On Optimal Multiversion Access Structures *

Bruno Becker¹, Stephan Gschwind¹, Thomas Ohler², Bernhard Seeger³, Peter Widmayer²

¹ Institut für Informatik, Universität Freiburg, Rheinstraße 10–12, D–7800 Freiburg
² Institut für Theoretische Informatik, ETH Zentrum, CH–8092 Zürich
³ Institut für Informatik, Universität München, Leopoldstraße 11b, D–8000 München

Abstract. We propose an asymptotically optimal multiversion B-tree. In our setting, insertions and deletions of data items are allowed only for the present version, whereas range queries and exact match queries are allowed for any version, present or past. The technique we present for transforming a (usual single version) B-tree into a multiversion B-tree is more general: it applies to a number of spatial and non-spatial hierarchical external access structures with certain properties directly, and it can be modified for others. For the B-tree and several other hierarchical external access structures, multiversion capabilities come at no extra cost, neither for storage space nor for runtime, asymptotically in the worst case. The analysis of the behavior of the multiversion B-tree shows that the constant loss of efficiency is low enough to make our suggestion not only a theoretical, but also a practical one.

1 Introduction

The importance of maintaining data not only in their latest version, but also to keep track of their development over time has been widely recognized. Version data in engineering databases [Kat90] and time oriented data [CA86] in geographical information systems [Lan92], e.g. to manage land register data, are two prime examples for situations in which the concepts of versions and time are visible to the user; in multiversion concurrency control [BK91, BHG87], these concepts are transparent to the user, but they are used by the system (e.g. the scheduler) for concurrency control and recovery purposes. In this paper, we are concerned with access structures that support version based operations on external storage efficiently. We follow the convention of [BHG87, DSST89] in that each update to the data creates a new version; note that this differs from the terminology in engineering databases, where an explicit operation exists for

* Partially supported by the ESPRIT Basic Research Project "AMUSING".
creating versions, and versions of design objects are equipped with semantic properties and mechanisms — such as inheritance or change propagation. Our choice of creating a new version after each update turns out not be restrictive, in the sense that the data structuring method we propose can be easily adapted to create versions only on request, without loss of efficiency.

We are interested in spatial access structures for external storage that support at least insertions, deletions, exact match queries (associative search) and locality based spatial queries such as e.g. proximity queries, in addition to application specific operations like purging of old enough versions in concurrency control. We limit our discussion to the situation in which a change can only be applied to the present version, whereas queries can be performed on any version, present or past. Some authors call this a management problem for partially persistent data; we call an access structure that supports the required operations efficiently a multiversion structure.

The problem in designing a multiversion spatial access structure lies in the fact that data are on external storage. For main memory, there is a recipe for designing a multiversion structure, given a single version structure. More precisely, any single version main memory data structure in a very general class, based on pointers from record to record, can be transformed into a multiversion structure, asymptotically at no extra amortized time and space costs, by applying a general technique [DSST89]. For the special case of balanced binary search trees, the extra time bounds hold even in the worst case per operation — clearly a perfect result.

Given quite a general recipe for transforming single version main memory data structures into multiversion structures, it is an obvious temptation to apply that recipe accordingly to external access structures. This can be done by simply viewing a block in the external structure as a record in the main memory structure. At first glance, this models block access operations well; unfortunately, it does not model storage space appropriately: the size of a block is not taken into consideration. That is, a block is viewed to store a constant number of data items, and the constant is of no concern. Even worse, the direct application of