Hybrid tables for speeding-up data accesses in hybrid database management systems

Joan Guisado-Gámez[1], Antoni Wolski[2], Calisto Zuzarte[3], Josep-Lluís Larriba-Pey[1], and Victor Muntés-Mulero[1]

[1] DAMA-UPC, Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, Campus Nord-UPC, 08034 Barcelona
{joan, larri, vmuntes}@ac.upc.edu
[2] IBM Helsinki Laboratory, Finland
antoni.wolski@fi.ibm.com
[3] IBM Toronto Laboratory, Markham, Ontario, Canada L6G 1C7
calisto@ca.ibm.com

With the introduction of unexpensive memory, new techniques for reducing the access time to data stored in databases have been proposed. On the one hand, buffer pool techniques have been proposed. On the other hand, main memory database management systems (MMDBMS) have been developed. While buffer pool techniques are based on speeding up the performance by copying the most accessed pages in memory, MMDBMSs propose a new paradigm consisting in designing the internal structures to store the whole amount of data into the main memory of the system. Although MMDBMSs are fast on the retrieval of information, they limit the total amount of data to the total amount of memory available. This restriction may be too strict for many databases whose sizes are often hundreds or even thousands of gigabytes. As a reaction, dual database management systems (DDBMS) appeared as a type of database developed with two different engines. One engine is in charge of providing access to those tuples that are stored in disk, while there is another engine in charge of providing access to those tuples that are stored in memory. Note that DDBMSs force tables to be either in-memory or on-disk. Thus, large tables that do not fit in memory must be classified as on-disk preventing them from benefiting from main memory techniques.

In our work [1], we presented TwinS, a hybrid database management system that aims to exploit the main memory techniques even for those tables that do not fit completely in memory. The main goal of this proposal is to take advantage of both the speed of the in-memory engine and the capacity of the disk. TwinS introduces the new concept of *hybrid tables*. Thus, one logical table is split into **two physical tables** kept on disk and in memory, respectively. Hence, for one single relation, tuples are partially managed by an in-memory engine and partially managed by an on-disk engine. This poses a challenge when it comes to decide the existence and location of a specific row, when performing an access by using the value of a unique attribute. Accessing the wrong physical table when locating the tuple will reduce the performance achieved by the in-memory engine. In order to manage the accesses into the physical tables that make up a hybrid table, TwinS has the *access path mediator* (APM). The access

path mediator is implemented using two different approaches: *latency priority* (LP) and *predictor based* (PB). In the first case, the APM takes the latency as the only information to decide the first physical table to be accessed, i.e. the table that is in memory. Only if the information is not found, the APM accesses the table that is stored on disk. In case of applying PB, the APM incorporates a new kind of structure that is called *predictor*. The predictor keeps information at the tuple level and it foresees if a tuple is in the physical table that is stored in memory or in the physical table that is stored on disk. Predictors must fulfill three restrictions: i) the access time has to be as fast as possible, ii) they have to be an accurate structure and iii) they may return false hits but no false misses. Due to this restriction we decided to base the predictors in a bloom-filter based structure: Partition Dynamic Count Filters (PDCF). It is a set of counters that are designed for being memory and CPU efficient. This structure may return false positives, because different keys can collapse into the same counter, but it never returns false negatives. The ratio of false positives can be configured by increasing or decreasing the number of counters that make up the PDCF.

We have compared TwinS with IBM®SolidDB® which is a DDBMS. Our experiments consists in accessing randomly the tuples of a table with different assumption of memory available, moreover, we run a set of experiments that contains a certain amount of queries on data that do not exist in the database. We compare the results obtained with a hybrid table of TwinS with those obtained when the table is classified as in-memory and as on-disk in IBM®SolidDB®. Our results are useful in order to see that TwinS clearly outperform buffer pool techniques. This is shown through a set of experiments that divide the available memory between the buffer pool and the space that the in-memory part of the hybrid table uses. We can also conclude that, in general, as the available memory increases, the results obtained by TwinS become similar to those obtained by using in-memory tables of IBM®SolidDB®. This is particularly true in case that the system has to deal with a certain amount of queries on data that do not exist in the database. In this particular scenario, the predictor is a very useful structure in order to reduce unnecessary accesses into the in-memory and the on-disk part of the hybrid table.

Hybrid tables are a good solution in comparison to all-or-nothing solutions that allow to improve the overall performance of the system, making our proposal important for real-time applications.

Acknowledgements: The authors would like to thank the Center for Advanced Studies (IBM Toronto Labs) for supporting this research. We also thank the Ministry of Science and Innovation of Spain and Generalitat de Catalunya, for grant numbers TIN2009-14560-C03-03 and SGR-1187 respectively.

References

1. Guisado-Gámez, J., Wolski, A., Zuzarte, C., Larriba-Pey, J., Muntés-Mulero, V.: Hybrid in-memory and on-disk tables for speeding-up table accesses. In: DEXA. pp. 231–240. Springer (2011)