International Journal of Computing Academic Research (IJCAR)
ISSN 2305-9184, Volume 7, Number 2 (April 2018), pp.12-21

http://www.meacse.org/ijcar

Extension for Workload Performance Measurement in an Educational Database System

Mehmet Akif Yanatma and M. Utku Kalay
Department of Computer Engineering, Yildiz Technical University, Esenler, Istanbul, Turkey

Abstract

In relational databases, there are many system parameters that affect the execution performance of workloads. Since new database trends and applications are more complex than ever, tuning system performance has become more critical. The most dominant impact that increase system productivity is the choice of physical structures such as indexes. Indexes, required for processing workloads efficiently may be determined manually by the database administrator. Due to increasing complexity of overall system parameters, even experienced adminitrators cannot determine the exact index set properly. This problem is called in literature as index selection problem. To observe the effect of this problem on system performance, we implemented an application extension that allows evaluation of workload execution performance in SimpleDB which is an educational skeleton database system. We believe that this extension will contribute to a better understanding of the query processing and planning techniques described in the database system implementation courses (DSI) offered at most universities. In this paper, we briefly explain the core and efficiency layers of the SimpleDB and then presented an example project related to buffer pool coordination. Also, we describe concurrently execution scheme that allows evaluation of workload execution performances with different index set and buffer pool size combinations.

Keywords: educational database systems, index selection, buffer pool coordination, workload performance

Introduction

Nowadays, with the emergence of the diversity and immense amount of digital data, widespread use of it increases rapidly. In this rapid development, the efficiency and productivity of database systems have become most important dynamic and this effect continues. With the promised data integrity guarantee, concurrently querying and/or updating of large data collections that are often not in primary memory are underlying main causes that make database systems complex. From this point of view, database systems can be thought of as a microcosm that contains all the basics of computer science.

In universities, database systems are basically taught on three different bases:

- Database modeling, querying and programming principles
- Disk-based data structures
- Database system implementation

For example, in Database Management course at the Yildiz Technical University (YTU), relational model, relational algebra, SQL and data integrity programming with XML that is an alternative current

model and related query languages are taught. The second important course in understanding databases is given under the name of File Organization. In this course, disk-based data structures that are called access methods such as hashing, B-tree, GridFile, and KD-tree are explained. Finally, the Database System Implementation (DSI) course that has previous two courses as a prerequisite is taught at the undergraduate and graduate levels.

To better understanding the Database System Implementation (DSI) based on the relational model, two elements that complement each other emerge as theoretical and practical. Log/buffer management, concurrency and data recovery, query processing, planning, and optimization are basic functional layers of the system and these layers involve several algorithms and methods. To realize these algorithms, writing a fully functional, real-time database system from scratch can be an extremely difficult task, perhaps require several years. Instead of this, examine of the source code of an open-source commercial database system comes to mind [1]. This is also diffucult, because source code of real database system whose all modules are full functional is very large, therefore becomes very diffucult for a student to examine and to extend. In that case, skeletal systems for educational purposes seem to be the best way to study this course efficiently.

The most important features expected from the educational system used in the DSI course are that the source code can be simple to understand and can be easily extended with additional new algorithms. Another important point is that the student has an implementation interface (API) that makes available to implement DSI algorithms without being overwhelmed by other programming details that are not related to the database. When we look at educational skeleton database systems used in universities abroad, such as MiniBase[2], MinSQL[3], Attica [4], ChiDB[5], SimpleDB[6], and SimpleDB[7] have attracted attention. When we examine these systems as general, we saw that they are running on the minimum bases expected from a skeleton system and also we observed that their source code is modifiable and extendible with the additional new algorithms. The learning curve provided by these systems is different from each other but almost all of them implemented some part of SQL and the rest is assigned to students. While Attica [4] and ChiDB [5] are single-user systems, others provide concurrent services to many users and provide a transactional database principles that are called ACID. In addition, while ChiDB[5] provides a particular virtual machine framework, specifically focusing on storing data in B-trees and SQL compiling, both SimpleDB[6,7], by contrast, is concerned with the larger scope of database systems, including transactions and concurrency. Minibase[2] is reported as having a heavy learning curve in[7], while MinSQL[3] and both SimpleDB systems[6,7] are reported that they have a steep learning curve. The MinSQL and both SimpleDB systems that are written in Java cover all parts of commercial database systems and their every module is presented to students as open source with simple algorithms as far as possible.

The SimpleDB educational system is used in DSI courses in YTU for many years. The rest of this paper, we examine main points of the system in section 2. In section 3, we describe a given project example. In section 4, we explained an application skeleton extension which allows measuring the performance of a workload run concurrently in the database server. Thanks to this application skeleton, we think that students will be able to understand better the principles of query planning / processing by testing the planner layer in the DB system more easily. Finally, paper is concluded with a summary and future work.

SimpleDB

SimpleDB, a skeletal database system for educational purposes contains all the layers of a commercial database system so that it provides a convenient platform for understanding the subjects in all aspects. The system is written in JAVA, originally contains about 5000 lines of source code in 12 packages (90 classes). With unit test capability that enables to run each package without initializing server, the ability to test each package internally makes a huge contribution to understanding the algorithms in the system. In addition, it is of special importance to debug the system code in order to understand the interaction of the packages in the single query processing and concurrentprocessing with multiple clients. The system is implemented on the basis of 2-tier client-server architecture.

Client-side Packages. On the client side, JDBC driver that allows communication with the server is implemented. With JDBC standard interfaces such as Driver, Connection, Statement and ResultSet, the

programmer can send queries and scan query results after connecting to the server. The server-client infrastructure is implemented with the RMI (RemoteMethod Invocation) protocol.

Server-side Packages. SimpleDB server can be handled in 2 parts. The first of these is the core of the system that provides basic functionality. The other is efficiency packages. Thus, students can better understand the core skeleton of the system.

System Core Packages. The general architecture of the core is layered and each layer is designed to take service from the underlying layers. The basic packages from bottom to top, file access, buffer organization, concurrency and recovery, record and catalog management, query processing and planning layers are implemented with the simplest algorithms. There are 4 types of files in the system, including catalog, log, access structures and data files. These files are random access files and the system can access these files at block level. File organization, such as placement and access of the file in the disk are factors that affect the overall efficiency of the system and they are performed with services received from the operating system like most of the commercial database systems. Thus, the software complexity of the database system decreases to an acceptable level.

Buffer organization contains the principles of the DB system's own buffer pool coordination acting independently from the operating system. The reading and updating of the data are done on pages that are taken from disk to pool. For example, since the pool is a limited resource, when a requested page is not already in the pool, the victim page from the pool is chosen more intelligently in accordance with access/usage characteristics. This is just one of the DBMS's own principles mentioned beforehand. The victim is selected randomly from the pool in the current system. The first assignment project we give in the course is about the buffer organization. LRU/MRU methods, selecting least/most recently used page as victim increase system performance, specifically the number of disk accesses. In this assignment, implementation and comparison of these methods are studied. This assignment is explained in more detail later in this article.

The next layer is the concurrency and recovery packages including the implemetation of ACID basics. Atomicity, Consistency, Isolation and Durability is the set of properties of a transactional database system. Many transactions called as indivisible programs access database concurrently with interleaving of read/write actions. While the concurrency is indispensable in point of system utilization and hence performance, it is a major threat in point of data integrity (i.e. consistency and isolation). Concurrent operation of transactions in SimpleDB is accomplished with lock-based protocols, as it is in most traditional systems. On the other hand, ensuring the atomicity and durability of database transactions is implemented with rollback function for incomplete transaction and commit function for providing durability within the scope of well-known protocols in the DB literature such as undo-redo, undo-only and redo-only.

The next upper layer is about records and catalog management. Organization of records within disk pages and files and scaning tables with record access ability are the main issues in the record management package. SimpleDB simply places fixed-length records within available slots in the file blocks. Thus, record management hides access to file blocks from the top layers thus it provides the appearance that the file is just a list of records. On the other hand, catalog management keeps track of names of existing tables, indexes, views and attributes they contain. Catalog management is also resposible for collecting and updating table statistical data which are very critical for query planning and processing. The statistical data of the tables such as the number of blocks, the number of records and the number of distinct values are calculated when the system starts up and are refreshed with regular intervals. This information is the minimum statistical information which is necessary for the planner in the upper layer to be able to establish the efficient plans.

Finally, the query processing is performed by processing basic relational algebra operators such as select, product, project which forms the query tree. The query tree is generated in the planner and optimized quickly as far as possible. Operators within the algebra tree are processed based on pipeline execution. In SimpleDB, the basic planner generate the simplest query tree without any statistical analysis. The essence of pipeline processing is based on efficiently iterating over table records so that records are only placed once in buffer pool, and the results are transmitted to the client without being stored elsewhere.

The basics of system implementation is covered in many well-known database textbooks [8, 9]. Here we only summarized the essential points of each component. In the next section we briefly introduce some efficiency extensions which is only a small part of a real database system.

System Efficiency Packages. The efficiency packages on the system core enables the system to run with a more realistic manner. Otherwise, even in the simplest queries, the system will probably have to scan files for hours (days). While it is always possible to increase the efficiency for each system component, there are some indispensible techniques. For example, query processing speed can be improved by hundreds or even thousands of times with indexing (access methods), highperformance external sorting, and some well-known heuristic approaches to query optimization. In addition, in buffer pool management, choosing victims from existing pages in buffer pool can be more intelligent way, such as with taking into account the reference patterns of common database operations. This idea is the main topic of our first project assignment that is given in our DSI course and is explained in section 3.

The existing efficiency packages in SimpleDB are based on 3 main extensions: 1-) B-tree and static hash index structures, 2-) external sort with 2-way merge and 3-) planner with heuristic methods for a better query plan. In Chapter 4, within the scope of the third extension, we will describe our extension that enables to exploit the advantage of index usage.

A sample Project: Improvement of the Buffer Pool

In DSI course, buffer management is studied after file access management. When the system's buffer pool coordination package are examined, it is easily seen that the data structures are very simple. While this is not a problem for small pools, for large buffer pool instances, scanning of the pool (i.e. controlling of whether a requested page is inside) will negatively affect system efficiency. In the first project of our course, we demand to keep the pages in the pool in a HashMap table instead of a simple list. Thus, it is possible to found the location of the requested file block without searching sequentially in the pool. More important in the other part of the project, implementation of LRU (least recently used) and MRU (most recently used) methods which are the victim selection rules is requested. The project only requires about 50 lines of change in the buffer package of the system. The student deliveries are tested with access pattern examples of sample data files. For example, in a test, 20 sample page requests are produced at an 8-page capacity buffer pool. With this test, the last state of the pool and the total number of disk accesses have been observed for the "random", "LRU" or "MRU" status of the victim selection strategy.

At the end of the project, a more comprehensive test is requested from students to see more effectively how the page replacement strategy affect the overall system performance. In the problem known as sequential flooding in the literature, importance of victim selection strategy arises in the continuous (repetitive) table scanning. Essentially the problem arises in the LRU. In continuous scanning of tables with more pages than pool capacity, due to LRU chooses a page as the victim at farthest back in history, this page is reloaded from disk in the following scan. On the other hand, due to MRU chooses a page as the victim at nearest back in history, it will result in less reloading in the following scans. For example, at the following figure (Fig. 1), the 4-page pool is represented by the continuous scanning of a 5-page table.

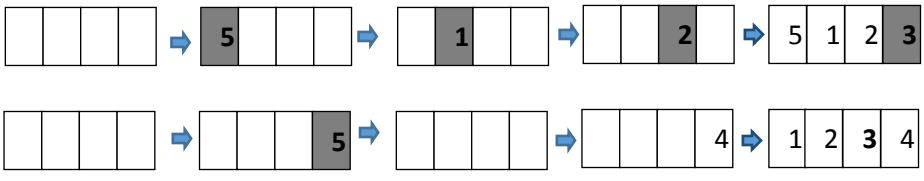

Fig. 1. Continuous scanning of 5-page table in LRU and MRU pools

In the first line of the figure, with LRU, the worst victim selection occurs resulting in 4 disk accesses and in the second line, with MRU, the best victim selection occurs resulting in only 1 disc access.

We prepared a test scenario to observe sequential flooding problem in SimpleDB buffer pool. Consider a query Q: < SELECT * FROM Department, Student > where "Department" table has 7 records in

1 page and "Student" table has 800 records in 80 pages. With the simplest implementation of the product operation – as it is in the core of the system - this SQL statement will scan all the Student pages seven times, and if the pool size is less than 81, the importance of a victim choice strategy will arise. The following graph (Fig. 2) is expected as the output of the project.

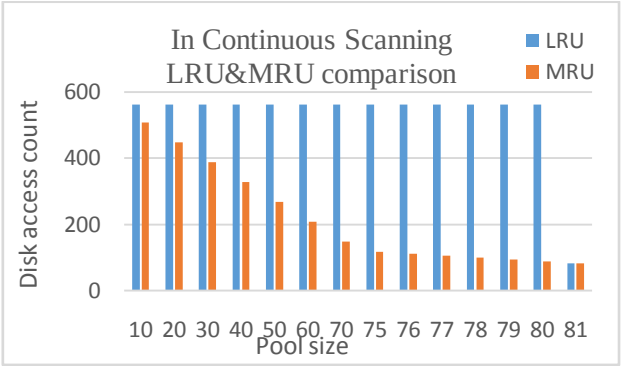

Fig. 2. Sequential flooding problem of LRU

Due to the project requires focusing only buffer coordination package and the LRU and MRU methods are easy to implement, we have experienced that this project is a successful experimental learning project as well as provides a good adaption of students to the system, consequently provides a good motivation for the following projects on the system.

An Extension that Measures Workload Performance

In this section we will describe an application skeleton extension. After loading of database tables and access methods, the system performance can be measured in concurrent processing of queries in the workload. The purpose of the extension is to provide better and faster understanding of query planning and processing in the SimpleDB system. The extension includes some changes and small extensions that are done in several packages.

After describing these add-ins briefly, we will explain how to measure workload performance with sample scenarios.

Our add-ins are split into 3 disjoint groups. These are some changes in access structures, loading of databases and selected indexes, and measurement of system performance with disk access count for a sample workload.

Extensions to Access Structures. Current access methods of SimpleDB are B-tree and static hash structures. Access methods, also known as index data structures allow the searched record to be found with minimal disk accesses and without sequential searching in the data file. For example, with a B-tree index structure which is a tree organization, billions of data records can be accessed with just a few disk accesses. The cost of accessing in B-tree depends mainly on the height of the tree and the efficient use of the buffer pool. From the point of access type, both point and range queries can be processed most effectively with the B-tree. Thus, B-tree structure is a ubiquitous structure providing the best way to retrieve records from disk for almost all times.

The existing B-tree structure in SimpleDB is not suitable for range queries because the leaf nodes are not connected to each other. For example, accessing the desired index records within the range requires in order traversal in the tree. Firstly, we linked the consequent leaf nodes to each other for which traversal becomes more efficient. Consequently, this add-in makes interval queries to be more efficient. This type of B-tree is commonly referred as B+ tree in the literature.

Another extension related to the B-tree is that the number of accessed (touched) pages are counted for performance measurement of point and range query processing. This count may be larger than the number of disk accesses because some frequently used pages (such as root and those near the root) are

caught in the buffer pool. However, the number of accessed (touched) pages in the tree is important for the students who are debugging the B-tree during query processing.

In the system, the heuristic planner (based on some well-known heuristics in the literature) was modified to place B+ tree indexes over the select and join operations in plan appropriately. Since our workload samples include both point and range queries, the B + tree is sufficient for us to measure system performance for the time being. This is also the basis for the extendable hash structures that will be able to implement in the future.

Loading Data Tables and Access Structures. In this section, some extensions that allow students to build a database of desired logical and physical characteristics are implemented. For example, the existing system enables the user to build some tables having the desired logical schema with the following standard SQL commands.

<create table student(sid int, sname varchar(12),gradyear int, majorid int);>

In addition to this, we added bulkload facility which requires some additions and changes in parser and catalog packages. Here is a sample SQL like command for bulkloading.

<bulkload 200 into student(sid:200:uniform:ordered, sname:100:Zipf:unordered,
gradyear:1:uniform:unordered:[2016_2017], majorid:4:uniform: unordered:[0_4])>

The above command is able to generate 200 student records with the attributes specified in the statement. For each attribute in this statement *attribute name(A): number of distinct values of the attribute (V): value distribution type (T): ordered/unordered(O): [value range] (R)* can be specified on the command line. In fact, those parameters, specially V,T and O are critical for planner to produce better plans. For instance, small V values, meaning small selectivity will decrese the efficiency of index methods since the result set size may become significantly larger. In such cases, sequential search will be better in terms of disk access. Similiarly, T value, distribution type (uniform or Zipf) is also an important for good predictions of number of records generated throughout the execution.

Similarly, some extensions were made to build B+ tree indexes on tables with SQL standard commands and to load them automatically. For example, once the B + tree indexes are built with the following commands they are saved to catalog immediately for which the planner can see and select the appropriate index. As an instance, with the following commands the corresponding data table is scanned and the index is loaded and the catalog is updated. Afterwards, with the drop index command, the index is removed from the file system and the catalog.

create index sidbtreeidx on student(sid);
drop index sidbtreeidx on student(sid);

Finally, in this section, we made some extensions that can report the logical and physical properties of the database after the data and index has been loaded. For example, the following report is a sample:

RECOVER EXISTING DB: DB_3_100
---
refreshing stats...
SYSTEM BLOCK_SIZE: 100
BUFFER_POOL PARAM: data_pool_size: 80, aux_pool_size= 40, buffer_replacement_policy: lru
RECOVERY PARAM: ckpt_mode: null, ckpt_threshold: 0
PLANNER PARAM: PlannerType: HeuristicPlanner
EXTERNAL SORT PARAM: split: 0 pages, merge: 2 pages, RepSel: 0 pages.
Using already existing STUDENT table:
BS: 100
RL: 32

RPB: 3
R: 200
B: 67
majorid stats: 4 uniform unordered 0 3
gradyear stats: 1 uniform unordered 2016 2016
sname stats: 100 Zipf unordered abseu zvc
sid stats: 200 uniform ordered 0 199

Performance Measurement of a Workload. In this section, we will describe the measurement of the workload execution performance. The performance testsare run under conditions such as different available indexes and buffer conditions. Best index set selection problemfor an existing workload has a historical background in database system design research. This problem is NP soinstead of finding the best set of indexes, many techniques have been put forward with the aim of becoming closeto best set. The automatic/adaptive selection of the index set by the system is beyond the scope of this article. Inthis context, we designed a student project to enable the students to see the contribution of the index selection tothe execution performance of small workloads.

The benchmark for workload performance is usually the number of disk accesses. There are many parameters thataffect the number of required disk accesses during the workload execution. Several system parameters at eachlayer of the system, ranging from the buffer pool size to the used recovery protocol settings, can affect this valuein a commercial database system. Since SimpleDB concerns with the large scope of database systems we mayexpect to experience many interesting observations. Mostly many system parameters in SimpleDB becomesimportant in terms of system tuning at the level of fine or coarse granuality. Fine level system tuning is a difficulttask. However, it is possible to observe dramatic effects of many parameters in the system with somewhat easyextensions to the system. This has been our actual motivation for this extension. Now, we will brifly explain somebasic add-ins to some packages of the system.

We observed unexpected number of disk accesses even for some basic queries. For example, the <SELECT *FROM Student;> command resulted in 115 disk accesses with 5-page LRU-buffer pool and 80-page "Student"table scan. The reason is that while planner is doing parsing and planning, it frequently accesses the catalog pagesand since the pool size is small, the many wrong victim selections caused this unexpected increase. LRU and/orMRU does not help because the problem here is that the common usage caharacteristics of catalog and data pagesmay be different. So, like the real databases, for now we divided the buffer pool into two parts:

- Data pool: The part where data blocks are held
- Auxilary pool: The part where the index, catalog and temporary file pages are held

Additional divisions of pool are possible in the future. We again set the victim selection strategy of data pool as LRU. We have achieved more consistent disk access counts since different usage characteristics are located in different pools. However, since the catalog pages are constantly accessed in many places of the planner (parser, index selection, the opening of files, etc.) and the catalog is continually scanned in each access, we have seen that our disk access count is still high if the auxilary pool is not enough. For now, we set the size of the auxilary pool to be large enough to contain all the catalog pages. For real databases, it would be better to search the catalog data (such as fldcat, tblcat, idxcat) with more robust techniques such as B-tree or hashing.

For our sample workloads execution, we have designed a number of concurrent clients to connect to the server with JDBC interfaces. Each connection has a transaction containing 1 query. When the first connection at workload is established and the server wakes up, we refresh “only” the data pool by cleaning all the buffers in the pool. Since there is no active transaction beforehand, this refreshment over “data” pool does not harm the integrity. Thus, we can more accurately measure the performance of different workloads arriving at the server.

Workload Scenario on a Sample Database. In this section, we show the exection performance of a workload without any index (with the basic planner of the system) and with different indexes. For this purpose, we designed a 5-query workload over student registration database with only 5-table. The database schema with some of its physical features and workload are as following in Table 1.

Table 1. Database Schema with Some of Its Physical Features

	B (block count)	R (record count)
student(sid int, sname varchar(60),gradyear int, majorid int)	45	450
dept(did int, dname varchar(28))	1	4
course(cid int, title varchar(24),deptid int)	2	32
section(sectid int, courseid int, prof varchar(60), yearoffered int)	25	250
enroll(eid int, sectionid int, studentid int, grade varchar(6))	500	15000

Q1: select sid, sname from student where sid=3;
Q2: select sid, dname from student, dept where majorid=did and sid=3;
Q3: select sid, sname from student where sid between 3 and 10;
Q4: select prof from section where sectid = 10;
Q5: select sname from section, enroll, student where sectionid=sectid and studentid=sid and prof= 'jgmvvply';

We observed the number of disk accesses with different data buffer pool sizes with system page size of 800B. For the auxilary pool containing catalog pages, we reserved the area about the size of catalog tables (10 pages). In this manner, it is possible to see the effect of indexes on query execution performance.

In our workload, the first 4 queries are light queries, each of which results in 4 disk accesses (when there are convenient indexes). We observed the number of disk accesses as 9 when we run these 4 queries concurrently with a 10-page data pool. Because the queries concurrently access to three different tables: student, dept, and section.

The 5th query in the workload is a slightly more complex and results in 600 records. We have seen that this query costs 1740 disk accesses with a 10-page data pool when there are convenient indexes (sid and sectionid indexes).

The graph below (Fig. 3) shows the number of disk accesses that occur when we run all the queries in the workload with different index combinations. The x axis shows the data pool size in linear scale while the number of disk access is shown in logarithmic axis. Initially, we can guess the most convenient indexes in the workload could be sid and sectionid. Because, as seen in the workload the most searched attribute is the student(sid) and also an index on enroll(sectioned) seems like valueable, because enroll is a relatively large table. Generated graphical results confirm our estimates. However, when we scrutinize the graph it has been discovered that only the sectionid index is sufficient for this workload. This is because the student table in which the sid index is used is much smaller than the enroll table in which the sectionid index is used.

When there is no index, it is seen that the minimum buffer pool required by the proper execution of the workload is higher because join operations in 2nd and 5th queries are exposed to multibuffer product. Multibuffer product requires more pages from tables to be in the pool at the same time. For example, the minimum pool capacity needed when there is no index is 50 pages. As more indexes became available, this value came down to 10.

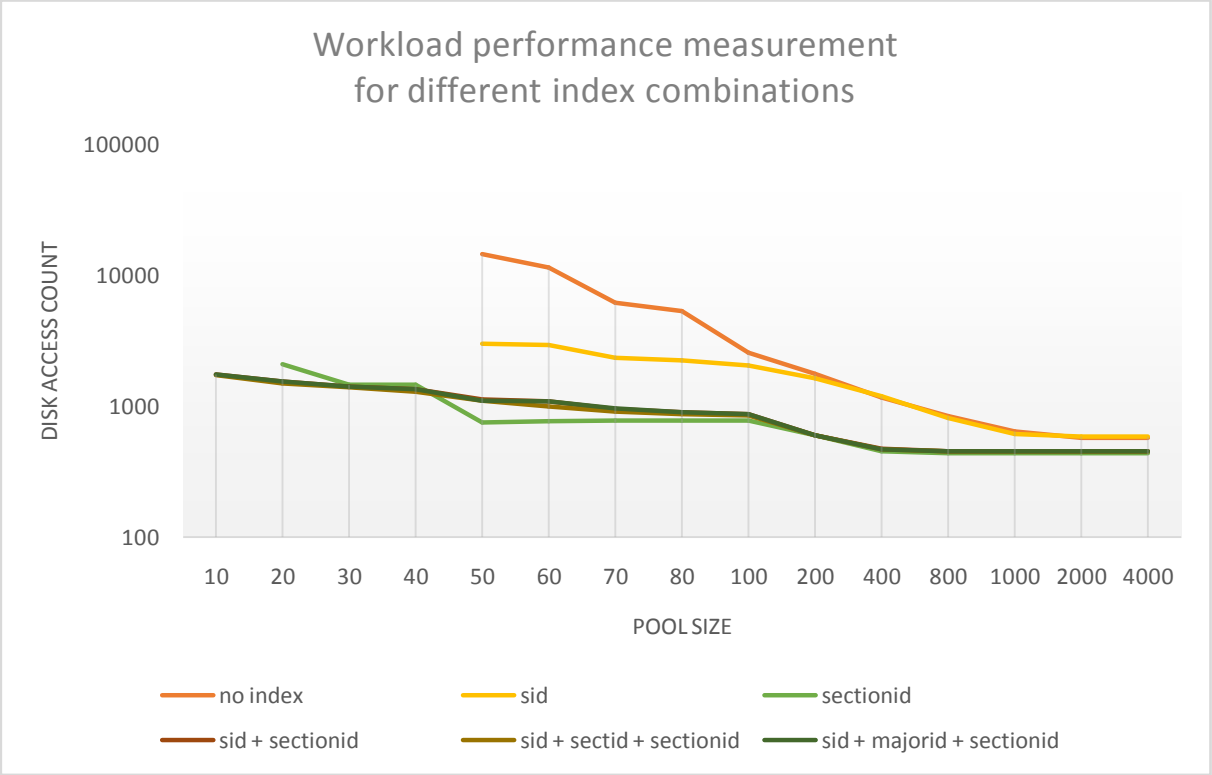

Fig. 3. Workload Execution Performance with Different Pool Sizes and Index Combinations

Another important deduction that can be made from the graph is that the increasing data pool size always relieves the performance. This is evidently a fact because less victim selection will decrease the disk accesses. However, unnecessary large pools will be useless when almost all tables' pages locates in the pool. As can be seen from the graph, for this workload, it can be said that a B+ tree index on only enroll(sectionid) attribute and about 50-page buffer pool are suitable for both efficient source utilization and fast access. The user (or student) can easily measure the workload performance for different pool size and index set conditions. Moreover, our evaluation framework seems like an instructive one that enables larger workloads to be tested.

Conclusions and Future Work

Educational skeleton systems provide a useful working platform for understanding and testing complex system softwares. As a result of our review on these systems used in universities, we found out that educational database system SimpleDB was a transactional system and it contains all the layers of the actual database systems. Based on project assignments that are given at YTU Computer Engineering database system implementation course, we can say that this extendible system brings out a steep learning curve.

In this work, we briefly explained the core and efficiency layers of the system, and then we presented a project example related to buffer pool coordination. As instructors, we feel the project has been a great success. Finally, our concurrent execution scheme that allows measurement of workload performances has been discussed. This scheme allows students to evaluate the execution performance of a workload for different sets of indexes and different pool sizes. When we present the framework extension in our lectures, the project has been the source of great interest and spurred interesting discussions about the inner workings of the system. We believe that our extension will be a basement for many upcoming projects.

Our first goal for the near future is to determine the appropriate index sets for the workload adaptively. This problem, known as physical design, has an important place in database research. In physical design, indexes on multiple attributes, not just single attributes need to be evaluated. However, this increases

the search space very quickly. Considering the fact that update commands are also included in the workload and that the burden of indexing is taken into consideration, the complexity of the problem significantly increases.

Finally, a more realistic test environment can be built for each layer of SimpleDB. For example, the implementation of different lock protocols (update locks or multi-granular locking) that allow transactions to be processed more concurrently without damaging data integrity is among our the near future projects. We also believe that with extensions that enables more SQL commands to be executed, TPC benchmarks [10] can be usedto measure the performance more realistically.

References

[1] Ailamaki, A., and Hellerstein, J. Exposing Undergraduate Students to Database System Internals. ACM SIGMOD Record, 32, 3 (September 2003), 18-20.
[2] R. Ramakrishnan. The Minibase Home Page (http://research.cs.wisc.edu/coral/minibase/minibase.html),1996.
[3] Swart, G. MinSQL: A Simple Componentized Database for the Classroom. In Proceedings of the 2nd international conference on Principles and Practice of Programming in Java (Kilkenny City, Ireland, June 16-18, 2003). ACM International Conference Proceeding Series Vol. 42, 2003, 129-132.
[4] Stratis Viglas. AtticaDB HomePage (http://www.inf.ed.ac.uk/teaching/courses/adbs/attica/)
[5] Sotomayor, Borja & Shaw, Adam. (2016). chidb: Building a Simple Relational Database System from Scratch. SIGCSE '16 Proceedings of the 47th ACM Technical Symposium on Computing Science Education. Pages: 407- 412. DOI: 10.1145/2839509.2844638.
[6] The course Page taught by Prof. Sam Madden at EECS, MIT. (https://github.com/MIT-DB-Class)
[7] E. Sciore. SimpleDB: A Simple Java-based Multiuser System for Teaching Database Internals. SIGCSE Bull., 39(1):561-565, Mar. 2007.
[8] E. Sciore. Database Design and Implementation. Wiley, 2008.
[9] Ramakrishnan, R. and Gehrke, J. Database Management Systems (Third Edition). McGraw-Hill, Boston, 2003.
[10] The Transaction Processing Council. TPC-C Benchmark. (http://www.tpc.org)