Recent Research on Database System Performance

Advanced Research Topics in Databases

Eda Baykan

28.06.2005

Abstract

Performance of databases highly depend on how data is stored and accessed. It is a challenging task to design a data layout scheme that gives good performance results under different query types and changing workloads when concurrent queries are given to the system. In this report, our aim is to present recent research proposals that aim to overcome performance bottlenecks in databases.

1 Introduction

DBMS bring data from secondary storage(I/O) to processor in order to execute queries. Data passes through the memory hierarchy which consist of I/O, main memory, L2 cache, L1 cache respectively. The number of disk I/O operations effect the performance of DBMS because disk seek time and transfer rate from disk to main memory constitutes the major part of time that ellapses during data transfer. As presented in Asilomar Report [4], cost of RAM is getting smaller and its capacity is becoming larger when compared to past. That's why reducing number of accesses to RAM is being studied as well as reducing number of disk I/O by database performance researchers. In order to reduce RAM accesses, caches are used between main memory and processor. As shown in Ailamaki et al. [2], misses in L2 cache is the main reason for bad performance in DBMS when data is kept only in main memory (when there is no access to disk).

Traditional DBMS use pool of threads to handle concurrent incoming queries to the system. Multithreading enables serving multiple clients simultaneously by using available resources in an efficient way. However this approach has some drawbacks. Firstly, it is not possible to estimate the number of threads that enables highest performance. Secondly, when context-switches occur between threads, the working set of the suspended thread is evicted from the cache. With this approach, the resumed thread can not get benefit of the suspended thread's work even they have some common work to do. This leads to waste of time and resource. Thirdly, the resumed thread loses time by redoing the work it has done before it has been suspended.

In this report we have aimed to present different approaches which are proposed to increase DBMS performance. This paper is organized as follows. We

give definitions of different query types and main approaches for improving database performance in Section 2. In Section 3, we explain and compare different Data Storage Models proposed for overcoming performance bottleneck. In Section 4, we present the work of Ailamaki et al. [3] who proposes a modular design which aims to give good performance results under multiple concurrent queries. Finally in Section 5, we conclude.

2 High Performance DBMS

In this report, we focus on papers that propose below techniques to have high performance DBMS.

- Cache-conscious data layouts
- Query adaptive data layouts
- Modular/Pipelined Design for Parallelism

Most of the papers in database performance research, focus on smart data storage techniques in secondary storage and in L2 cache. Data storage schemes can be classified as static and dynamic. Static data layouts can not give good performance under changing query workloads since they are designed a priori for predetermined query workload. Dynamic data schemes are prepared according to queries that are being executed. That's why these schemes give good performance when query workload changes during. In [9] is given a page layout technique, called Clotho, that gives good performance results under dynamic workload. Moreover queries can be classified as full-record access and partial record access. Full-record access queries require most of the attributes of a tuple while partial record-access queries only need a subset of the attributes. Some data schemes like DSM [5] are designed for partial-record access queries. DSM can not handle fast query executions under full record ones. Clotho [9], Data Morphing[6] and Fractured Mirror [8] are some of the proposed schemes for enabling DBMS work fast under each type of query. Another strategy for using memory efficiently is, to have a modular design that avoids the problems encountered during context switching. A solution that divides query execution steps into stages, is given in [3]. This approach aims at maximum use of utilities by assigning works that require same resources to the same stage.

3 Data Storage Models

In this section we give a comparison of data layouts which are proposed to increase DBMS performance. They can be classified as below:

- N-ary Storage Model (NSM)
- Decomposition Storage Model (DSM)
- Partition Attributes Across Model (PAX)
- Data Morphing (DM)
- Clotho Storage Model (CSM)
- Fractured Mirrors

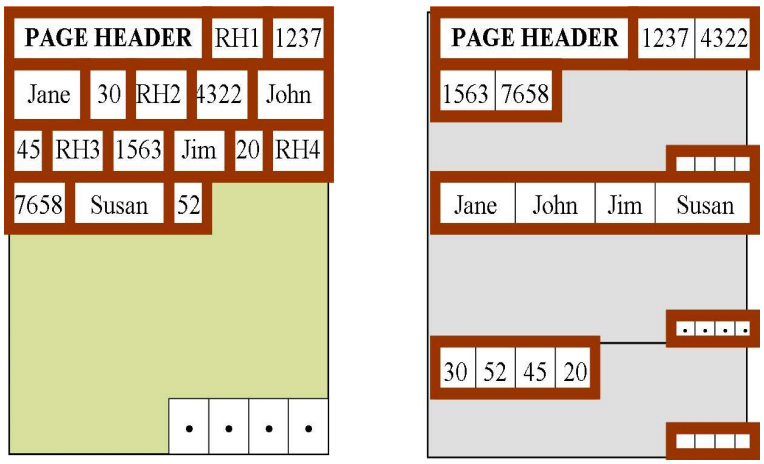
3.1 NSM

Commercial databases use N-ary Storage Model (NSM) [7]. All attributes of a record are stored consecutively in one page as $\langle record\_id, attribute\_set \rangle$. At the end of the page there exists an array holding pointers to the beginning of each record on that page. NSM storage model enables good performance for full-access queries however it does not perform well for partial access queries. When a cache miss occurs, a full record will be brought to the cache. For a partial-access query, all attributes are unnecessary. That's why unreferenced attributes cause inefficient usage of cache.

3.2 DSM

Decomposition Storage Model [5] stores each attribute of a record in a different page as $\langle record\_id, attribute \rangle$ format. In one page, same type of attributes from different records are placed consecutively. For a partial-access queries DSM is a good data layout technique because it enables spatial locality. On the other hand for full-access queries, that require multiple attributes from the same record, join operation must be applied between different pages. This can lead to an increase in query response time, in other words lead to decrease in system performance.

3.3 Partition Attributes Across Model(PAX)



Authors of [1] aim to improve performance of NSM against partial-access queries. PAX, a data layout scheme proposed for L2 cache, stores all attributes of a record in the same page like NSM. However it reorganizes the records in a page. Given a record that consists of n attributes, PAX partitions each page into n mini pages and puts each attribute of the record into different minipages such as putting n^{th} attribute in the n^{th} minipage [1]. With this approach, PAX reduces cache misses against partial-access queries because it imitates DSM behavior by putting attributes of a record into different mini pages. In the above figure on the right side is the NSM page layout where attributes of records are stored contiguously on the other hand, in PAX one page is divided into minipages.

3.4 DM

PAX, NSM and DSM are schemes that perform better on specific type of queries. They are designed before query execution. Hankins et al. [6] proposes a cache-conscious data scheme, named as Data Morphing, for L2 cache taking into consider query workload. In other words, Data Morphing technique prepares data layout schemes according to queries that will be executed. Main idea in DMS is to place the attributes, which will be accessed together by the query, consecutively in memory. These attributes form groups. Data Morphing arranges data layout schemes according to query demands. After calculating cache efficient data layout, data is rearranged into partitions,sets of groups. Working principle of Data Morphing can be explained with the below example.
Given

- Relation R($priority : int, location : int, usage : int$)
- Query
  SELECT location, usage
  FROM R WHERE priority < 12

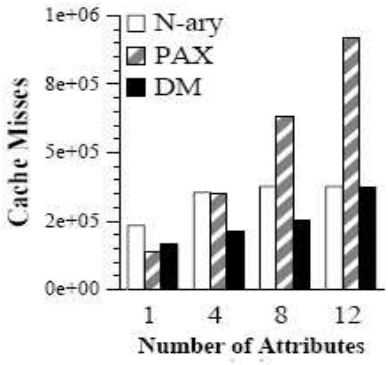
Suppose selectivity of priority attribute is %12.5. This means that the frequency at which attributes location and usage will be accessed is %12.5. Let's calculate the cache miss cost for the below group.
g = $\{priority, usage\}$
When the size of cache line is taken as 32 bytes, 4 instants of this group can be fetched in one cache miss (since the size of group is 8 bytes). This group is accessed for every record because selection is done over priority attribute. Briefly cost of a query on this group is 0.25 cache misses per record. Following this logic, in the below table is given the cache miss costs for all possible groups.

Group	Cost
$\{priority\}$	0.125
$\{location\}$	0.125
$\{usage\}$	0.125
$\{priority, location\}$	0.25
$\{priority, usage\}$	0.25
$\{location, usage\}$	0.25
$\{priority, location, usage\}$	0.375

Partition	Cost
$\{priority\}$, $\{location\}$, $\{usage\}$	0.375
$\{priority, location\}$,$\{usage\}$	0.375
$\{priority, usage\}$,$\{location\}$	0.375
$\{priority\}$,$\{location, usage\}$	0.25
$\{priority, location, usage\}$	0.375

- If NSM were used as data model partition would have one group of attributes such as: $\{priority, location, usage\}$
- If DSM were used as data model partition would have three group of attributes such as: $\{priority\}$, $\{location\}$, $\{usage\}$

As it can be seen from the above tables, partitions with NSM and DSM do not give the minimum cache miss cost. Calculating the optimum group of attributes that minimize cache misses is a challenging task. Authors of [6] propose algorithms which calculate the cost of cache misses for all possible groups and for all queries in the query workload. The partition that gives minimum cost is chosen as page layout. According to the above table, partition that gives minimum cost is $\{priority\}$,$\{location, usage\}$ for the given query workload. Data layout scheme will be prepared so that location and usage attributes will be stored on the same page while priority attributes will be stored on a different page.



As it can be seen from figure Data Morphing causes fewer cache misses when compared to NSM and DSM under all types of workloads. The drawback of this technique is that, if the query workload changes dynamically reorganizing data according to new data layout can take time and make performance decrease.

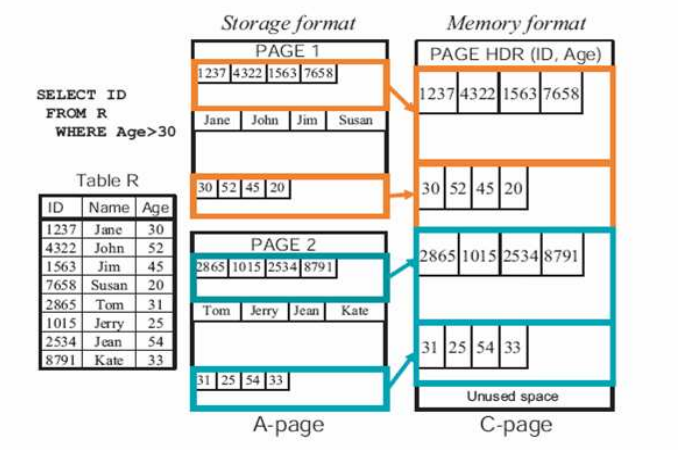
3.5 CSM

Authors of [9] make critics of above mentioned data schemes because of their assumption which say that pages in secondary storage and main memory have same content. As mentioned in [9], different data layouts must be designed for different levels of memory hierarchy by taking into consideration memory level specific properties. They propose below data layout schemes for different levels of memory hierarchy.

- A-page: Data layout scheme for disk
- C-page: Data layout scheme for RAM

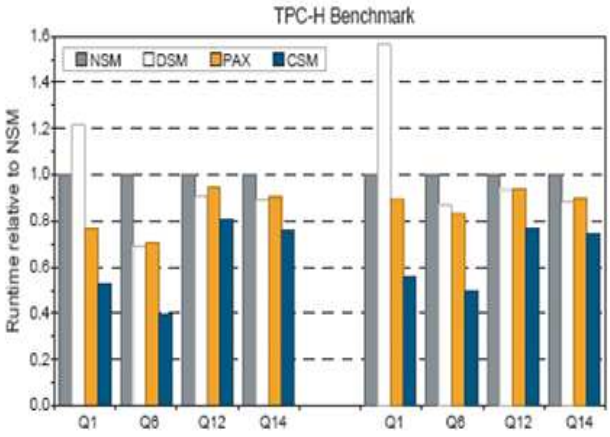
A-page stores records like PAX model[1] on the other hand C-page(CSM) contains attributes according to queries' demands. In other words attributes which are needed for query execution are transfered from disk to main memory. CSM

can easily adapt to changing workloads and performs well for both full-access and partial-access queries because C-page layout is designed during query execution. Furthermore, if queries need common attributes of records, Clotho buffer pool manager does not create a C-page from scratch. Instead it merges the new schema with the most overlapping one in the memory [9].



As it can be seen from the above figure, only id and age attributes are brought into RAM since query includes only those attributes. This dynamic selection of which attributes to bring into RAM is the key idea of Clotho.

In [9] is given a comparison of different storage model under different queries of TPC-H Benchmark. As it can be seen from the below figure, Clotho gives better performance results under each type of query when compared to NSM, DSM and PAX.



3.6 Fractured Mirrors

Authors of [8] propose to store data in NSM format as well as DSM format to have small query execution time under both partial and full access queries. Main idea in this approach is to be able to switch between NSM and DSM format when type of queries change. This is achieved by storing tables in both NSM and

DSM format in secondary storage. The easy approach would be to store tables which are in NSM format on one disk and store the tables in DSM format onto other disk. However this approach can lead to unfair utilization of disks when query workload has a skew towards one type of query. In order to remedy this problem following approach is proposed: NSM representation of table is divided into two equal size segments, lets call them as NSM_0 and NSM_1. Also DSM representation of table is divided into two equal size segments, lets call them as DSM_0 and DSM_1. On one disk NSM_0 and DSM_1 and on the other disk NSM_1 and DSM_0 is stored. With this approach disk accesses will be uniformly distributed among the two disks. Disk read requests are partitioned between mirrors based on expected seek times during query execution. The drawback of Fractured Mirrors design is duplication of space.

4 Modular Database Design

In order to have high performance traditional Database Systems use parallel threads while executing queries. Each thread is assigned to execute different queries. When CPU gives turn to a thread by preempting the running thread, the query being executed is quited. Even threads use common data, when context switching takes place all data is evicted from memory. In other words context switching is done without taking into consideration efficient use of memory in current DBMS. In [3] is given a modular design that aims to overcome the drawbacks introduced by context switching. Authors of [3] propose a modular design based on staged server programming which is introduced by Operating System researchers. In this design there are two levels of thread scheduling:

- Global scheduling across stages
- Local thread scheduling within a stage.

Each stage has its own server code, resource management and thread scheduling. The main idea in Staged Database Design is to break execution into modules so that cache misses are minimized and query response time is decreased. Keeping accesses to the same data structures is one of the reasons for breaking execution into stages.

Stages

- Connect
- Parse
- Optimize
- Execute
- Disconnect

Stages communicate with each other through queues where packets are kept. Packets carry pointers to query's data structures and private states. When CPU gives turn to a stage, threads of that stage dequeue packets from stage's queue and restores the corresponding query's state. After that stage runs its code. As indicated in [3] the implementation of this design has not been completed.

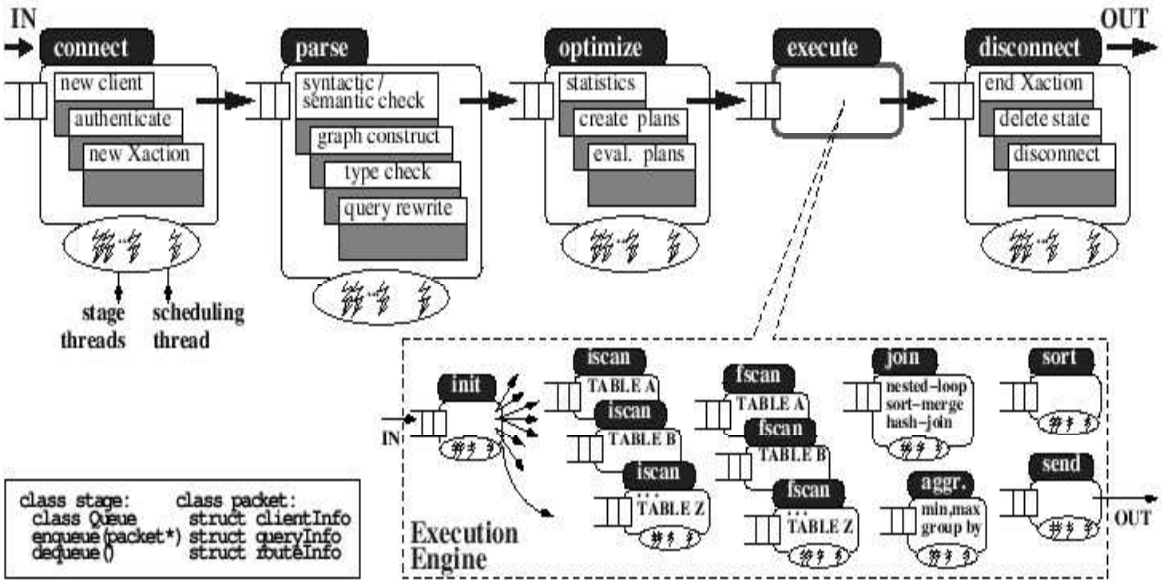


Figure 1: DBMS with Stages

Like all staged designs, staged design for DBMS has the drawback of scheduling too. There is a possibility of increase in response time because queries that need to access other stages are postponed until the rest of queries in that stage complete execution. In order to clarify this issue, [3] presents an experiment that compares prevailing scheduling and modular design in terms of response time.

As it can be seen in figure2 below, modular designs decrease the query response time by %40 when compared to processor-sharing policy(PS) which does not reuse cache contents because of context switching between threads.

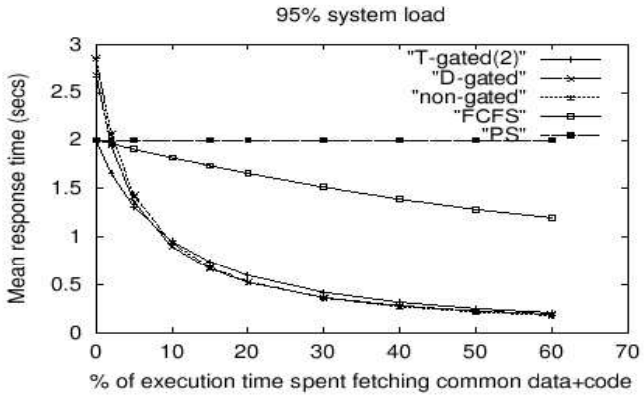


Figure 2: Staged vs. Traditional

5 Conclusion

In this paper we have aimed to give a survey of proposed techniques to overcome database performance. All proposed techniques aim efficient use of memory hierarchy to decrease query response time. NSM, DSM and PAX decide on the data layout before query execution. On the other hand Clotho and Data Morphing gives good performance results under changing workloads. In our opinion if Clotho uses for storing data in secondary storage the algorithm that Data Morphing uses, better results can be taken under each type of workloads. Staged Server Programming which suggests to break query execution into stages enables efficient use of memory by assigning query execution steps which require same resources to the same stage.

References

[1] A. Ailamaki, D. J. Dewitt, M. D. Hill, and M. Skounakis. Weaving Relations for Cache Performance. *In Proc. VLDB*, 2001

[2] A. Ailamaki, D. J. Dewitt, M. D. Hill, and D. A Wood. DBMSs on a Modern Processor: Where does time go? Weaving Relations for Cache Performance. *In Proc. VLDB*, 1999

[3] A. Ailamaki and S. Harizopolous. A Case for Staged Database Systems. *In Proc. CIDR Conference*, 2003

[4] P. Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Franklin, H. Garcia-Molina, J. Gray, J. Held, J. Hellerstein, H. V. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, and J. Ullman. The Asilomar Report on Database Research. *SIGMOD Record*, 1998

[5] G.P. Copeland and S. F. Khoshafian. A Decomposition Storage Model. *In Proc. ACM Sigmod Conference on Management of Data*, 1985

[6] R. Hankins and J. Patel. Data Morphing : An Adaptive,Cache-Conscious Storage Technique *In Proc. VLDB*, 2003

[7] R. Ramakrishnan and J. Gehrke. *Database Management Systems.* WCB/McGraw-Hill, second edition, 2000.

[8] R. Ramamurthy, D. J. Dewitt, and Q. Su. A Case for Fractured Mirrors *In Proc. VLDB*, 2002

[9] M. Shao, J. Schindler, S. Schlosser, A. Ailamaki, and G. Ganger. Clotho: Decoupling Memory Page Layout from Storage Organization *In Proc. VLDB*, 2002