Impact of Data Placement on Parallel I/O Systems*

J. B. Sinclair, J. Tang, P.J. Varman, B. R. Iyer†

Abstract

The I/O performance of several concurrent external merge jobs sharing a parallel I/O system is studied. The placement of the runs is found to have a significant impact on performance. Placements leading to serialization among jobs were identified and analyzed, and solutions discussed. Contrary to the behavior of a single merge, increasing the buffer size in this situation may actually degrade performance.

1 Introduction

This paper studies the impact of data placement on the performance of a parallel I/O system. The workload consists of a number of independent and concurrent external-merge [3] jobs. The problem of deciding on which disks the data of different jobs should be placed arises frequently in high-performance database management systems executing multiple, concurrent sort queries. The results show that external merge sorting programs can achieve significant speedup through better data layout schemes, and demonstrate the importance and relevance of providing this capability to application processes. We investigate how the placement of the input and output runs of each merge job affects the disk utilizations, and the job completion times.

The reader is referred to [1, 10, 5, 8, 7, 6, 11] for external merging methods for a single job, using multiple disks. Here we study the interaction of several concurrent merge jobs in a parallel I/O system.

2 Run Placement Policies

In this section, we compare the performance of data layout policies for multiple, independent merge jobs. We observe that the dynamic interaction of identical, symmetrically placed jobs can sometimes lead to a race among them, resulting in significant disk serialization. A model of such racing behavior is analyzed in Section 3.

2.1 Simulation Model

The workload consists of J identical merge jobs. The input of a job consists of several sorted runs, and its output is a single sorted run. We assume (in keeping with current high-performance database systems like DB2, SQL/DS, etc.) that each run is placed entirely on a single disk without striping.

The system model consists of CPUs, a set of independent disks and a disk cache. The merge is simulated using the random-block-depletion model [4]. The block to be depleted is chosen with uniform probability from all non-empty runs. The CPU depletes this block and requests the next one from the same run. If this block is in the cache the request is immediately satisfied; else the CPU is blocked until the I/O for the block is completed. Reads are done in units of several physically contiguous blocks (N), to amortize disk latency. Anticipatory prefetching is employed. A read of N blocks is initiated when the first block of the *previously fetched* N blocks is processed. For each depleted block, a write block is generated; when M blocks have been buffered, they are written to disk in one I/O operation. If a free cache block is unavailable when requested, a job waits until one is released. To focus on the I/O performance, we assume infinite-speed CPUs here. A linear seek model with seek time of 0.04 ms/cylinder, average rotational latency of 8.33 ms, and block transfer time of 1.024 ms/block are used. A job has 20 runs of 1000 blocks each. The read and write blocking factors are $N = 12$ and $M = 40$, respectively. Our study was carried out using the YACSIM simulation package [2].

2.2 Performance of Placement Policies

The four data layout policies studied are shown in Figure 2.1 for the case of $J = 2$ and $D = 5$. The input runs of a job are indicated by Read and its output run by Write. Disks containing only input (output) runs are called *read (write) disks*; disks with both input and output runs are called *shared disks*. No disk will hold more than one output run. To utilize all disks efficiently the I/O load should be divided as equally as possible among the D disks. Since every output run is placed on a separate disk, we need only consider the placement of the input runs.

- **Case 1**: *Dedicated Write Disk for Each Job.* Disk k, $0 \leq k \leq J-1$, is used exclusively for the output run of job k; the input runs of each job are spread evenly among the remaining $D-J$ read disks.
- **Case 2**: *Intra-job Separate Read and Write Disks* Job k , $0 \leq k \leq J-1$, uses disk k for its output run, and its input runs are spread evenly among the remaining $D-1$ disks.
- **Case 3:** *Intra-job Shared Read and Write Disks* This allocation is obtained by permuting the allocation of Case 2, such that the input runs of job k, $0 \leq k \leq J-1$, are moved from disk $(k-1)$ *mod* J to disk k.

*Partially supported by a grant from IBM Corporation and by NSF and DARPA Grant CCR 9006300.

†The first three authors are with the ECE Department, Rice University, Houston, TX 77251. B. Iyer is with IBM, DBTI, P.O. Box 49023, San Jose, CA, 95161.

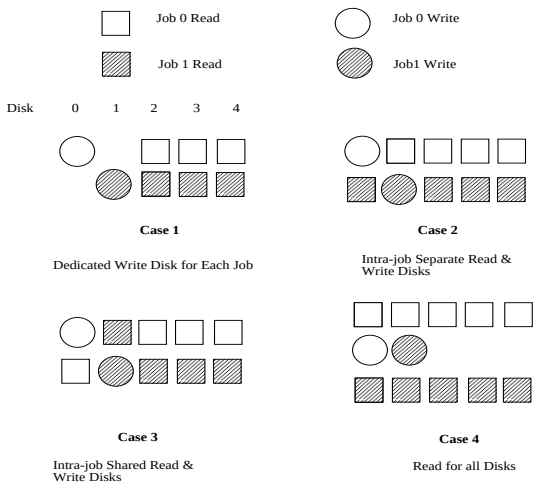

Figure 2.1: Run placement policies, D = 5 and J = 2.

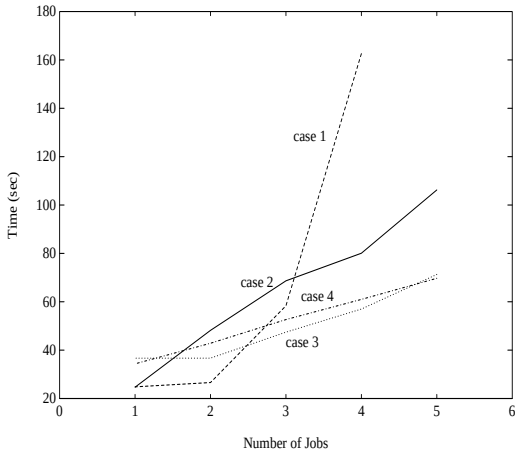

Figure 2.2: Completion time of all jobs

- **Case 4:** *Read from All Disks* A job uses all D disks for input and one disk for its output run.

Case 1 is motivated by the fact that a single job is write bound; hence an output disk is not loaded further. Figure 2.2 shows that as J increases, and the system becomes input-bound, the performance degrades steeply. This allocation is suitable if it is known in advance that there will be only a small ($\leq D/2$) number of jobs. Case 2 has the advantage that a job's allocation is independent of other jobs, and can therefore be easily implemented in the case of staggered job arrivals. Note that the increase in the time for increasing J *cannot* be accounted for by just the increased load on a disk. Despite symmetry in the run placement jobs progress at different rates, with a significant amount of serialization among all jobs. The race conditions occur because the data placement strategy allows a large number of writes from a single job to be queued together, effectively blocking other jobs which

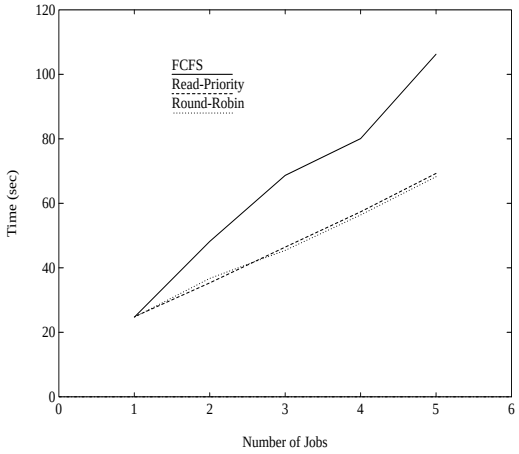

Figure 2.3: Completion time with different disk scheduling policies

attempt to read from the same disk. We present solutions for the race are in Section 2.3, and study an analytic model in Section 3. In Case 3, the input runs of the jobs are placed so that every write disk also contains input runs from the *same* job. This data placement scheme prevents a job from racing ahead, since the effect of its queued writes would slow down its own reads as well. Figure 2.2 shows the significant performance improvement of Case 3 over that of Case 2. A disadvantage of Case 3, however, is that the number of jobs must be known prior to laying out data on the disks. In Case 4 all input runs are distributed evenly on the set of D disks. Because each disk has some number of input runs from each job, no race can develop. The performance is consistent, albeit suboptimal, over the entire range.

2.3 Race Condition and Solutions

While appropriate run placement can sometimes be used to eliminate the race condition (Case 3), global constraints (staggered arrivals) may make the method unsuitable. We discuss disk-scheduling and buffer management policies to control this behavior.

One mechanism is to use either a Round Robin or Read Priority disk scheduling policy. These policies allow reads to get service even though there may be a large backlog of pending writes at that queue. Figure 2.3 shows that both policies are very effective in improving the performance. Simulation data also suggests that the performance is quite insensitive to the buffer size.

A second mechanism for avoiding races is to partition the cache and give each job a fixed, limited amount of buffer space. When a job fills up its allocated space it automatically must wait until some of its writes are serviced. Figure 2.4 shows something quite unexpected; that is, increasing buffer size by even a small amount may degrade performance significantly. This phenomenon is contrary to our expectation that more buffer space improves performance.

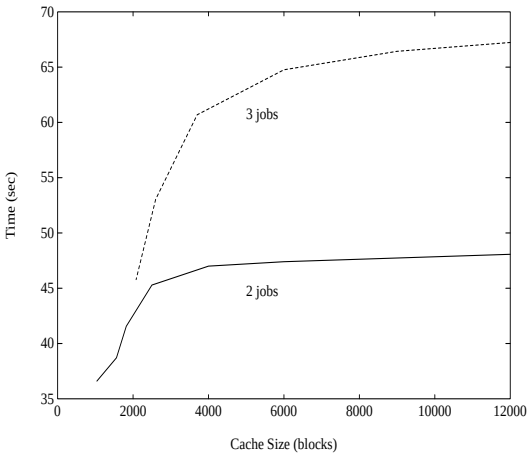

Figure 2.4: Completion time versus Buffer Size.

3 Analysis of Race Conditions

The observed race condition occurs even though the run placement strategy balances the read and write demands equally among all of the disks. In this section, we present simple analytic models of concurrent merge tasks that exhibit race conditions. We show that the race conditions occur because the data placement strategy allows a large number of writes from a single task to be queued together, effectively blocking other tasks which attempt to read from the same disk. The analysis permits us to predict potential races based on I/O system parameters, including the number of blocks read or written at a time, and the average access time per block for a read and a write.

We first analyze a simple model of a two-disk, two-job system which may exhibit a race condition, depending on how many blocks are read or written on each disk access, as well as other factors. The race condition causes the two jobs to be effectively serialized, with one finishing far behind the other.

Consider a system consisting of two disks, d_0 and d_1, and two jobs, j_0 and j_1. j_0 reads from d_0 and writes to d_1, while j_1 reads from d_1 and writes to d_0. Scheduling at each disk is FCFS. Jobs block on reads but not on writes. To simplify the analysis, we assume times for reads and writes that are functions only of the number of blocks read or written at one time. Let

b_r = read blocking factor
b_w = write blocking factor
r = read time per block
w = write time per block
$f = \frac{w}{r}$

Assume that at time $t = 0$ each disk queue contains one read job, d_0 has no queued write demand, and d_1 has N_0 total write service demand. Let T_1 be the time required for d_1 to service the demand queued at $t = 0$. Then

$$T_1 = N_0 + rb_r$$

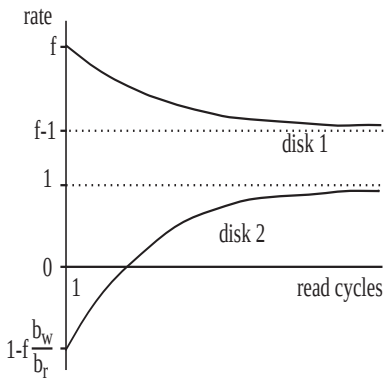

Figure 3.5: $f > 2$

Let N_1 be the total write service demand that arrives at d_1 during the interval $(0, T_1)$. During $(0, T_1)$, d_1 services only one read request. As soon as j_1 has this read request serviced, it sends $\frac{b_r}{b_w}$ write requests to d_0. The rate at which d_0 completes reads is $\frac{[T_1 - wb_r]}{T_1} \frac{1}{rb_r}$. The total write service demand N_1

$$\begin{aligned} N_1 &= T_1 \frac{[T_1 - wb_r]}{T_1} \frac{1}{rb_r} \frac{b_r}{b_w} wb_w \\ &= N_0 f + wb_r(1 - f) \end{aligned}$$

For the system to be stable, $N_1 \leq N_0$, or $N_0(1 - f) \geq wb_r(1 - f)$. Thus, the conditions for stability are $f \leq 1$ and $wb_r < N_0$ or $f > 1$ and $wb_r > N_0$. wb_r, the critical threshold for N_0, is the amount of write demand generated for each read completed.

This race condition arises because j_0, by writing to d_1, interferes with the only mechanism that can slow j_0, that is, writes by j_1 to d_0. This suggests that the race condition might not exist in a system with more jobs and disks, so that j_0's writes to a disk do not interfere directly with the job which writes to d_0. However analysis shows that, while adding another disk and another job does change the system behavior, a race condition may still exist, but it will manifest itself in an markedly different way.

Consider a system with three disks, d_0, d_1, and d_2, and three jobs, j_0, j_1, and j_2. j_0 reads from d_0 and writes to d_1, j_1 reads from d_1 and writes to d_2, and j_2 reads from d_2 and writes to d_0. As before, scheduling is FCFS, and jobs block only on reads. Assume that the queues for d_0 and d_1 contain no writes and one read each, and d_2's queue contains N_0 total write demand (from j_1) and one read (from j_2).

A *read cycle* is the interval during which the initial demand in d_0's queue is serviced. In [9] we show that at the end of the read period, d_0's queue still contains a single read (perhaps partially completed), d_2's queue contains a single read (just beginning service), and d_1's queue contains a read plus a number of writes corresponding to a total write demand of N_1. This is similar to the state in which the read period began, but with each disk ready to play a different role in a

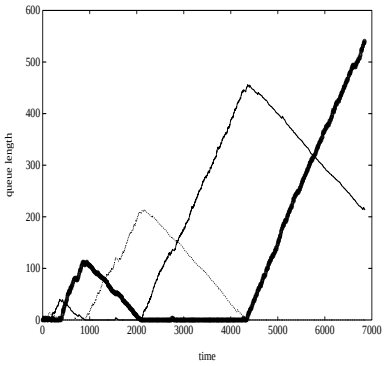

Figure 3.6: Simulation results for 3 disks, 3 jobs, f = 3

new read period just beginning.

The graph in Fig. 3.5 illustrates one (of three) possible scenarios for the sequence of values $\{N_0, N_1, N_2, \ldots\}$, the total write demand in the backlogged queue at the beginning of successive read periods. The curve labeled disk 1 (disk 2) shows the rate at which the service demand changes at the disk with increasing (decreasing) queued service demand during a read period. For $f > 2$ (Fig. 3.5), the two rate curves do not intersect. Consequently, regardless of N_0, N_1 will be greater than N_0, and when N_0 is large enough, $N_1 \approx (f-1)N_0 > N_0$. For $2 > f > 1$ the rate curves intersect and hence the system has a steady-state behavior in which the lengths of the read periods are all approximately the same. For $f < 1 - f\frac{b_w}{b_r}$, the two rate curves never intersect, and the rate at which N_0 is depleted is always greater than the rate at which the backlog increases on d_1. Consequently, queues are always quite short.

Simulation of a similar model with uniformly distributed rotational latencies gives results which agree closely with this deterministic analysis. Fig. 3.6 illustrate this for $f = 3$. The y-axis is the number of writes in a disk queue. The solid, dotted, and bold lines correspond to d_0, d_1, and d_2, respectively. Fig. 3.6 corresponds to the situation shown in Fig. 3.5. Successive peaks eventually grow by a factor of 2 as predicted.

During a read period one job receives virtually all the service capacity of its read and write disks, one job uses the remaining disk almost 100%, and the remaining job is essentially blocked. In the next read period, this last job receives nearly 100 % of the service from its read and write disks, and so forth. The jobs alternate taking the lead in the number of reads completed, and the job that finishes first is determined by how this sequence first started and the total number of reads to be performed.

4 Summary

When several concurrent external merge jobs share a parallel I/O system, the placement of the runs on the disks has a significant impact on the I/O performance. The difference in the time for reads and writes (due to differing blocking factors or input parallelism) may result in a race among the jobs. resulting in disk serialization and a corresponding performance penalty. Contrary to the behavior of a single merge, increasing the buffer size in this situation may actually degrade performance. Analytical models for the race condition are presented. Solutions based on run placement, buffer control, and disk scheduling policies are shown to be effective.

References

[1] A. Aggarwal and J. S. Vitter. The Input/Output Complexity of Sorting and Related Problems. *Comm. ACM*, 31(9):1116–1127, 1988.

[2] J. R. Jump. *YACSIM Reference Manual, Version 1.1.* Electrical and Computer Engineering, Rice University), April, 1992.

[3] D. Knuth. *The Art of Computer Programming. Volume 3: Sorting and Searching.* Addison-Wesley, 1973.

[4] S. C. Kwan and J. L. Baer. The I/O Performance of Multiway Mergesort and Tag Sort. *IEEE Transactions on Computers*, 34(4):383–387, 1985.

[5] M. H. Nodine and J. S. Vitter. Large-Scale Sorting in Parallel Memories. In *Proc. 1991 ACM Symposium on Parallel Algorithms and Architectures*, pages 29–39, 1991.

[6] M. H. Nodine and J. S. Vitter. Optimal Deterministic Sorting in Large-Scale Parallel Memories. In *Manuscript(To appear in ACM SPAA 92)*, 1992.

[7] V. S. Pai, A. A. Schäffer, and P. J. Varman. Markov Analysis of Multiple-Disk Prefetching Strategies for External MergeSort. In *Proc. Int. Conf. on Parallel Processing*, 1992.

[8] V. S. Pai and P. J. Varman. Prefetching with Multiple Disks for External Mergesort: Simulation and Analysis. In *Proc. 8th Intl. Conference on Data Engineering*, pages 273–282, 1992.

[9] J. Sinclair, J. Tang, P. Varman, and B.Iyer. Performance Study of Concurrent I/O. Technical Report TR 9209, Rice University, ECE, July 1992.

[10] J. S. Vitter and E. H. A. Shriver. Optimal Disk I/O with Parallel Block Transfer. In *Proc. 1990 ACM Symposium on Theory of Computing*, pages 159–169, 1990.

[11] L. Q. Zheng and P.-A Larson. Speeding Up External Mergesort. Technical Report TR CS-92-40, University of Waterloo, Computer Science, August 1992.