LINEAR VIRTUAL HASHING :A NEW TOOL FILES AND TABLES IMPLEMENTATION

LITWIN WITOLD

,0

MOD-I.021

C66L

Paper submitted to IFIP TC-2 Working Conference on Data Base Architecture.
77 June 26-29 Venice, ITALY.

Linear Virtual Hashing : a new tool for files and tables implementation

LITWIN Witold

I.R.I.A.

78150 Le-Chesnay France

C66L

ABSTRACT

Virtual hashing is a new addressing technique first proposed in /LIT77/. The idea in this technique is to resolve collisions not only as the well known hashing does by overflow records creation, but also by changes to the hashing function. Typically such changes split a set of keys mapped on one address into two sets, one of them being henceforward mapped on a new address. The reorganizings avoid then most of overflow records which would exist if the classical hashing was used. It follows that insertions lead to smaller deterioration of search performance, especially when more insertions occur that it was expected. We propose an algorithm which for files avoids search performance deterioration almost at all i.e finds almost any record in one disk access, no matter how few records were expected and how many insertions the file had finally to support. The algorithm also applies to tables, providing in the same conditions less than 1.7 probes per average search. No algorithms for files and tables implementation offering such performances were known until now.

The algorithm was called Linear Virtual Hashing, LVH in short . We present its principles, analyze its performances and compare them to principles and performances of algorithms for virtual hashing already known. We show that LVH needs very few core and so may be implemented on any microcomputer. We also show that LVH is very simple and so may programed by casual user. All these properties render LVH a new and important tool for a database physical structures design.

1. INTRODUCTION

Virtual hashing shortly called VH is a new addresing technique first proposed in /LIT77/. The idea in VH is to resolve collisions not only by overflow records creation, as the classical i.e. the well known hashing does (/AHO75/, /BLA77/, /CHO75/, /CUI73/, /KNU74/, /LUM73/, /MAR77/, /SPR77/), but also by changes to the hashing function. Typically such changes split a set of keys mapped on one address into two sets, one of them being henceforth mapped on a new address (that is why we called such a change split). The corresponding reorganizings avoid to create most of overflow records necessary if the hashing function was static. As /LIT78c/ shows, the file which with the classical hashing would overflow with very few insertions, may with VH support millions of insertions and still allow to find almost any record in one disk access. On the other hand, even after many deletions the load factor stays high. These properties render virtual hashing a new and very important tool for databases implementation.

Up to day four major VH algorithms were defined : VHO /LIT77,a,78b,c/, Extendible Hashing (EH) /FAG78/, Dynamic Hashing (DH) /LAR78/ and VH1 /LIT78,a,b,c/. Hashing functions dynamically created with VH1 are represented with help of a bit table. We need less than 1 bit per address provided by the created function. The table may then stay in core even after millions of insertions and that is why, up to such number of insertions, VH1 finds almost any record in one disk access. During insertions the load factor typically oscillates between some ρ and $\rho/2$ where ρ is close to 1, the average load beeing 3/4 of ρ. The oscillation is due to the fact that each time the address space extends, it doubles.

VHO, EH, and DH make the address space growing linearly i.e. any split extends the upper bound by one. It may be shown that the load factor may then be equal to 70 % approximatively /LIT77b/,/FAG78/,/LAR78/,/MAR78/. There still is an oscillation around this value but negligeable for bucket capacity less than 10 records and reaching 13 % for 50 records per bucket, /LIT77b/. However this nice behaviour of load factor is obtained at the price of an index which must be large after few insertions. Thus it must be partially or totally stored on the disk and in practice we need at least two disk accesses per record. The three algorithms differ by index structure.

All the above VH's share the same assumption : the address of a collision and the address of the corresponding split are the same. The basic idea in the VH introduced below, which we call Linear Virtual Hashing (LVH), is that it may be worthwhile to make these addresses different. We show that a few bytes of core are then sufficient after milliards of insertions, i.e. when VH1 would need hundredth of kbytes and the others VH's many mbytes. This property allows LVH search performance to be constantly very close to one disk access per record, even if we have a very small core (microcomputers) and a very large file. On the other hand, despite the absence of the index, the address space of LVH grows linearly. That is why the load factor practically does not oscillate, staying close to this of VHO.

Below, Section 2 defines the principles of LVH. Section 3 is devoted to performance analysis. Section 4 recalls principles of VH1 and of VHO and compares performances of these algoritms with performances of LVH. We end up with conclusions.