Summary of "QUICKR: Lazily Approximating Complex AdHoc Queries in BigData Clusters"

Chung-Wen Liao (cwliao), Bor-Kae Hwang (borkaehw), Hongyu Pang (vicpang)

Problem and Motivation

Since queries process large volumes of data on expensive clusters, decrease in resource usage would reduce the computing cost, and allows capacity limited production clusters to run more queries. Approximating big-data queries can improve query response time by trading off degradation in answer quality. Two important use cases are: (a) speeding up queries that analyze logs to generate aggregated dashboard reports would increase the refresh rate of dashboards with no extra cost and (b) machine learning queries that build models by iterating over datasets can tolerate approximations in their early iterations.
However, state-of-art techniques are not able to approximate complex queries, such as joins over more than one large table, queries that touch less frequently used datasets, or query sets that use a diverse set of columns, which dominate in big-data clusters.
This paper presents a system called QUICKR. To solve the problem described above, it has four goals:

1. Given a query, decide whether or not it can be sampled and output an appropriate query plan with samplers.
2. Support complex queries.
3. Do not assume that input samples exist or that future queries are known.
4. Ensure that answers will be accurate.

Hypothesis

- Big-data queries usually have aggregation operators, large support, and output << input, so they are approximable.
- Distribution of queries over input datasets is heavy-tailed, i.e. individual query uses many columns and a diverse set of columns, so the additional storage space for samples can be prohibitively large. Instead of performing multiple passes over heavy-tailed data, QUICKR samples data in the first pass and have no apriori samples overhead.
- For the universe sampler in QUICKR projects the join keys of join inputs into some high dimensional universe, and sample both inputs from the same random portion of this universe. They argue that joining a p probability universe sample of samples is equivalent to a p probability universe sample of the join output.
- QUICKR deals much more extensively with query optimization over samplers due to three reasons.
  1. The space of possible plans is much larger.
  2. The choice of sampler types and locations in the plan involves complex trade-offs between performance and accuracy.
  3. Adding samplers can lead to dramatically different query plans.

Solution Overview

The paper then discuss its solution mainly from two aspects.

1) What are the samplers and implementations
2) How to pass sampler states through various operators effectively.

We give a brief overview of the three types of samplers and their possible implementations

- Uniform sampler:
  - Function: let a row pass through with probability p uniformly at random.
  - Weakness:easy to miss answers to queries with group by
  - Strength: efficient memory use, better performance with reservoir
- Distinct Sampler
  - Function: deal with skew data
  - Implementation: The design is very complicated but mainly focus on partitionability, small memory footprint and less bias. And one novelty is to support stratification over functions of columns
- Universe Sampler
  - Uniquely allows QUICKR to sample the inputs of joins.
  - Advantage over others: uniform sampling both the join inputs is not useful, distinct sampling both the inputs has limited gains. Correlation-aware samplers are inefficient.
  - Implementation: with hashing, universe sampler takes an input column set and a fraction, it chooses a fraction of value space and passes all rows whose values belong to the subspace.

Next we discuss the algorithm for inserting these samplers. Every operator in the plan can be followed by the sampler, so what is the most efficient plan to place those samplers? ASALQA gives the answer.

Design choice: ASALQA incorporates samplers as native operators into a cascade-style cost-based optimizer. It modifies both during logical plan generation and physical plan generation

Problem Modeling: requirement on a sampler are encoded in a state. As sample move by transformation rules, its state will change. ASALQA picks the state that meets the requirements. Now the problem becomes how to pass samplers through transformations. And the paper define various rules for doing this.The paper shows an example of pushing the sampler past SELECT , pushing past join operation is further presented in detail.

Cost Modeling: Cost should estimate the cardinality per relational expression and the number of distinct values in each column subset. The statistics are computed by the first query that reads the table, which is yet another algorithm described in another paper.To best implement the sampler to meet the requirement encoded in the logical states, some other minor details are presented in the paper like disallowing sampling with probability above 0.1, using fixed error goal, etc. A sequence of checks is presented that does equivalent things the above restrictions does

Limitations and Possible Improvements

1. QUICKR triggers parallel plan improvements that build upon the reduced cardinality due to samplers. That is, when the data in flight becomes small, QUICKR decreases degree-of-parallelism (DOP) so that pair joins are replaced with cheaper hash joins etc. The benefit is faster completion time. The cost is that more data has to be shuffled and written to adjust the DOP.
2. Applying LIMIT 100 on the answer after sorting over the aggregation column will result in up to 20% of missing groups for every query. Errors in aggregates change the rank of groups and hence the approximate answer picks a different top 100.
3. For the aggregation error, 80% of the queries have error within ±10% and over 90% are within ±20%. Careful examination of the outliers reveals (a) Support is skewed across groups. Since QUICKR assumes an even distribution except for heavy hitters, some groups receive fewer rows than desired and have high error. (b) SUMs that are computed over values with high skew have high error. The fix is to stratify on such columns. However, a complication is that value skew changes when passing through predicates.

Summary of Class Discussion

1. Why queries are typically approximable?
   - ❖ One way to put this is that, some operators are approximable and most queries have these operators. But when UDF exists, it may be difficult to approximate. It's later shown in the evaluation, there are 25% queries that are not approximable.
2. Why does QUICKR make sense?
   - ❖ BlinkDB focuses only on frequently running queries and drops infrequently running queries, while QUICKR provides general solution. It makes sense to consider infrequently running queries because sometimes you only need some intuition, you don't need the exact accuracy.
3. If storage is extremely slow, how to store cleverly to run query, some approximating is fine, but do not trade extra storage for time?
   - ➢ If we know what queries to run, store them accordingly. The problem is we don't know the queries. Coming up with summary of data is the key problem.
   - ➢ Train model on the data, come up with hypothesis. Possibly using deep learning. But cost need to be low, this is still a problem.
   - ➢ If you get one improvement, you need to give up something else. This rule in general holds, except you have clever algorithms, where you get things done without extra space. And this does happen sometimes.
   - ➢ Some other guesses: Use deep learning models to do the job. Calculate query statistics going along so that we can get better performance later. Or transform the query plan into a form that is better for approximating, or at least approximate the part we are confident.
4. Do we need to change execution engine when inserting the sampler?
   - ❖ Don't need to change. There are two ways to implement: treat it as high level operator, insert into query plan. QUICKR use this approach.
5. Who uses AQP?
   - ❖ A paper is briefly presented. “No Silver Bullet”? It summaries characteristics of AQP but also points out it is difficult to adopt. Then it offers suggestions both from theory and practice.
   - ❖ Examples of using AQP: Facebook may use them internally. Querying large dataset, real-time click logs, sensor networks, analog systems, and also from architecture community where people do approximation on CPU computation.
6. What does it mean by biased input for distinct sampler?
   - ❖ Due to its design, it grabs easy things first.
7. The distinct sampler only picks skewed data, but how do you know it is skewed in the first place?
   - ❖ Heavy header analysis, first is not approximated, but gathered statistics used for later.
8. Any interface the samplers provide for error bound, time bound?
   - ❖ Only confidence interval is provided.
9. Why place samplers closer to the source
   - ❖ The closer you get, more gains you get as indicated by experiments.
10. How often to use the samplers?
    - ❖ Uniform is used 2x as much as distinct and universe. But it is based on TPC-DC where join happens more or less different from real production.