Random Sampling over Large Datasets

When building an interactive data exploration tool, it is important that developers allow users to connect to a legacy system, such as a PostgreSQL or Oracle database, and interact with their data in sub-second response times. However, as data sets have grown in size over the decades, the performance of blocking, aggregating queries to databases has suffered. To make this burden easier to bear, data exploration tools may provide incremental feedback on the results of these long-running queries as they continue to execute. The goal then is to provide statistically meaningful incremental feedback, the quality of which depends on the input samples provided to the aggregating query. This project investigates techniques for random sampling of relational databases as a means of facilitating incremental feedback. After examining existing literature on random sampling, we explored our own implementations on a Amazon Web Services (AWS) system running PostgreSQL, with the primary goal of developing a scheme that could be used to return pseudo-random samples from a dataset with high bandwidth.[1]

The following methods were used to evaluate the quality and performance of each approach to random sampling. To assess the quality of a sampling method, given knowledge of the data, we streamed out the results of a sampling query, examining which rows were returned (based on row ID) to verify that there were no unexpected patterns. In addition, we also computed the mean and standard deviation of the data returned by the sampling queries, as a function of time. We compared the sample mean and sample standard deviation to the true mean and standard deviation of the dataset, then analyzed the results in context of the law of large numbers. Specifically, after streaming out a sufficiently small percentage of the data set, if we found that the sample mean and sample standard deviation were not within five percent of the true mean and standard deviation, then we concluded that the sampling method was not random. To gauge the performance of a sampling method, we examined the number of samples (and the total size in bytes) returned over time in order to determine the bandwidth of our sampling scheme.

We explored several approaches to random sampling. After initial experimentation, we determined that the benefits of page locality were so great that a series of linear traversals through a table was the only tractable approach for performant random sampling. Subsequently, one approach we used was to define a Bernoulli random variable for each row of the table, returning that row as a sample with some probability. A second approach was to partition the dataset using a modulus over row ID, then return a different equivalence class on each call for random samples. A third approach utilized a linear congruential generator to govern how samples were returned. These three approaches were each augmented to ensure that duplicate samples were not returned, then compared against native implementations for random sampling, including SQL's `TABLESAMPLE` clause and a naive approach that used the `ORDER BY`

[1] All of our code from this project is on Github: https://github.com/evanfuller/sampling-research

`RANDOM` clause. By using C-language user-defined functions and PostgreSQL's libpq library, we achieved a bandwidth of approximately 70 MB per second.