Creating Statistics During Bulk Load

NOTE: The features described on this page are Technology Preview (Complete But Not Tested) features in Trafodion Release 1.1.

Introduction

Trafodion requires the presence of detailed, accurate statistics to be able to perform cost-based optimization of queries. These statistics take the form of histograms that reflect the distribution of data values, as well as the number of distinct values and high-frequency (skew) values. The cost of generating these statistics has long been a problematic issue for Trafodion and its predecessor systems. Many performance improvements have been made, but updating statistics remains a bottleneck for many applications. The basic methodology for statistics collection involves scanning a table (usually selecting a sample of the rows), sorting, grouping, and forming intervals for an equi-height histogram for each column. These resource-intensive operations place limits on how fast the overall task can be accomplished. Moreover, as the content of the table changes over time, the entire process must be repeated to replace histograms which have become stale and no longer accurately represent the current table content.

This blueprint describes an alternative approach to constructing histograms, while preserving the existing form of the histograms. Rather than sorting the data and grouping common values, a Counting Bloom Filter (CBF) can be used to record the frequency of each distinct value, and then used to build the histogram intervals, and to derive the distribution of frequency values needed for estimating UECs (unique entry counts) based on the sample used.

The immediate work proposed in this blueprint addresses the initial collection of statistics for a table initially populated by a bulk load. It includes adding Update Statistics as an optional task of the bulk load utility (controlled initially by a CQD, but possibly by amending the bulk load syntax in the future), creating the sample table by randomly selecting rows as they pass through the loader and writing them to a Hive table, and using the sample table in conjunction with CBFs and a new "fast stats" algorithm to create histograms for the table's columns upon conclusion of the bulk load. Subsequent work will address making the sample table persistent, and updating it (and perhaps the histograms as well) when the HBase flush mechanism creates a new HFile from a MemStore, but this work will be undertaken post-1.1.

Creating the Sample Table

When a previously empty table is populated via the Trafodion bulk load utility, rows will be randomly sampled as they pass through the loader for inclusion in the sample table, using the default sampling rate. The sample table will be an external Hive table in a fixed directory, with a naming convention that incorporates the fully-qualified name of the HBase table it represents. The choice of Hive for the sample table is due to the higher scan rates for Hive.

A small subset of the rows selected for the sample table (a sample of the sample, so to speak), will be used to form a rough estimate of the overall UEC for each column. One of the parameters when creating a CBF is $n$, the anticipated number of keys (distinct values) that will be represented by the filter. If $n$ is much lower than the actual number of unique values, the number of distinct values detected by the filter will be too low and their frequency of occurrence too high. If $n$ is much higher than the actual number of keys, the memory required by the filter will be excessive, which could limit the number of histograms produced in a single pass over the sample table and require extra I/O. Just as the eventual UEC estimate for a column of the table is derived from the data in the sample table using a linear weighted combination (LWC) estimator, the value we use for $n$ for a given column is derived from the data collected in the subsample. The frequency of occurrence for each distinct value in the subsample is determined, and used to build an array of frequency-of-frequencies, $f'$. A value of an element of this array, e.g. $f'[2]=10$, is interpreted as "there are 10 distinct values that each appear twice in the sample". To get the required frequency information for the subsample, a CBF could be used (with its own value of $n$ being set to the subsample size), but for the small amount of data involved, a hash table mapping value to frequency will probably be used.

The LWC estimator used to calculate the value of $n$ for each column's CBF, and later to calculate the UECs that are stored with the histograms, is a linear weighted combination of the Jackknife and Schlosser estimators. This is the same estimation method currently used by Update Statistics.

Generating Value/Frequency Pairs

When the bulk load has completed and the sample table is in place, generation of histograms for the table's columns can begin. The remaining steps will be described in terms of a single histogram, but it should be noted that, subject to prevailing memory pressure, multiple columns can be read from a single scan of the sample table.
Each value of the column in the sample table is counted in the column's CBF. The value is hashed with \( k \) different hash functions (typically, \( k = 5 \)), each hash function yielding a value in the range \( 0..m - 1 \), where \( m \) is the number of counters in the CBF. The counter at each of these \( k \) indexes is incremented, and the frequency of the value at that point is the minimum of those \( k \) counter values. Using multiple hash functions and taking the minimum counter value as the frequency reduces the effect of collisions and avoids overcounting the frequency of values. In general, CBF parameters are chosen such that the incidence of overcounting is less than 1 percent.

In addition to registering each value occurrence in the CBF, the minimum and maximum values are kept track of, so we know the range of values to be represented by the histogram for the column. Each distinct value is also added to a list when its first occurrence is registered in the CBF.

Upon completion of scanning all values of the column in the sample, we have the following information:

1. A list of the distinct values contained in the column.
2. The frequency of occurrence of each of these values, available via the CBF.
3. The minimum and maximum values contained in the column.

Creating an Equi-Width Histogram

Although the ultimate goal is an equi-height histogram, the intermediate step of creating an equi-width histogram produces a partial ordering of the data from which the final result can be created without the need for a full sort of the data. Knowing the minimum and maximum value contained in the column, we can partition the value range into intervals of equal width, and assign each distinct value to the appropriate interval with a simple calculation. For our purposes, assigning a value to an interval entails the following actions:

1. Incrementing the interval's UEC.
2. Adding the value's frequency of occurrence (from the CBF) to the interval's row count.
3. Updating the interval's min or max if the value is outside those bounds.

The notion of equal width requires a measure of linear distance between any two values of the domain. To make this possible for character types and make it easier to measure for other non-numeric (e.g., datetime, interval) types, before being allocated to an interval of the equi-width histogram, values are encoded in a canonical double precision form that preserves the ordering of the original data. The minimum and maximum values are also converted to this form. The target interval for a value \( v \) is then simply \( (e(v) - \text{min}) / \mu \), where \( e \) is the encoding function and \( \mu \) is the number of intervals in the equi-width histogram.

The frequency is also used to index into a frequency-of-frequency array, and the array element at that index is incremented. This frequency-of-frequency data is used by the LWC estimator along with the UECs from the sample data to estimate overall and per-interval UECs for the full table.

The placement formula for the equi-width histogram imposes a partial ordering of the data; for a given interval, although the contained values are not ordered within the interval, each is known to be greater than any value in a preceding interval, and less than any value in the intervals that follow. The equi-height histogram can now be derived from the equi-width histogram.

Creating the Final, Equi-Height Histogram

The final histogram distributes the value occurrences (not the distinct values) as evenly as possible among \( \mu \) intervals. \( \mu \) should be substantially greater than \( \hat{f} \); the larger \( \mu \) is compared to \( \hat{f} \), the smaller the subset of distinct values that need to be sorted will be. Knowing the total number of value occurrences and desired number of intervals, we can determine the target number of occurrences to represent in each interval of the final, equi-height histogram. Each distinct value must be wholly represented by a single interval, so it is usually not possible to hit this target height exactly for every interval. The frequency of a single distinct value may even exceed the target height, and will be accorded its own interval.
The width-to-height transformation proceeds as follows. Beginning with the first interval of the equi-width histogram, the row count of each successive interval is summed until the addition of the next interval would cause the target height of the equi-height histogram to be exceeded. That interval must be split between the first and second intervals of the equi-height histogram. Since the distinct values that comprise the interval are unordered, they must be sorted, at which point the lowest values are added to the 1st equi-height interval until it is full (i.e., achieves the target height). The remaining values become the first values assigned to the 2nd equi-height interval, and the lowest of these values is saved as the boundary value of the 2nd interval. We continue in this fashion until all the intervals of the equi-width histogram have been distributed to the equi-height histogram.

The final step, as with the original Update Statistics algorithm, is to scale up the row counts and UECs so that they represent estimates for the full table rather than the sample. While this is a simple scaling operation for row counts based on the sampling rate, the UEC estimation is more complicated, and is performed by the aforementioned LWC estimator.

References

