Scheduling Query Plans with Buffer-Requirement Estimates *

John C. Sieg, Jr.¹, David Pinkney¹, and James Lamoureaux²

¹ Computer Science Department, University of Massachusetts Lowell, Lowell, MA 01854, USA
² GTE Government Systems, Needham, MA 02192, USA

Abstract. The cost of a database query plan can often be drastically reduced by allocating more buffers to it. But allocating an optimal number of buffers may be infeasible when there is high contention for the buffer space. Previous research has explored the idea that query plans have multiple hot points, which yield local minima of buffer consumption. We describe and analyze query-scheduling strategies that use knowledge of the hot points of each query and actual buffer availability when a query is about to be scheduled.

1 Introduction

This paper explores one dimension along which buffer managers and database-system load controllers can collaborate to more efficiently process queries. While conventional database-system load controllers may use a single estimate of buffer usage of a query, techniques described in [21, 22] assign to a query multiple hot points or buffer allocations likely to yield good combinations of buffer usage and disk I/O. We describe and evaluate scheduling policies that use knowledge of the number of available system buffers and the various hotpoints of the queries awaiting database-system service.

2 Background

2.1 Buffer Management

Database systems typically store their data on secondary-storage devices such as magnetic disks. Before queries access storage objects, the objects must be brought into main memory. Storage objects are brought into memory in fixed-sized pages. The database-system buffer manager brings referenced pages into slots of memory called buffer frames or buffers. If no buffers are empty, the buffer manager chooses a victim page from a set of look-aside buffers. Look-aside buffers contain database pages that have been recently referenced. If the victim was updated while in the buffer, its new value is written back to secondary storage. The newly referenced page is copied over the buffer version of the victim, and a pointer to the buffer is passed back to the calling program.

* This research was supported by the National Science Foundation under Grant No. IRI-9211060.
2.2 Scheduling Queries

Scheduling of queries is typically done at several levels, e.g., (1) scheduling a query’s code to be loaded in memory; (2) scheduling a query to compete for buffers, processors, and I/O bandwidth; (3) scheduling requests to read or write data items; and (4) scheduling I/O requests. In this paper we consider only the second of these levels.

Motivating factors in the design of query schedulers include fairness and high throughput. Fairness is usually achieved by scheduling using a first-in-first-out discipline. The key factor in maintaining high throughput is load control, i.e., avoiding thrashing [9, 10] in the buffer space by not always allowing every submitted query to be active.

Conventional query schedulers are naive about resource usage [17]. IBM’s Database 2 is typical: demand on resources is constrained only by multiprogramming level (mpl) set at installation time [5]. The scheduler does not match query resource demands to resource availability. If too many queries contend for a finite number of buffer frames and if the replacement policy is global, processes waste their time stealing pages from each other. This phenomenon is called external thrashing. If the replacement policy is local to each query, a query can victimize pages likely to be re-referenced, resulting in internal thrashing. Either kind of thrashing can easily occur in a system constrained only by mpl-scheduling, if queries make unpredictable demands on the buffer pool. The problem is that the buffer space becomes over-committed. The solution is to refrain from scheduling new queries unless there are enough buffers to handle their predicted demands. Various models based on this idea have been explored [6, 21, 22], which are variations on Denning’s working-set model [9, 10] for scheduling processes in virtual-memory environments. Schedulers derived from these models consider the buffer demand of a process before scheduling it.

2.3 Hot Sets

Our research is derived from Sacco and Schkolnick’s notion of hot sets [21, 22]. A hot set of a query is a collection of pages that the query repeatedly accesses.

Sacco and Schkolnick’s hot-set scheduler postpones the execution of new queries when buffer availability is less than a particular hot point, i.e., the size of one of the hot-sets.

The buffer pool is partitioned. Each partition element is owned by an active query. No query triggers the victimization of another query’s hot-set pages. Partitioning by query does not preclude sharing data pages. Shared pages are charged to one query that most recently referenced the page.

A query can have more than one hot point. For example, a query implemented as multiple nested loops scanning relations \( r_1 \) (innermost), \( r_2 \), \ldots, \( r_k \) (outermost) has hot points \( 1, k, size(r_1) + k - 1 \), and \( size(r_1) + size(r_2) + k - 2 \) among others, where \( size(rel) \) is the number of pages in \( rel \). Hot point 1 is enough to get the query evaluated, hot point \( size(r_1) + k - 1 \) is enough to keep the innermost relation resident, and hot point \( size(r_1) + size(r_2) + k - 2 \) is enough to keep the innermost two relations resident.