Summary of "BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data"

Peter Paquet (pmpaquet), Wenting Tan (wentingt)

Problem and Motivation

Increasingly, query applications demand near real-time response rate. Many approximation techniques have been proposed to trade-off query accuracy for response time, including sampling, sketches and online aggregation (OLA). However, existing sampling and sketch based solutions make strong assumptions about the predictability of workloads, and thus loses the generality of the queries they support; online aggregation systems have highly variable performance in terms of both accuracy and response time. Thus, a query processing system that achieve a balance between efficiency and generality is needed.

Hypothesis

Predictable query column set (QCS) as the workload model: the exact values in the columns are unpredictable, but the frequency of the sets of columns used for grouping and filtering is stable over time.
Achieve bounded response time or bounded error according to user specification.
Scalability: distributed queries on 100-machine cluster, where each node stores 100GB data should have latency within tens of seconds.
When doing sample creation, assume that underlying data contains many rare subgroups.
When doing sample selection, assume that the statistical error is proportional to 1 divided by the square root of subsample size, and that the latency is proportional to subsample size.

Solution Overview

System Architecture

BlinkDB extends Apache Hive framework by adding BlinkDB Query Interface above the Hive Query Engine, adding sample selection module and uncertainty propagation module above the execution layer Shark, and adding sample creation and maintenance module above the storage layer HDFS.

Programming Model

Provides approximate results for SQL aggregation COUNT, AVG, SUM, and QUANTILE with grouping and filtering, and annotated with an error bound, e.g. ERROR WITHIN 10% AT CONFIDENCE 95%, or a time constraint, e.g. WITHIN 5 SECONDS.

Uncertainty Propagation module

Modify all pre-existing aggregation functions with statistical closed form to return error bars and confidence intervals in addition to the result.

Runtime Sample Selection

Select one or more samples from all samples created: If there are samples whose QCS is a superset the query's QCS, pick the sample(s) with smallest QCS cardinality; otherwise pick the sample(s) with highest selectivity (number of rows selected divided by number of rows read), by running the query on in-memory subsamples of all samples in parallel.

Create Error Latency Profile (ELP) to select the sample with highest selectivity (lower latency or lower error), and select appropriate subsample size to run the query: For each sample selected previously, run the query with increasing subsample size, then estimate parameters including query selectivity, sample variance, and input data distribution based on error profile, or the parameter data processing rate based on latency profile. Finally pick the best sample, and calculate the number of rows, i.e. the subsample size, for the query to run on.
Bias correction: For each sample, maintain a hidden column of effective sampling rate to weigh different subgroups and correct statistical bias due to stratified sampling.
Offline Sample Creation
BlinkDB selects QCSs by formulating the optimization problem as a mixed integer linear program (MILP), and create multi multi-dimensional uniform or stratified samples using distributed reservoir or binomial sampling techniques.
The MILP aims to maximize the weighted sum of the coverage of past queries of these QCSs, under the constraint of storage capacity budgets. It considers the sparsity of the data to support stratified sampling. It also supports uniform sampling in the degenerate case.
Background sample maintenance
It periodically rerun the MILP against workload changes to ensure optimal QCSs are selected. It also recompute the samples daily, to reduce the correlation among the answers to the different queries using the same sample.

Limitations and Possible Improvements
Supports only a limited set of SQL-style declarative queries.
Scalability suffers under the bulk queries, due to high communication cost between machines storing the large number of data. This can be improved by shuffle optimization techniques.
The sample creation module is offline. Adding OLA technique could improve performance.
The MILP cannot generate QCSs of more than 3 columns, otherwise the query optimizer takes more than one minute to complete. Could trade-off more accuracy for performance.

Summary of Class Discussion
When running queries on subsamples, only the first rows are read. But since the samples were stored in a randomly distributed way among subgroups and samples, this would not cause additional error in sample selection.
Upon machine failures and restarts, whether the queries will be re-executed on the exact same subsamples is not very clear. The paper does not discuss much about fault tolerance.
In fig.10b, actual error can be zero because the sample is small, and the subsample is selected to be the entire sample.
In fig.10c, the scalability problem is said to be caused by communication overhead, but this may be caused by other internal limitations of the system. But we might wrongly assume that the same total query workload is distributed among all the machines. Instead, the same query workload is executed on each machine.
The ELP and the calculation is said to be two approaches. But it seems that ELP is a visualization of the estimation of the parameters, while calculation is the next step.