CS386D
4/11/20
V1.0

Homework 9a

Assigned: 3/11/20
Due: 3/17/20 on canvas, 11:59PM

Part 1: "Bandwidth" to a Server Over the Network

In homework 5a you were asked to load ~1.25Gbytes of data into a database and time how long it took. In homework 8, part 1, you were asked to do the arithmetic to determine, as a measure of bandwidth, how fast you were able to move data from your client program into a relational database.

You should now have Postgres running on your local machine, and on a server in the AWS cloud.

Repeat just the load part of random inserts of homework 5a using the Postgres Server running on your local machine, timing how long it takes. Repeat again for the Postgres Server running in the AWS cloud, timing how long it takes.

Turn in, the amount of time in seconds and as a bandwidth for each run. i.e, 4 numbers.

Part 2: The Deep Dive into Query System Performance Analysis, Optimizing Query Execution *and* Further Depth in Physical Database Optimization (the use of indexes)

Preamble:

Part 2 of the homework is itself being is broken into two parts, one for this week and one for next week. As an accommodation for the Covid-19 situation, you will find in the homework directory, a folder, hw9 supplementary material. In that folder you will find the preamble to the related assignment as given to the undergraduate class. The related assignment at the undergraduate level has much more hand holding. So in that supplement you will find a little more explanation, but mostly links to some very detailed descriptions. Do not hesitate to engage with that extra information.

Introduction:

The workload, in particular the database instance, for homework 5a was a brief introduction to this lab-like homework. In addition to the basic dev-ops like work, an objective of homework 5a was to give you prompt visibility that the query and access path material in this class isn't just a bunch of algorithms and the occasional analysis problem (i.e. 8.4.1), but those algorithms really are under the covers of a database, and understanding them is critical both application practice and the development of future database management systems.

A first goal of this lab is to observe that query optimizers do consider the statistical properties of the data and respond with different query plans. An operational intent is that between this lab and the previous lab you will finish the semester with experience with tools that are used to assess a database

workload and optimize the execution of individual queries and the physical organization (partitioning and indexing data) of a database.

Further, the database instance in this lab is inspired by classic results in database query performance analysis. Although not named as such in the original paper, this work is now generally known as THE Wisconsin benchmark. You will promptly observe that the database schema and database instance were designed to enable evaluation of queries, and even individual operators, parameterized on selectivities.

Required Reading:

1. The original paper, "Benchmarking Database Systems , A Systematic Approach" http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.7764

2. A relatively recent retrospective paper: "The Wisconsin Benchmark: Past, Present, and Future" http://firebird.sourceforge.net/download/test/wisconsin_benchmark_chapter4.pdf

The earlier homework targeted the relationship between select predicates, secondary indexes and disk locality. This homework targets the join operator and the optimizer's cost function wrt the size of intermediate join results.

In the previous homework, I defined a database workload; a database and a set of queries. I then had you build several different physical databases and measure the performance of a set of queries for each database. This is the standard methodology.

In this next homework, to simplify it and greatly reduce the time it takes to do the lab I am going to have you do something different. I'm going to have you build a database with 6 identical tables. However, I will specify different indexing on the different tables. Then, rather than have you create a database, run and measure a query, build a new database with different indexes and run the same query, I will specify different versions of the same query. The differences are limited to which tables are to be used. Thus, you will see different plans for, effectively the same query, but per different indexing. The result will be equivalent to running the same query on different databases, but you will only have to load the database once.

Deliverable:

The deliverable for this homework includes the physical plans determined by the query optimizer for each of the queries. Per below, this is best done using the pgadmin graphical interface. (It produces pretty pictures consistent with diagrams you have been seeing all semesters.)

At the command line, there is a SQL standard command, *EXPLAIN*. The explain command directs the query processor to take a query argument and process the query all the way to the point of creating a physical plan, *but does not then execute the plan*. Explain, at the command line, will return text-based details of the physical plan; the expression tree, the physical operators used and execution cost related details. Though the command is part of the SQL standard, the details of the return result are vendor specific.

At this juncture most DBMS vendors provide a graphical system for explaining plans. For those that do not, there is almost always a choice of third-party vendors with software that fills in that gap. The Postgres pgadmin tool provides a high quality solution:
https://www.pgadmin.org/docs/pgadmin4/dev/_images/query_output_explain.png for an example

Important details:

- Recall some column statistics used by the optimizer are collected when the DBA directs the database to analyze itself. You must be careful, after you have loaded the database, to make sure the analysis has been done.
- Explain, by itself, does not execute the query. And, there is no requirement in this homework that you execute the queries. Certain queries, this week or next, takes 10's of minutes to several hours, (according to past students who missed this instruction. Also, in the supplemental material are pointers to a tool that will, if you run a query, analyze discrepancies between cost estimates and actual costs. Making the undergraduates work queries though that tool is part of the "hand holding". It is very optional for you as the workload specifies a uniform distribution so such discrepancies are minimal.)

Note, these details are highlighted as pgadmin, and Postgres itself, has a confusing mix of pull down command and behaviors. E.g. when you build an index on a column Postgres automatically updates statistics for that column.

BTW:
Finally, advances in machine learning are, recently, beginning to fulfill a 40 year old vision for products that can monitor database workloads, accumulate for themselves the kind of data revealed by explain commands and suggest database optimizations. The nature of these products, necessarily, is that they make suggestions. While there are certainly many organizations that blindly hit a button and implement the suggestions, and do so with success, it will be quite a while before an organization with a significant database implementation will do so without someone with your (newly acquired) knowledge reviewing the suggested changes. (These intelligent systems are not placing product recommendations or ads on a web page hoping to improve the probability of selling something. They are assessing *trade-offs* in performance and do not necessarily know all the expectations of the database, the dbms and its "larger" operating environment. E.g. A query runs only rarely, so the AI could conclude its optimization is not critical, but the DBA knows, (because the CEO asked him directly to make it run fast), that the CEO randomly asks for a particular report that uses that query to spot check that his people are on top of certain things.

The Test Database:

Per the last assignment, budget time for the initial database load. Note,(hint), if you are clever, you will only have to experience the overhead of your database generator (running on a client) loading just one of the 6 copies of the table.

Test data: The test data will comprise the same data table replicated 6 times. Let that table be defined as *TestData*(pk, ht, tt, ot, hund, ten, filler) Let there be at least 5,000,000 rows. Recall you don't have to execute the queries. The columns are defined:
a unique primary key, pk.
Column ht should contain uniformly distributed values from 0 to 99,999 (i.e. **h**undred **t**housand),
Column tt, should contain uniformly distributed values from 0 to 9,999 (i.e. **t**en **t**housand),
Column ot, should contain uniformly distributed values from 0 to 999 (i.e. **o**ne **t**housand),
Column hund, should contain uniformly distributed values from 0 to 99 (i.e. etc.),
Column ten, should contain uniformly distributed values from 0 to 9
Column filler – same as before to make a row at least, but not much larger than 256 bytes.

The values should be generated at random, with replacement. Each column's contents should be generated independent of the other columns (i.e. → the easiest possible way of going about this).

Per the introduction, compare this database instance to the database specified in "Benchmarking Database Systems , A Systematic Approach", and a retrospective paper are assigned reading: "The Wisconsin Benchmark: Past, Present, and Future"
http://firebird.sourceforge.net/download/test/wisconsin_benchmark_chapter4.pdf

Physical Schema: Create three copies of the test data table. Call the tables A, B and C. The contents should be identical. Only the names of the tables are different.

Create three more copies of the test data table. Call them A', B', C', (or Aprime, Bprime, Cprime). Build secondary indices on all eligible columns of A',B', and C'

BTW – How big a database (total rows and bytes) are you now being asked to manage?

Queries: Determine physical query plans for SFW queries with the following where clauses

You should consider the structure of the resulting expression tree, the physical operators chosen, the estimated runtime and size of the final result. Per instructions in the supplement, turn on all possible measurement

For all queries below you should write the SQL SFW statement that start with SELECT *. In addition for queries, 1-5, 23 and 24, (numbering from the full assignment has been retained. Numbered queries not appearing in this homework will be assigned next week.)
you should also write the SQL query and generate the explained physical plan such that the SELECT clause contains just the arguments in the query. E.g. for query 1 this means, SELECT A.pk, B.pk . Recognize that this should result in relational project operators being pushed to the beginning of the plan and shrinking the width of the rows.

Section 1: Choice of physical operators
Determine if and when data range size, T(R), V(R, column-name) and index impact choice of join order and algorithm. The spacing of the questions is suggestive of particular comparisons you should examine, within and across groupings.

Turn in, a screen shot of each physical plan. Preceed those screen shots with answers for A through I.

1. A.pk = B.pk -- special case of merge-join reigning supreme. **A)** Does your system choose merge-join?
2. A.ht = B.ht -- a highly selective join. **B)** what join algorithm was chosen?
3. A'.ht = B'.ht -- **C)** What is the impact of a secondary index?

4. A.ten = B.ten -- **D)** same as B) but, a not very selective join; low selectivity.
5. A'.ten = B'.ten -- **E)** same as B) note, relatively speaking, a more selective join will have a lower numeric value for
-- join selectivity, than the value of join selectivity for a less selective join. Confused?
-- then, please, go turn up the air conditioner.

6. A.ht = B.ten -- **F).** does the optimizer choose its own order for the join, independent of
7. B.ten = A.ht -- what order you wrote the query. Be sure to reverse the mention of the tables
-- in both the FROM and WHERE clauses

8. A.ht = B'.ten -- **G)** how about now?
9. B'.ten = A.ht

Section 3: -- Let's try some big queries, just "for fun".
23.

A'.ht = C'.ot AND
A.ht = A'.ten AND
B.pk = C'.hund AND
A'.ht = C'.ot AND
B'.ten = C.ot AND
B'.hund = A.ht

H) In addition to developing a plan, draw the query graph.

I) How big is the estimated result, in rows?

24. **--** Something, not as big but with richer predicates

A'.ht = C'.ot AND A'.ten = 5 AND
A.ht < A'.ten AND
B.pk = C'.hund AND B.ot < 500 AND
A'.ht = C'.ot

Notice, the A.ht < A'.ten join predicate is equivalent to A.ht < 5 AND A.ht < A'.ten.

[