Extendible Hashing

Donghui Zhang[1], Yannis Manolopoulos[2], Yannis Theodoridis[3], and Vassilis J. Tsotras[4]
[1]Paradigm4, Inc, Waltham, MA, USA
[2]Aristotle University, Thessaloniki, Greece
[3]University of Piraeus, Piraeus, Greece
[4]University of California-Riverside, Riverside, CA, USA

Definition

Extendible hashing is a dynamically updateable disk-based index structure which implements a hashing scheme utilizing a directory. The index is used to support exact match queries, i.e., find the record with a given key. Compared with the B+-tree index which also supports exact match queries (in logarithmic number of I/Os), extendible hashing has better expected query cost O(1) I/O. Compared with linear hashing, extendible hashing does not have any overflow page. Overflows are handled by doubling the directory which logically doubles the number of buckets. Physically, only the overflown bucket is split.

Historical Background

The extendible hashing scheme was introduced by [1]. A hash table is an in-memory data structure that associates keys with values. The primary operation it supports efficiently is a lookup: given a key, find the corresponding value. It works by transforming the key using a hash function into a hash, a number that is used as an index in an array to locate the desired location where the values should be. Multiple keys may be hashed to the same bucket, and all keys in a bucket should be searched upon a query. Hash tables are often used to implement associative arrays, sets, and caches. Like arrays, hash tables have O(1) lookup cost on average.

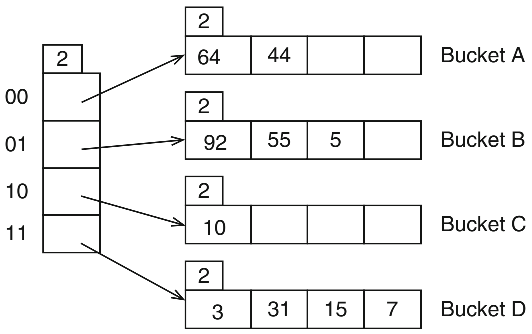
Foundations

Structure

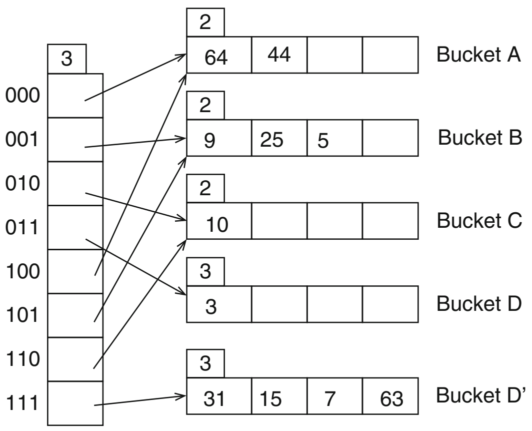
Extendible hashing uses a directory to access its buckets. This directory is usually small enough to be kept in main memory and has the form of an array with 2^d entries, each entry storing a bucket address (pointer to a bucket). The variable d is called the global depth of the directory. To decide where a key k is stored, extendible hashing uses the last d bits of some adopted hash function $h(k)$ to choose the directory entry. Multiple directory entries may point to the same bucket. Every bucket has a local depth *leqd*. The difference between local depth and global depth affects overflow handling.

An example of extendible hashing is shown in Fig. 1. Here there are four directory entries and four buckets. The global depth and all the four local depths are 2. For simplicity assume the

L. Liu, M.T. Özsu (eds.), *Encyclopedia of Database Systems*,
DOI 10.1007/978-1-4899-7993-3_741-2



Extendible Hashing, Fig. 1 Illustration of the extendible hashing



Extendible Hashing, Fig. 2 The directory doubles after inserting 63 into Fig. 1

adopted hash function is $h(k) = k$. For instance, to search for record 15, one refers to directory entry 15% 4 = 3 (or 11 in binary format), which points to bucket D.

Overflow Handling

If a bucket overflow happens, the bucket is split into two. The directory may or may not double, depending on whether the local depth of the overflown bucket was equal to the global depth before split.

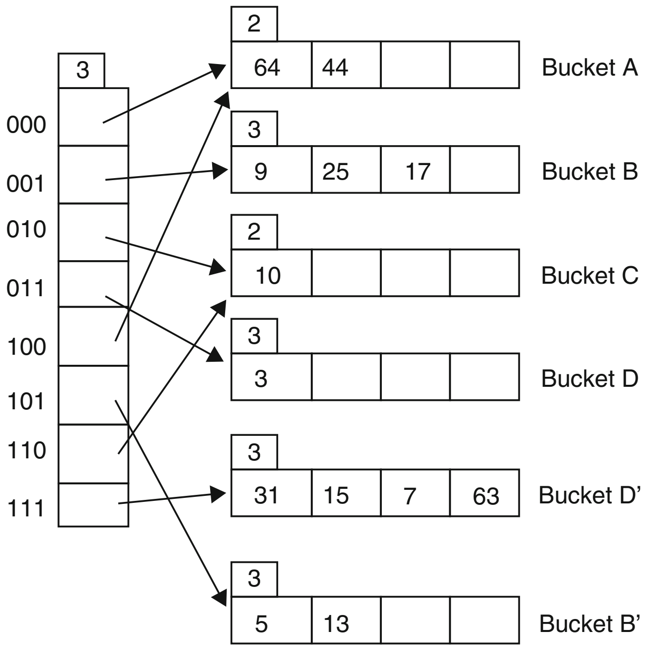
If the local depth was equal to global depth, d bits are not enough to distinguish the search values of the overflown bucket. Thus a directory *doubling* occurs, which effectively uses one more bit from the hash value. The directory size is then doubled (this does not mean that the number of buckets is doubled as buckets will share directory entries). As an example, Fig. 2 illustrates extendible hashing after inserting a new record with key 63 into Fig. 1. Bucket D overflows and the records in it are redistributed between D (where the last three bits of a record's hash value are 011) and D' (where the last three bits of a record's hash value are 111). The directory doubles. The global depth is increased by one. The local depth of buckets D and D' are increased by one, while the local depth of the other buckets remains to be two. Except 111, which points to the new bucket D', each of the new directory entries points to the existing bucket which shares the last two bits. For instance, directory entry 101 points to the bucket referenced by directory entry 001.

In general, if the local depth of a bucket is d', the number of directory entries pointing to the bucket is $2^{d-d'}$. All these directory entries share the last d' bits.

To split an overflown bucket whose local depth is smaller than the global depth, one does not need to double the size of the directory. Instead, half of the $2^{d-d'}$ directory entries will point to the new bucket, and the local depth of both the overflown bucket and its split image are increased by one. For instance, Fig. 3 illustrates the extendible hashing after inserting 17 and 13 into Fig. 2. Bucket B overflows and a split image, bucket B', is created. There are two directory entries (001 and 101) that pointed to B before the split. Half of them (101) now points to the split image B'. The local depth of both buckets B and B' are increased by one.

Discussion

Deletion may cause a bucket to become empty, in which case it can be merged with its *buddy* bucket. The buddy bucket is referenced by the directory entry which shares the last (local depth − 1) bits. For instance, the buckets referenced by direction entries 1111 and 0111 are buddy buckets. Many deletions can cause the directory to halve its size and thus decrease its global depth. This is triggered by the bucket merging which causes all local depth to be strictly smaller than the global depth.



Extendible Hashing, Fig. 3 The directory does not double after inserting 17 and 13 into Fig. 2

The fact that extendible hashing does not use any overflow page may significantly increase the directory size, as one insertion may cause the directory to double more than once. Consider the case when global depth is 3, and the bucket referenced by directory entry 001 overflows with five records 1, 17, 33, 49, 65. The directory is doubled. Directory entries 0001 and 1001 point to the overflown bucket and the split image. All the five keys will remain in the original bucket which is again overflowing. Therefore the directory has to be doubled again.

To alleviate this problem, one can allow a certain degree of overflow page links. For instance, whenever the fraction of buckets with overflow pages becomes larger than 1%, double the directory.

Key Applications

Extendible hashing can be used in applications where exact match query is the most important query such as hash join [2].

Cross-References

- ▶ Bloom Filter
- ▶ Hash-Based Indexing
- ▶ Hashing
- ▶ Linear Hashing

Recommended Reading

1. Fagin R, Nievergelt J, Pippenger N, Strong HR. Extendible hashing: a fast access method for dynamic files. ACM Trans Database Syst. 1979;4(3):315–44.
2. Schneider DA, DeWitt DJ. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In: Proceedings of the 16th International Conference on Very Large Data Bases. San Francisco; 1990. p. 469–80.