop scheduling model.

We show below that the techniques of [15, 20] can be extended to apply to this more general model of SDPB as well. Let N be the total number of blocks in all fbws combined. Recall that the length of the optimal schedule for fbw f_i with memory allocation m_i is denoted by T_i, and that $T = max_{1 \leq i \leq n}\ T_i$ is the maximum length of the schedule

for any fbw. For any buffer size m_i the optimal schedule for f_i can be constructed using the algorithm in [4, 5]. Let the total number of blocks fetched from disk D_i be B_i, and let $B = max_{1 \le i \le d} B_i$ denote the maximum number of blocks fetched from any disk. Finally let the length of the optimal schedule for SDPB be T^*. Then $T^* \ge T$ and $T^* \ge B$; so $T + B \le 2T^*$. We show that with high probability all accesses can be scheduled to complete in $cT^* \log N$ steps, for a constant c.

The scheme parallels that in [15, 20]. The schedule for fbw f_1 begins at time step 1. The start time for the schedule for each fbw f_i, $i > 1$ is staggered by an integer number of steps randomly chosen between 1 and B. The n schedules with start times between 1 and B when overlaid result in a schedule of length at most $T + B$. However, there may be contention for disks at different time steps, depending on how the rectangles representing the schedules line up. We show below that the worst-case disk conflict is small with high probability.

Theorem 3. *Let N be the total number of blocks in the reference strings. It is possible to find a schedule for SDPB that with high probability is within a factor $c \log N$ of the optimal length, for a constant c.*

Proof. The overlapped schedules obtained by staggering the fbws is of length no more than $T + B$. We show that with high probability the maximum load on any disk is upper bounded by $O(\log N)$.

Consider an arbitrary time step u and an arbitrary disk s. At most B blocks are candidates for scheduling on disk s at time step u. The probability of any of the candidate blocks being scheduled in that time step is $1/B$, since the random offset of each fbw gives the block a probability $1/B$ of being scheduled in that slot. Hence the probability that k or more blocks in are scheduled in this slot is upper bounded by $({}^{B}C_k)(1/B)^k \le (eB/k)^k (1/B)^k = (e/k)^k$. The probability that k or more blocks are scheduled in any of the disks at time step u is therefore less than $d(e/k)^k \le N(e/k)^k$. If we choose $k = 2e \log N$, then this probability is upper bounded by $(1/N^{2e-1})$. Now $T \le N$ and $B \le N$, and so we have $T + B \le 2N$. Since there are at most time $T + B$ steps in the schedule, the probability that the maximum load on a disk at any time step is greater that $k = 2e \log N$ is upper bounded by $2N * (1/N^{2e-1}) < 1/N^3$. Hence with high probability each time step in the schedule of length $T + B$ can be simulated contention free in $k = O(\log N)$ steps. Since $T + B \le 2T^*$, the overall schedule is of length $cT^* \log N$, for a constant c, with high probability.

4.2 SDSB Model

For the SDSB model we first statically partition the buffer among the fbws so that in the absence of any disk contention, it minimizes the maximum schedule length of any fbw. We then proceed exactly as in the SDPB model. That is first construct the shortest schedule for each fbw individually based on the storage it has been allocated, and then stagger their starting times randomly to reduce disk contention. To capture phases of the workload the buffer can be reparititoned for small disjoint sections of the reference string.

Let $T_i(k)$, $1 \le i \le n$, denote the length of the optimal schedule of fbw f_i with a buffer allocation of k blocks. The buffer allocation is denoted by a vector $\mathscr{M}$ =

$[m_1, m_2, \cdots, m_n]$, $\sum_{i=1}^{n} m_i = M$, where m_i is the amount of buffer allocated to f_i. The aim is to find the vector $\mathscr{M}$ that minimizes the completion time of all flows; *i.e.* minimizes $max\{T_i(m_i) \mid \sum_i m_i = M, 1 \leq i \leq n, \}$. A straightforward algorithm will evaluate $T_i(k)$ for all f_i, $1 \leq i \leq n$, and for all values of k, $1 \leq k \leq M$. This can be done in $\Theta(nML)$ time, where L is the length of the longest reference string. It then considers all n-partitions of M, and chooses the partition that minimizes the maximum length schedule. Since there are $\Theta(M^{n-1})$ such partitions, the total time required is $\Theta(nML + nM^{n-1})$; this straightforward algorithm is clearly infeasible in practice. We present a more efficient algorithm that runs in time $\Theta(Ln \log^{\log n} M)$.

The algorithm for the case $n = 2$ is shown in Figure 5. The main idea is as follows. We compute $T_1(m)$ and $T_2(M-m)$. If $T_1(m)$ is the larger of the two, we allocate more memory to f_1 and less to f_2, or vice versa if $T_2(M-m)$ is larger. By reallocating the excess buffer between the two strings, using a binary-search like pattern, we can converge on the optimal in a logarithmic number of probes.

```
l = 1; h = M − 1
while (l < h)
Note: L_n^m is the minimum schedule length of flow n using m buffers
   m = (l + h)/2; /* Probe mid point of memory range */
   d1 = L_1^l - L_2^{M−l};   d2 = L_1^m - L_2^{M−m};
   if (d1 × d2 < 0)  h = (m − 1)
   else if (d1 × d2 > 0)  l = (m + 1);
   else return m
return l;
```

Fig. 5. $O(L \log M)$ algorithm to optimally allocate memory to two flows

For n reference strings, we use a recursive version of the above method. Divide the strings into two sets with $n/2$ strings in each. Assign m_1 buffer blocks to the first set and m_2 buffer blocks to the other partition. Initially, $m_1 = m_2 = M/2$. Recursively, find the best schedule length for each partition using m_1 and m_2 buffers respectively. If the first set has the longer schedule length, then move buffers from set 2 to set 1, or vice versa as necessary, exactly as if there were two strings. Continue the process till one converges on the best allocation. We will require to do $\Theta(\lg M)$ probes to each of the subproblems to find the best allocation. Formally this recursion can be represented as:

$$T(n,M) \leq c_1 \lg M \ (2\, T(n/2,M)) \text{ and } T(1,M) \leq c_2 L \tag{1}$$

Solving the recurrence, we get:

$$T(n,M) \leq c_1^k \cdot \log^k M \cdot 2^k \cdot T(n/2^k, M) = \Theta(n\, L \lg^{\lg n} M) \tag{2}$$

5 Conclusions

In this paper, we present an analytical model to study scheduling of multiple flows in a parallel I/O system. We show that obtaining a minimum length schedule in this case is NP-*complete* for both the partitioned buffer and shared buffer models. A randomized algorithm based on the congestion removal technique of [15] was analyzed and the dilation in the schedule length for the SDPB model was shown to be bounded by a factor logarithmic in the number of blocks, with high probability. A heuristic for the SDSB model was presented based on quasi-static partitioning of the buffer among the flows,

followed by randomized congestion removal. An interesting open issue is the complexity of the problem with a fixed number of reference strings, and a variable number of disks or buffer size. Another interesting problem is to come up with approximation algorithms for the SDSB model with guarantees on performance.

References

1. Gulati, A., Varman, P.: Lexicographic QoS scheduling for parallel I/O. In: Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures, Las Vegas, Nevada, United States, ACM Press (2005) 29–38
2. Albers, S., Garg, N., Leonardi, S.: Minimizing stall time in single and parallel disk systems. J. ACM **47** (2000) 969–986
3. Cao, P., Felten, E.W., Karlin, A.R., Li, K.: A study of integrated prefetching and caching strategies. In: Proc. of the Joint Intl. Conf. on Measurement and Modeling of Computer Systems, ACM Press (1995) 188–197
4. Kallahalla, M., Varman, P.J.: PC-OPT: optimal offline prefetching and caching for parallel I/O systems. IEEE Transactions on Computers **51** (2002) 1333–1344
5. Kallahalla, M., Varman, P.: Optimal read-once parallel disk scheduling. Algorithmica, (A preliminary version appeared in the 6th ACM IOPADS, 1999) (2005)
6. Hutchinson, D.A., Sanders, P., Vitter, J.S.: Duality between prefetching and queued writing with parallel disks. In: Proceedings of the 9th Annual European Symposium on Algorithms, Århus, Denmark, Springer-Verlag (2001) 62–73
7. Kimbrel, T., Karlin, A.R.: Near-optimal parallel prefetching and caching. SIAM J. Comput. **29** (2000) 1051–1082
8. Patterson, R.H., Gibson, G., Ginting, E., Stodolsky, D., Zelenka, J.: Informed prefetching and caching. In: Proc. of the 15th ACM Symp. on Operating Systems Principles. (1995) 79–95
9. Vitter, J.S., Shriver, E.A.M.: Optimal disk I/O with parallel block transfer. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing, ACM Press (1990)
10. Jin, W., Chase, J.S., Kaur, J.: Interposed proportional sharing for a storage service utility. SIGMETRICS Perform. Eval. Rev. **32** (2004) 37–48
11. Huang, L., Peng, G., Chiueh, T.C.: Multi-dimensional storage virtualization. SIGMETRICS Perform. Eval. Rev. **32** (2004) 14–24
12. Lumb, C., Merchant, A., Alvarez, G.: Façade: Virtual storage devices with performance guarantees. File and Storage technologies (FAST'03) (2003) 131–144
13. Gulati, A.: Scheduling with QoS in parallel I/O systems. Master's thesis, Rice University, Department of Computer Science (2004)
14. Gulati, A., Varman, P.: Scheduling with QoS in parallel I/O systems. In: International Workshop on Storage Network Architecture and Parallel I/Os. (2004)
15. Leighton, F.T., Maggs, B.M., Richa, A.W.: Fast algorithms for finding O(congestion + dilation) packet routing schedules. In: Combinatorica. Volume 19. (1999) 375–401
16. Barve, R.D., Kallahalla, M., Varman, P.J., Vitter, J.S.: Competitive parallel disk prefetching and buffer management. Journal of Algorithms **36** (2000) 152–181
17. Kallahalla, M., Varman, P.J.: Optimal prefetching and caching for parallel I/O systems. In: Proc. of 13th ACM Symp. on Parallel Algorithms and Architectures, ACM press (2001)
18. Garey, M.R., Johnson, D.S.: Computers and intractability; a guide to the theory of NP-completeness. W. H. Freeman (1979)
19. Gonzalez, T., Sahni, S.: Flowship and jobshop schedules: Complexity and approximation. Operations research **26** (1978) 36–52
20. Shmoys, D.B., Stein, C., Wein, J.: Improved approximation algorithms for shop scheduling problems. In: SIAM Journal on Computing. Volume 23. (1994) 617–632