High Performance Cache Management for Sequential Data Access

Erhard Rahm Donald Ferguson
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
rahm@uklirb.informatik.uni-kl.edu dfferg@watson.ibm.com

1 Introduction

This study presents and evaluates new caching algorithms for sequentially accessed data that is being concurrently used by several processes or jobs. This type of access is common in many data processing environments. In this paper, we focus on *batch processing*, which is often dominated by concurrent sequential access to data. Several of our algorithms are applicable to other domains in which sequential scanning of data is common, such as query processing or long running transactions in database systems. A survey of related prior work can be found in the full version of this paper [1].

2 Problem Statement

There are M datasets (files) in the system, which are denoted $D_1, D_2, \ldots, D_M$. The system processes a set of batch jobs denoted $J_1, J_2, \ldots, J_N$. All of the jobs are ready to execute when the batch window begins and we assume that they can be processed in any order. A job J_i reads $M(i)$ datasets $D_{i,1}, D_{i,2}, \ldots, D_{i,M(i)}$. Job J_i reads every record in a dataset in order from the first to the last. A *granule* (e.g. - Track, Cylinder) is a group of records and is the unit of transfer between the disks and the memory of the computer system. Given this model, the problem is to minimize the time between starting the batch jobs and the completion of the last job. One way of decreasing the elapsed time is to *cache* granules from the datasets in memory.

0

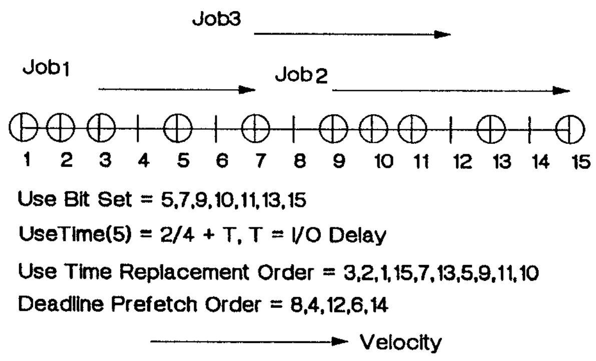
Job3
Job1
Job2
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Use Bit Set = 5,7,9,10,11,13,15
UseTime(5) = 2/4 + T, T = I/O Delay
Use Time Replacement Order = 3,2,1,15,7,13,5,9,11,10
Deadline Prefetch Order = 8,4,12,6,14
Velocity

Figure 1: An example of the replacement and prefetching algorithms

3 Algorithms

We present four new cache management algorithms: the *Binary Use Count (BUC)*, *Weighted Binary Use Count (WBUC)* and *Use Time* cache replacement algorithms, and the *Deadline Prefetch* Algorithm. All four algorithms rely on estimation of job velocities. The *velocity* of job J_i through dataset D_j is the number of granule read requests for granules in D_j that job J_i submits per unit of active time. All algorithms are parameterized by a *look ahead time* L and use the job velocities to determine which granules will be referenced and in what order during the next L seconds.

Figure 1 presents an example of the cache management algorithms. The horizontal line represents a dataset, the numbers are granule IDs and the circles represent cached granules. The vector associated with each job represents its velocity vector. The Use Time algorithm and the Deadline Prefetch algorithm use the velocity estimations and predicted misses to estimate the next *use time* for granules. The Use Time algorithm replaces granules from largest to smallest use time and the Deadline Prefetch algorithm prefetches from smallest to largest use time. The BUC algorithm randomly selects a granule not in any look ahead for replacement. The WBUC algorithm follows the same policy, but it preselects the dataset from which a cached granule is stolen based on expected future readers of the dataset.

4 Simulation Results

We compared the cache management algorithms using a detailed simulation model [1]. In figure 2 the three cache replacement algorithms are compared with LRU and no caching of data. The X-axis is cache size relative to the sum of the dataset sizes and the Y-axis is elapsed time to complete the batch workload. The Use Time algorithm ehibits the best performance, especially for relatively small cache sizes (30% better than the other algorithms at 5%). In our simulations, Most Recently Used achieved performance very similar to LRU.

Figure 3 shows the performance of the 4 replacement algorithms when the Deadline Prefetch algorithm is also active. The relative performance of the replacement algorithms is similar to the performance without prefetching. Using prefetching decreases elapsed time by approximately 10% for all algorithms for this workload.

5 Summary

We also studied the performance of an algorithm that schedules jobs based on datasets accessed to increase inter-job reuse of data. There are several possible areas for further work on the problems studied in this paper. These include enhancing the algorithms to deal with updates and non-sequential I/O activity.

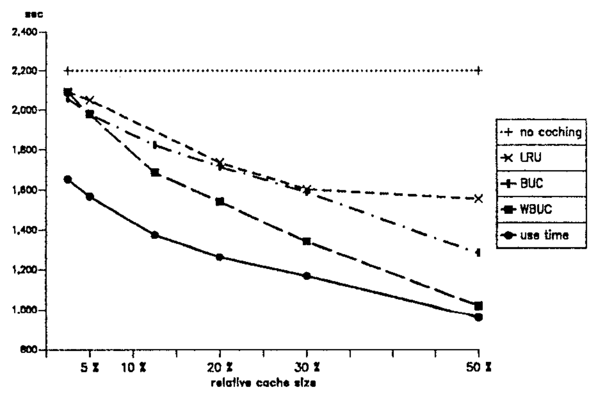

Figure 2: Elapsed time versus cache size for 4 replacement algorithms

References

[1] Erhard Rahm and Donald Ferguson. Cache management algorithms for sequential data access. Technical Report RC15486, IBM Research, 1990.

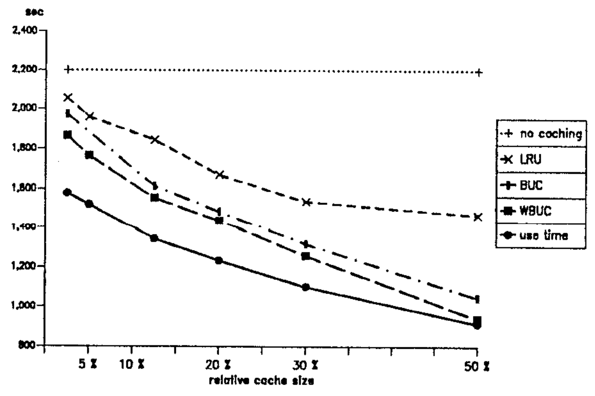

Figure 3: Elapsed time versus cache size for 4 replacement algorithms with prefetching