Multi-dimensional Data Statistics for Columnar In-Memory Databases

Curtis Kroetsch*
David R. Cheriton School of Computer Science
Faculty of Mathematics
University of Waterloo
Waterloo, Ontario, Canada
cckroets@uwaterloo.ca

ABSTRACT

The research presented here studies the multi-dimensional data statistics in the context of columnar in-memory database systems. Such systems, for example SAP HANA [4], SQL Server Apollo, or IBM BLU, use an order-preserving dictionary with dense encoding on the read-optimized storage which encodes the values of a single column in an ordered, dense-domain dictionary. The dictionary maps variable-length domain values to fixed-size dictionary entries. This encoding reduces memory consumption as dictionary values can be represented with fewer bits than the original values, and allows queries to be evaluated efficiently on the encoded data. The main characteristics of the dictionary encoding is that it results in a dense domain of values which can be exploited for building efficient data statistics objects.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query Processing*

Keywords

Multi-dimensional data statistics, Histograms

1. INTRODUCTION

Columnar in-memory databases divide its data into two storages: one read-optimized storage, aka *main storage*, and one write-optimized storage, aka *delta storage*. The delta storage is merged into main storage in a process known as *delta-merge* operation. During a delta-merge operation, for a specific column, the column dictionary will be rebuilt to reflect the data in the new main storage. Hence, any structure built using the main's dictionary has to be rebuilt after a delta-merge operation. For example indexes, materialized views, or data statistics built on main storage have to be rebuilt after a delta-merge operation. In this context, the requirements for data statistics objects to be fast to (re)build, space efficient, and precise are more stringent than in a classical database system, as the benefit for using dense-domain dictionaries must be balanced to the requirement for rebuild after each delta-merge operation.

The work presented in ACM SIGMOD 2014 in [3], in the context of SAP HANA, shows how ordered dictionaries are exploited for the efficient construction of precise and concise one-dimensional histograms. One-dimensional data statistics built for dictionary-encoded columns can be made very efficient from point of view of space, built time, and quality of estimates [3].

In contrast, we study here if multi-dimensional histograms can also be made space-efficient, built-time efficient, and have good quality of estimates when built on ordered dictionary encoded data, hence on dense domains of values.

In this paper we present some preliminary results on building and using multi-dimensional histograms based on r-trees [1], similar to rK-Hist proposed in [2]. The main focus is to improve the construction algorithms, from point of view of build-time, space efficiency, and the quality of the estimates for range queries. The proposed design and implementation proved so far to provide very accurate cardinality estimates for the data points with dense integer domains while its space and build time is comparable to previous work.

The purpose of a multi-dimensional histogram is to summarize a dataset over d dimensions. A dataset consists of N distinct points, each with d values and a measure. A point p has the form $p = (v_1, \cdots, v_d)$ where v_i represents the value of dimension i in p. Each point in the dataset also has an associated measure equal to $f(p) \in \mathbb{Z}^+$ which is the number of occurrences of the point p in the dataset. For a dataset with N distinct points, there are $M = \sum_{i=1}^{N} f(p_i)$ non-distinct points. A histogram for a dataset is a collection of buckets, where each bucket stores some information about a hyper-space it represents. Given a range query Q over d dimensions, the histogram estimates the number of points in the dataset that intersect with Q.

We designed a series of improvements to the original rK-Hist algorithms presented in [2], the most important are described below.

*The student author and the adviser, Dr. Anisoara Nica, SAP AG, Waterloo, Ontario, Canada, Anisoara.Nica@sap.com, worked on this project during the student's co-op term in Summer 2013.

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.
Copyright 2014 ACM 978-1-4503-2376-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2588555.2612663 .

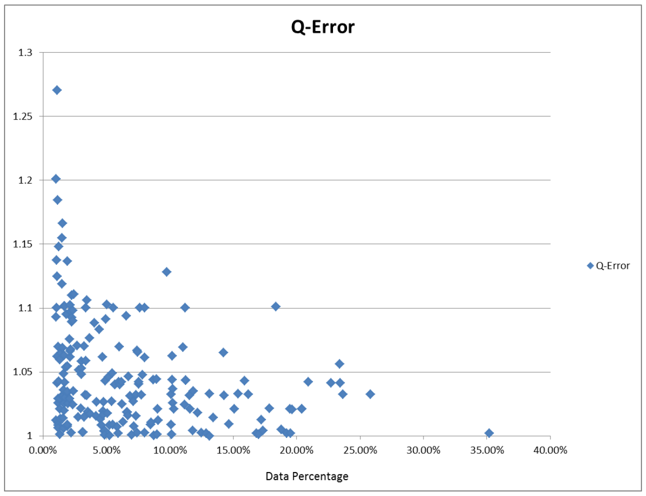

Figure 1: Cardinality Estimate Accuracy:
6-dimension points;
10 million distinct points;
516896321 non-distinct points;
200 range queries;
q-error = $max\{$ (actual cardinality / estimated cardinality), (estimated cardinality / actual cardinality) $\}$
q-error: Average: 1.0449, Maximum: 1.2702, Minimum: 1.0000

Improvement to Bucket Decomposition Method: The initial bucketing scheme may produce sub-optimal, non-uniform buckets which can lead to poor cardinality estimations. Bucket decomposition is one of the optimizations the rK-Hist uses to produce better results. Consequently, the rK-Hist uses a cost function to evaluate the uniformity of a bucket to determine if the points inside should be re-bucketed. Specifically, instead of creating the desired number of buckets initially, only 90% are created. The cost function is applied to each bucket and 5% of the absolutely worst buckets are considered for decomposition. Then a "sliding window" [2] approach is used to decompose each of the worst buckets into two buckets of greater quality if possible. These newly created buckets are also considered for further decomposition. Decomposition continues until enough buckets are created or the buckets can no longer decompose.

For the bucket decomposition algorithm, we observed that once a bucket ω becomes decomposed into ω_1 and ω_2, these new buckets are likely to be more uniform than those outside of the 5% decomposition list. This is because the 'sliding window' technique is typically very affective in removing dead space from the bucket. Decomposing further ω_1 and ω_2 is a waste if there are worse buckets to consider. So instead of initially moving the worst buckets to a separate list, we store all buckets in a priority queue Υ, according to their cost. When decomposing, the worst bucket, which appears at the top of Υ, is processed. Newly created buckets are also inserted back into Υ, hence the best bucket to be considered for decomposition is the bucket with the worst quality according to the cost function. Algorithm 1 describes the procedure in more detail.

Improvement to cost function C: The original cost function C in Algorithm 1 is calculated based on the K-Uniformity metric of a bucket. This metric aims to assess the relative emptiness of a bounding box. This is done by recursively partitioning the points in a bucket into their own box, via a k-d-tree. The K-Uniformity is the standard deviation of the volumes of each box. The higher this metric, the worse the distribution is inside the bucket.

Algorithm 1 Improved Bucket Decomposition

```
Input: A set $B$ of $I$ initial buckets, a cost function $C$, and
  the total buckets to be produced $\alpha$.
Output: A set $\Upsilon$ of $\alpha$ buckets.
  $\Upsilon \leftarrow PriorityQueue(B, C)$
  while $size(\Upsilon) < \alpha$ do
    $\omega \leftarrow ExtractMax(\Upsilon)$
    $(\omega_1, \omega_2) \leftarrow Decompose(\omega)$
    $Push(\Upsilon, \omega_1)$
    $Push(\Upsilon, \omega_2)$
  end while
```

The complexity of calculating this metric for a bucket ω, where $n = |\omega|$ is $O(n \lg n)$. This computation also requires that the original points be read again (potentially bring back into main memory) which is very expensive. We use instead $C(\omega)$ = Volume(minimum bounding box (ω)) [1] which is much cheaper to compute. The main advantage is that it doesn't require revisiting the raw data again and has a much smaller complexity, $O(d)$ for a d-dimensional dataset.

Figure 1 shows the cardinality estimate accuracy for randomly generated range queries, on large datasets of 516896321 non-distinct points. We measure the q-error metrics as introduced in [3]. The x-axis represents the actual cardinality (in percentage) of the generated queries.

2. CONCLUSION

We studied a certain type of multi-dimensional data statistics for dense domains which naturally characterize the dictionary encoded columns in a columnar in-memory databases. We plan to further exploit the properties that characterize such systems in building and maintaining different types of data statistics objects which are efficient and accurate.

3. REFERENCES

[1] Daniar Achakeev and Bernhard Seeger. A class of r-tree histograms for spatial databases. In *SIGSPATIAL/GIS*, pages 450–453, 2012.

[2] Todd Eavis and Alex Lopez. Rk-hist: an r-tree based histogram for multi-dimensional selectivity estimation. In *ACM CIKM*, pages 475–484, 2007.

[3] Guido Moerkotte, Dave DeHaan, Anisoara Nica, Norman May, and Alexander Boehm. Exploiting ordered dictionaries to efficiently construct histograms with q-error guarantees in SAP HANA. In *ACM SIGMOD*, 2014.

[4] Vishal Sikka, Franz Färber, and Anil Goel andWolfgang Lehner. SAP HANA: The evolution from a modern main-memory data platform to an enterprise application platform. In *PVLDB*, volume 6, pages 1184–1185, 2013.