CS511 Project Evaluation Plan

Thesis

This project reproduces column sketch, a robust encoding scheme for accelerating evaluating predicates on columnar data for modern OLAP database workload, on top of the open source Apache Parquet data format. This project attempts to argue that column sketch can bring performance improvement in addition to the existing techniques like zone map and page index in Parquet, especially in cases where the target column is not clustered and the predicate selectivity is not extremely low.

Claim

The project claims that a parquet table augmented with column sketch encoding is superior to plain parquet file on the following task: reading and scanning the table while evaluating a predicate on a certain column, producing a batch of records satisfying the predicate, and the column is not clustered or sorted and the predicate is not of super low selectivity. The plain parquet file will use parquet's default optimizer to derive the native encoding and compression scheme. On the other hand, the augmented parquet file will contain a column of compressed code for the target column on which the predicated is evaluated and a serialized compression map. Given the specific workload and dataset distribution, we believe that existing techniques like zone maps and page indexes both have difficulties producing a satisfactory result, which is where column sketch could come in.

Evaluation Design

Due to the exploratory nature of the project, the evaluation is designed in several stages, aiming at:

1. Reproducing and evaluating the reproduction of column sketch on an in-memory micro-benchmark setups

This micro-benchmark is designed to be a reproduction of the experiment done in the original column sketch paper, which performs the task of pure in-memory scan and predicate evaluation.

Task: given an array of original data and any auxilary data structured (like min/max metadata or index), and a predicate to be evaluated, output a bitvector of the positions of values that satisfy the given predicate.

Independent variables: the auxilary structures and algorithms for using the structures to perform the predicate evaluation and scanning, and the statistical properties of the input data like distribution and sortedness.

Dependent variables: task completion time, memory bytes accessed per record, etc.

For this evaluation, we chose the baseline to be normal zone-maps. That is, the input is a `Vec<Vec<f64>>` simulating multiple chunks of values in a complete column and a `Vec<Statistics>` representing the zone-map metadata for each chunk. Our column-sketch-based approach accepts a `Vec<Vec<f64>>` and a `Vec<Vec<u8>>` and a `CompressionMap`, which correspond to the original input array (big), the compressed code array (small) and the compression map (constant size). The input data is normally distributed 64-byte floating point values of roughly 500MB in total. The predicate is checking the greater-than condition against a random floating point value. We collect the metrics of task completion time and memory bytes accessed during the task.

1. Evaluate the effectiveness and shortcomings of zone map and page index in Apache Parquet files

For this evaluation, we aim at showing the deficiency of existing zone map and page index techniques in some controlled experiment steps.

Task: Run a table scan followed by filter evaluation on dataset of controlled statistcial properties. The exact implementation of the task is adapted from the physical plans of apache arrow-datafusion to simulate a realistic workload.

Independent variables: the distribution and clustered-ness of the input dataset and the selectivity of predicate.

Dependent variables: the effectiveness of the data-skipping provided by zone maps and page indexes.

We perform the experiment along two axis: the auxillary structure used for query acceleration, i.e., zone maps or page index, and the clustered-ness of the dataset and selectivity of the query. We leverage parquet's row-group and column chunk architecture to collect metrics about how many rougroups and chunks are filtered by the existing techniques. This experiment intends to show an illustrative study of why existing techniques fall short in certain workloads and on dataset of certain properties.

1. An end to end evaluation

This experiment is the final end-to-end testing about the performance of column-sketch-enhanced parquet scanning in a realistic workload adapted from the physical execution plans of datafusion.

Task: The task is roughly the same as the previous experiment. We adapted the physical execution plan of arrow-datafusion (specifically the filter and parquet scan operator), which boils down to produce a vector of `RecordBatches`, which is roughly a chunk of row values in columnar format in arrow terminology, from a given parquet file and a predicate.

Independent variables: The first dimension of the independent variables are the combination of encoding, compression and the algorithms used for performing the task. In this experiment we test three configurations: Apache parquet native zone map, Apache parquet page index, and colun sketch. The second dimension is the dataset distribution and the predicate selectivity.

Dependent variables: We focus on the latency and throughput for finishing the end-to-end job for this holistic experiment.

The experiment is set up to use dataset generated from a custom tool we implemented, which can provide fine-grained control of the distribution and clustered-ness of the dataset.

Final Writeup

This project reproduces column sketch, a robust encoding scheme for accelerating evaluating predicates on columnar data for modern OLAP database workload, on top of the open source Apache Parquet data format. This project attempts to argue that column sketch can bring performance improvement in addition to the existing techniques like zone map and page index in Parquet, especially in cases where the target column is not clustered and the predicate selectivity is not extremely low. We claim that a parquet table augmented with column sketch encoding is superior to plain parquet file on the following task: reading and scanning the table while evaluating a predicate on a certain column, producing a batch of records satisfying the predicate, and the column is not clustered or sorted and the predicate is not of super low selectivity. The plain parquet file will use parquet's default optimizer to derive the native encoding and compression scheme. On the other hand, the augmented parquet file will contain a column of compressed code for the target column on which the predicated is evaluated and a serialized compression map. Given the specific workload and dataset distribution, we believe that existing techniques like zone maps and page indexes both have difficulties producing a satisfactory result, which is where column sketch could come in.

Due to the exploratory nature of the project, the evaluation is designed in several stages, aiming at the following three goals respectively:

1. Reproducing and evaluating the reproduction of column sketch on an in-memory micro-benchmark setups
2. Evaluate the effectiveness and shortcomings of zone map and page index in Apache Parquet files
3. End to end evaluation

Firstlt, a micro-benchmark is designed to be a reproduction of the experiment done in the original column sketch paper, which performs the task of pure in-memory scan and predicate evaluation. Given an array of original data and any auxilary data structured (like min/max metadata or index), and a predicate to be evaluated, output a bitvector of the positions of values that satisfy the given predicate. We control the auxilary structures and algorithms for using the structures to perform the predicate evaluation and scanning, and the statistical properties of the input data like distribution and sortedness, and benchmark the task completion time, memory bytes accessed per record, etc.For this evaluation, we chose the baseline to be normal zone-maps. That is, the input is a `Vec<Vec<f64>>` simulating multiple chunks of values in a complete column and a `Vec<Statistics>` representing the zone-map metadata for each chunk. Our column-sketch-based approach accepts a `Vec<Vec<f64>>` and a `Vec<Vec<u8>>` and a `CompressionMap`, which correspond to the original input array (big), the compressed code array (small) and the compression map (constant size). The input data is normally distributed 64-byte floating point values of roughly 500MB in total. The predicate is checking the greater-than condition against a random floating point value. We collect the metrics of task completion time and memory bytes accessed during the task.

Secondly, we evaluate the effectiveness and shortcomings of zone map and page index in Apache Parquet files. For this evaluation, we aim at showing the deficiency of existing zone map and page index techniques in some controlled experiment steps. We run a table scan followed by filter evaluation on dataset of controlled statistcial properties. The exact implementation of the task is adapted from the physical plans of apache arrow-datafusion to simulate a realistic workload. We test the effectiveness of the data-skipping provided by zone maps and page indexes against different data distribution. We perform the experiment along two axis: the auxillary structure used for query acceleration, i.e., zone maps or page index, and the clustered-ness of the dataset and selectivity of the query. We leverage parquet's row-group and column chunk architecture to collect metrics about how many rougroups and chunks are filtered by the existing techniques. This experiment intends to show an illustrative study of why existing techniques fall short in certain workloads and on dataset of certain properties.

Thirdly, we do an integrated final end-to-end testing about the performance of column-sketch-enhanced parquet scanning in a realistic workload adapted from the physical execution plans of datafusion. The task is roughly the same as the previous experiment. We adapted the physical execution plan of arrow-datafusion (specifically the filter and parquet scan operator), which boils down to produce a vector of `RecordBatches`, which is roughly a chunk of row values in columnar format in arrow terminology, from a given parquet file and a predicate. The first dimension of the independent variables are the combination of encoding, compression and the algorithms used for performing the task. In this experiment we test three configurations: Apache parquet native zone map, Apache parquet page index, and colun sketch. The second dimension is the dataset distribution and the predicate selectivity. We focus on the latency and throughput for finishing the end-to-end job for this holistic experiment. The experiment is set up to use dataset generated from a custom tool we implemented, which can provide fine-grained control of the distribution and clustered-ness of the dataset.

####