Workload-Aware Histograms for Remote Applications

Tanu Malik and Randal Burns

[1] Cyber Center
Purdue University
[2] Department of Computer Science
Johns Hopkins University
tmalik@cs.purdue.edu, randal@cs.jhu.edu

Abstract. Recently several database-based applications have emerged that are remote from data sources and need accurate histograms for query cardinality estimation. Traditional approaches for constructing histograms require complete access to data and are I/O and network intensive, and therefore no longer apply to these applications. Recent approaches use queries and their feedback to construct and maintain "workload aware" histograms. However, these approaches either employ heuristics, thereby providing no guarantees on the overall histogram accuracy, or rely on detailed query feedbacks, thus making them too expensive to use. In this paper, we propose a novel, incremental method for constructing histograms that uses minimum feedback and guarantees minimum overall residual error. Experiments on real, high dimensional data shows 30-40% higher estimation accuracy over currently known heuristic approaches, which translates to significant performance improvement of remote applications.

1 Introduction

Recently several applications have emerged that interact with databases over the network and need estimates of query cardinalities. Examples include replica maintenance [1], proxy caching [2], and query schedulers in federated systems [3]. Accurate estimates of query cardinality are required for core optimization decisions: pushing updates versus pulling data in replica systems, caching versus evicting objects in proxy caches, choosing a distributed query execution schedule in a federated system. Optimal decisions improve performance of these applications.

We are interested in the performance of Bypass Yield (BY) cache, a proxy caching framework for the National Virtual Observatory (NVO), a federation of astronomy databases. BY caching reduces the bandwidth requirements of the NVO by a factor of five. It achieves this reduction by replicating database objects, such as columns (attributes), tables, or views, near clients so that queries to the database may be served locally, reducing network bandwidth requirements. BY caches load and evict the database objects based on their expected yield: the size of the query results against that objecti.The five-fold reduction in bandwidth is an upper bound, realized when the cache has perfect, a priori knowledge of query result sizes. In practice, a cache must estimate yield or equivalently the cardinality of the query.

Query cardinality estimation has long been employed by query optimizers to evaluate various query execution plans. For this, the most popular data structure is the

I.-Y. Song, J. Eder, and T.M. Nguyen (Eds.): DaWaK 2008, LNCS 5182, pp. 402–412, 2008.

histogram. However, traditional approaches for constructing histograms do not translate to proxy caches. Traditional approaches [4,5] require complete access to data, and involve expensive data scans. Distributed applications such, as proxy caches, however, are remote from data sources and are severely constrained in the data they have access to or resources they can expend on query cardinality estimation [6]. Resource constraints include lack of access to source data, or limitations in storage space, processing time, or amount of network interactions with the remote data sources.

Distributed applications benefit from ”workload-aware” histograms that are constructed by exploiting workload information and query feedback. The key idea is to use queries and the corresponding feedback to construct a data distribution over heavily accessed regions while allowing for inaccuracies in the rest of the regions. However, previous approaches [7,8,9] for constructing workload aware histograms either require extremely detailed query feedback or lack in consistency and accuracy. Approaches such as STHoles [7] and ISOMER [8] require sub-query cardinality feedback at each level of a query execution plan to construct a consistent and accurate histogram. These approaches are practical for applications that are co-located with the data source. For remote applications, such detailed feedback not only comes at a high cost, it is also redundant: The applications only need a top level query cardinality estimate; estimating sub-query cardinality is an over-kill [10]. STGrid [9] is seemingly suitable for distributed applications as it requires only the query and its feedback to construct a histogram. However, it uses heuristics to refine the histogram leading to inconsistencies and inaccuracies in the constructed histogram.

In this paper, we propose a novel technique for histogram construction that is suitable for distributed applications such as the Bypass-Yield proxy cache. It uses queries and only the top-level cardinality feedback to learn the data distribution of a cached object. The histogram is maintained incrementally in that query feedbacks refine the bucket frequencies. Unlike previous approaches [9] in which the refinement is done heuristically, our technique uses recursive least squares technique to gradually ”regress” the distribution. This incremental regression technique takes into account all previous query feedbacks and thus produces a consistent histogram. Histogram accuracy is maintained as the refinement procedure avoids inclusion of erroneous assumptions and heuristics into the histogram. In addition, the overhead of our technique has a negligible cost in constant time for each query feedback, making it extremely suitable for resource-constrained distributed applications.

Extensive experimental results show that refining a histogram consistently and in a principled manner leads to 30-40% higher accuracy. We conduct our experiments on the Sloan Digital Sky Survey (SDSS), which is the largest federating site of the NVO. We generate synthetic workload whose access patterns mimic the workload logs of the SDSS When the cardinality is known apriori, BY caches reduce network traffic requirements of SDSS by a factor of five. Our workload-aware histograms reduce network requirements by a factor of 5.5 in addition to being computationally and space efficient.

2 Related Work

There has been intensive research in constructing accurate histograms for cardinality estimation in query optimizers. Chen and Roussoupoulos [11] first introduced the idea

of constructing a data distribution using queries and their feedback. However, they approximate a data distribution of a relational attribute as a pre-chosen model function and not a histogram. By virtue of using histograms our approach easily extends to multi-dimensional queries, which are not discussed by Chen et. al. and left as future work.

Using queries and its cardinality, STGrid [9] progressively learns a histogram over a data distribution. A histogram is refined by heuristically distributing the error of the *current* query amongst histogram buckets that overlap with the selectivity range of the query. While this gives a very fast algorithm for refining the histogram, it lacks in consistency. It favors in reducing error for the current query but does not take account feedback from previous queries over the same histogram.

In STHoles [7] and ISOMER [8], a superior histogram structure is proposed, which improves accuracy, but at the expense of additional feedback [10]. This additional feedback comes either by constructing queries with artificial predicates and obtaining their cardinality from the database, or by closely monitoring the query execution plan of each query and obtaining feedbacks at every level of the plan. This makes these approaches suitable for query optimizers, which have closer proximity to data, and can monitor and create such feedbacks with low overhead. However, they are an overkill for distributed applications which are oblivious to query execution plans, resource constrained and need only a top-level cardinality estimate [10]. In this paper, we have introduced consistent and efficient workload-aware histograms that can be used for cardinality estimation in remote, distributed applications such as the Bypass Yield proxy cache to make core optimization decisions.

3 Cardinality Estimation in BY Caches

In this section, we briefly describe the core caching decision made by the bypass yield application, how cardinality estimates are required and used in cache replacement algorithms, and metrics which define the network performance of the system. We then introduce workload-aware histograms which can be used in Bypass caches for cardinality estimation.

3.1 The BY Cache Application

The bypass-yield cache [2] is a proxy cache, situated close to clients. For each user query, the BY cache evaluates whether to service the query locally, loading data into the cache, versus shipping each query to be evaluated at the database server. By electing to evaluate some queries at the server, caches minimize the total network cost of servicing all the queries.

In order to minimize the WAN traffic, a BY cache management algorithm makes an economic decision analogous to the rent-or-buy problem [12]. Algorithms choose between loading (buying) an object and servicing queries for that object in the cache versus bypassing the cache (renting) and sending queries to be evaluated at sites in the federation. The BY cache, caches objects such as tables, columns or views. At the heart of a cache algorithm is byte yield hit rate (`BYHR`), a savings rate, which helps the cache

make the load versus bypass decision. BYHR is maintained on all objects in the system regardless of whether they are in cache or not. BYHR is defined as

$$\text{BYHR} = \sum_j \frac{p_{i,j} y_{i,j} f_i}{s_i^2} \tag{1}$$

for an object o_i of size s_i and fetch cost f_i accessed by queries Q_i with each query $q_{i,j} \in Q_i$ occurring with probability $p_{i,j}$ and yielding $y_{i,j}$ bytes. Intuitively, BYHR prefers objects for which the workload yields more bytes per unit of cache space. BYHR measures the utility of caching an object (table, column, or view) and measures the rate of network savings that would be realized from caching that object. BY cache performance depends on accurate BYHR, which, in turn, requires accurate yield estimates of incoming queries. (The per object yield $y_{i,j}$ is a function of the yield of the incoming query.) Since the yield of the incoming query is a function of the number of tuples, we estimate cardinality for each query.

3.2 Learning Histograms

Histograms are the most popular and general model used in databases for cardinality estimation. Histograms model the data distribution as piece-wise constant functions. In our workload-aware technique, histograms are initialized as a constant function (uniformity assumption), with a fixed number of buckets. With each query and its feedback, the current function is updated to obtain a new piecewise constant function, the latter providing a better approximation of the underlying data distribution. This function update is not arbitrary, but is based on past queries and their feedback. Specifically, that function is obtained which minimizes the sum of square of residual errors, i.e., the squared error between the actual and estimated cardinality over all queries. For this, cardinality estimation error over all queries need not be maintained and our technique uses the well-known recursive-least-square (RLS) algorithm to incrementally minimize the sum of residual errors. Periodically bucket boundaries are readjusted to give a new function which further improves accuracy. For ease of presentation, we describe the technique by constructing histograms using queries with single range clauses. We later revise the technique for queries with multiple range clauses. Finally, we show how bucket boundaries are periodically organized.

Single-Dimensional Histograms. Let A be an attribute of a relation R, and let its *domain* be the range $D = [A_{\min}, A_{\max}]$ (currently assuming numerical domains only). Let f_A be the actual distribution of A, and F_A be the corresponding cumulative distribution function (CDF). A *range* query on attribute A, $\sigma_{l \le R.A \le h}(R)$, where $l \le h$ is denoted as $q(l, h)$. The cardinality s of query q defined as $s_q = f_A([l, h])$, is the number of tuples in the query result, and is equal to

$$s = \textstyle\sum_l^h f(x) = F_A(h) - F_A(l)$$

The query feedback is then defined as $\tau = (l, h, s)$. Given several such feedbacks, the goal is to learn $\hat{f}_A$ (or equivalently $\hat{F}_A$), an approximation of f_A that gives the minimum sum of squared residuals.

We model $\hat{f}_A$ as a histogram of B buckets. The corresponding $\hat{F}_A$ is a piece-wise linear function with a linear piece in each bucket of the histogram.

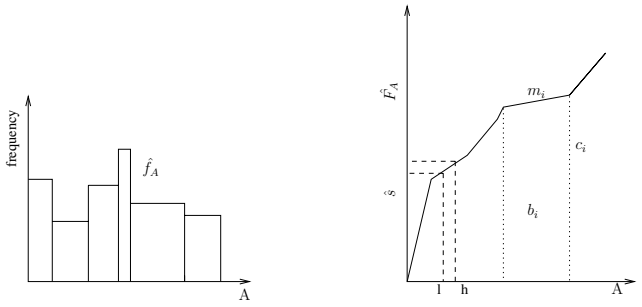

Fig. 1. The cumulative frequency distribution on attribute A can be learned to estimate query cardinality

To model (compute) $\hat{F}_A(\cdot)$, we need to maintain the following data: For each bucket i, the values $i_{\min}, i_{\max}$ are the boundaries of the bucket m_i the slope of the linear segment within bucket i, and c_i is the y-intercept of the linear segment. Thus $\hat{F}_A(x)$ is computed as: $\hat{F}_A(x) = m_i x + c_i, i \in B$, and $i_{\min} \leq x < i_{\max}$.

Using this form of approximation the estimated cardinality, $\hat{s}$ for a query $q = (l, h)$ is computed as follows:

$$\hat{s} = \hat{F}_A(h) - \hat{F}_A(l) = m_u h + c_u - (m_v l + c_v), \tag{2}$$

in which $u_{\min} \leq h < u_{\max}$ and $v_{\min} \leq l < v_{\max}$. Figure 1 shows a histogram of 6 buckets constructed over attribute A. The adjacent figure shows the corresponding $\hat{F}_A$ and how it can be used to estimate cardinality of queries. However, to estimate the cardinality precisely we need to approximate the parameters m_i and c_i in each bucket of $\hat{F}_A$. For this we first cast our parameters m_i and c_i as vectors. This gives a concise representation and a neat expression that can then be estimated using RLS.

$\mathbf{M} = (m_1, \ldots, m_B)$ is a $B \times 1$ vector in which each element is the slope of the bucket. Similarly, $\mathbf{C} = (c_1, \ldots, c_B)$ is a $B \times 1$ vector in which each entry is the y-intercept. To obtain $\hat{s}$, we cast the range values h and l in a $1 \times B$ vector in which the uth element is h, the vth element is $-l$ and the rest of the elements are 0. For the intercept, we define a $1 \times B$ unit vector in which the uth element is 1, the vth element is -1 and the rest of the elements are 0. This gives a concise vector notation for $\hat{s}$:

$$\hat{s} = \big((0, .., h_u, .., -l_v, .., 0)(0, .., 1_u, .., -1_v, .., 0)\big) \begin{pmatrix} \mathbf{M} \\ \mathbf{C} \end{pmatrix}, \tag{3}$$

The vectors $\mathbf{M}$ and $\mathbf{C}$ are estimated after a sequence of query feedbacks $\tau_1, \ldots, \tau_n$ have been collected, where $n \geq B$. A standard criterion [13] to find the optimal $\mathbf{M}$ and $\mathbf{C}$ is to minimize the sum of the squares of the estimation error or the least square error:

$$\sum_n (\hat{s} - s)^2 = \sum_n \big(\big((0, .., h_u, .., -l_v, .., 0)(0, .., 1_u, .., -1_v, .., 0)\big) \begin{pmatrix} \mathbf{M} \\ \mathbf{C} \end{pmatrix} - s\big)^2. \tag{4}$$

Alternatively, above problem can be reformulated as:

$$minimize \parallel \mathbf{X}\hat{\theta} - \mathbf{Y} \parallel^2, given \tag{5}$$

$$\mathbf{X}_{n\times 2B} = \begin{pmatrix} (0,..,h_u,..,-l_v,..,0)(0,..,1_{u1},..,-1_{v1},..,0) \\ (0,..,h_{u2},..,-l_{v2},..,0)(0,..,1_{u2},..,-1_{v2},..,0) \\ \cdots \\ (0,..,h_{uN},..,-l_{vN},..,0)(0,..,1_{uN},..,-1_{vN},..,0) \end{pmatrix},$$
$$\mathbf{Y}_{n\times 1} = \big(s_1, s_2, \ldots, s_n\big)^T, \text{ and}$$
$$\hat{\theta}_{2B\times 1} = \big(\mathbf{M}, \mathbf{C}\big)^T = \big(m_1, \ldots, m_B, c_1, \ldots, c_B\big)^T$$

If $\mathbf{X}^T$ is the transpose of $\mathbf{X}$ then the solution to above equation is

$$\hat{\theta}_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{6}$$

The above formulation assumes that all the feedbacks $\tau_1 \ldots \tau_n$ are known and thus $\hat{\theta}_{LS}$ can be computed by computing the inverse of X^TX. However, feedbacks come one at a time and we use recursive least square (RLS) [13] method to calculate $\hat{\theta}_{LS}$ incrementally. The RLS algorithm can be started by collecting 2B samples and solving 6 to obtain an initial value of $\hat{\theta}_{LS}$. Each incremental feedback now takes $O(B^2)$ time to compute.

The obtained $\hat{\theta}_{LS}$ is our $\hat{F}_A(\cdot)$. Since there is a one-to-one mapping between $\hat{F}_A(\cdot)$ and $\hat{f}_A(\cdot)$, the corresponding histogram can be easily obtained.

Input: Initialize $\mathbf{P}_N = [\mathbf{X}^T\mathbf{X}]^{-1}$ and $\hat{\theta}_N$ from an initial collection of N examples using (4) and (5)
foreach *subsequent example* $(\mathbf{x}_i, y_i)$ *compute* **do**
$\hat{\theta}_i = \hat{\theta}_{i-1} + \frac{\mathbf{P}_{i-1}\mathbf{x}_i(y_i - \mathbf{x}_i^T\hat{\theta}_{i-1})}{1+\mathbf{x}_i^T P_{i-1} x_i}$
$\mathbf{P}_i = \mathbf{P}_{i-1} - \frac{\mathbf{P}_{i-1}\mathbf{x}_i\mathbf{x}_i^T\mathbf{P}_{i-1}}{1+\mathbf{x}_i^T\mathbf{P}_{i-1}\mathbf{x}_i}$
end

Algorithm 1. Recursive least squares (RLS) algorithm

Multi-dimensional Range Queries. We show how the above formulation can be extended to multi-dimensional (m-d) range queries. For m-d range queries we need to learn $f_{\mathcal{A}}$ the actual distribution over the attribute set $\mathcal{A} = A_1, \ldots, A_k$ of R. The corresponding range query on $\mathcal{A}$ is $\sigma_{\bigwedge_d\ l_d \le R.A_d \le h_d}(R)$, in which $1 \le d \le k$ and $l_d \le h_d$. The cardinality of q is

$$s = \textstyle\sum_{x_1=l_1}^{h_1} \cdots \sum_{x_k=l_k}^{h_k} f(x_1, \ldots, x_k) = F_{\mathcal{A}}(h_1, \ldots, h_k) - F_{\mathcal{A}}(l_1, \ldots, l_k)$$

We model $\hat{f}_{\mathcal{A}}$ as a histogram of $B = B_1 \cdots B_k$ buckets. The corresponding $\hat{F}_{\mathcal{A}}$ is a piece-wise linear function with a planar segment in each bucket of the histogram. To model (compute) $\hat{F_{\mathcal{A}}}(\cdot)$, we need to maintain the following data: For each bucket i, and each attribute A_d the values $i_{d_{\min}}, i_{d_{\max}}$ are the boundaries of the bucket, m_{i_d} the slope of the planar segment along A_d within bucket i, and c_i is the intercept of the planar segment. Thus $\hat{F_{\mathcal{A}}}(x_1, \ldots, x_k)$ is computed as: $\hat{F_{\mathcal{A}}}(x_1, \ldots, x_k) = \sum_d m_{i_d} x_d\ + c_i$, where i is the bucket such that $i_{d_{\min}} \le x_d < i_{d_{\max}}$ for each attribute A_d.

Using this form of approximation the estimated cardinality, $\hat{s}$ for a query $q = q((l_1, h_1), \ldots, (l_k, h_k)\ q = (\{(l_d, h_d)\})$ is computed as follows:

$$\hat{s} = \hat{F_{\mathcal{A}}}(h_1, \ldots, h_k) - \hat{F_{\mathcal{A}}}(l_1, \ldots, l_k) \tag{7}$$

Given a sequence of feedbacks, an algebraic form similar to equation 3, and finally equation 5 can be obtained which can be minimized recursively using RLS.

In the above, we have maintained $\hat{F}(\cdot)$ corresponding to the sequence S of query-feedbacks we have received at any stage. For each query-feedback $\tau \in S$ there is a corresponding error squared wrt to $\hat{F}(\cdot)$, which is $(\hat{F}(h_1, \ldots, h_k) - \hat{F}(l_1, \ldots, l_k) - s)^2$. Based on the formulation of $\hat{F}(\cdot)$ and the RLS algorithm [13], we can prove the following theorem.

Theorem 1. *For any set of query-feedbacks S, the corresponding function $\hat{F}(\cdot)$ has the minimum sum of squared error with respect to S over all piece-wise linear functions defined over the set of buckets used by $\hat{F}(\cdot)$.*

4 Bucket Restructuring

In the above section, a histogram of B buckets is chosen to approximate the data distribution. However, the choice of B is arbitrary and a lower B can result in even lower error over the same workload. Bucket restructuring refers to the iteration process in which bucket boundaries are collapsed or expanded to obtain a histogram of B' buckets in which $B' \neq B$ such that the overall error is minimized. The restructuring process is not a learning process and does not depend on query feedback. Thus any good restructuring process suffices. In order to test the accuracy of our approach, in this paper, we have used the same restructuring process as described in [9]. Specifically, high frequency buckets are split into several buckets. Splitting induces the separation of high frequency and low frequency values into different buckets, and the frequency refinement process later adjusts the frequencies of these new buckets. In order to ensure that the number of buckets assigned to the histogram does not increase due to splitting, buckets with similar frequencies are reclaimed. The restructuring process is performed periodically.

5 Experiments

The goal of our experiments is two-fold: Firstly to test the accuracy of an incremental, self-tuning approach in learning histograms and secondly to measure the impact of the learned histogram on the performance of the BY cache. We first describe our experimental setup and then report our results.

Datasets. For both the experiments we use a synthetic workload over a real astronomy database, the Sloan Digital Sky Survey. The Sloan Digital Sky Survey database consists of over 15TB of high dimensional data. These dimensions consist of various properties of the astronomical bodies. Over here we show results for one, two and three dimensional datasets.The datasets are a projection of light intensities and spatial coordinates of 5,000,000 astronomical bodies obtained from the Sloan Digital Sky Survey. In 1-dimension, the data is often queried for light intensities and in 2 and 3-dimensions, for spatial coordinates. Data density varies significantly in the patch of sky that we have considered ranging from million of bodies in 1 arc second to tiny black holes with no

objects. One of our motivations of using real data is that while the authors in [9] report high accuracy of their technique over synthetic data, as our experiments show, is not true over real data that have complex density functions.

Workload. We generate synthetic workloads (similar to [9]) consisting of random range queries in one or more dimensions. Our workload represents queries as seen in the real workload log of the SDSS. While the corner points of each selection range are generated independently, workloads exhibit locality of reference. The attribute values used for selection range corner points in these workloads are generated from piecewise uniform distributions in which there is an 80% probability of choosing a value from a locality range that is 20% of the domain. The locality ranges for the different dimensions are independently chosen at random according to a uniform distribution.

Histograms. We use 100 buckets for 1-d histograms and 50 buckets per dimension for 2-d and 3-d histograms, respectively. The one, two and three dimensional histograms occupy 1.2, 10.5 kilobytes of memory, respectively, which is much smaller than constructing histograms over the entire dataset. The histogram is first constructed using a training workload and its accuracy then tested over a test workload which is statistically similar. To measure accuracy over a test workload, we use the average relative estimation error i.e., (100 * abs(actual result size - estimated result size) / N * actual result size) to measure the accuracy.

Evaluation. Finally, we compare our approach (denoted here as least squares (`LS`)) with `STGrid`, which also uses minimum feedback to construct histograms in one and higher dimensions. STGrid uses the heuristic that buckets with higher frequencies contribute more to the estimation error than buckets with lower frequencies. Specifically, they assign the "blame" for the error to the buckets used for estimation in *proportion to their current frequencies* [9].

Table 1. Error Comparison with Increasing Training Workload

Dim	1		2		3	
Wkld Size	`LS`	`STGrid`	`LS`	`STGrid`	`LS`	`STGrid`
2000	3.04%	5.70%	13.29%	15.14%	23.15%	49.56%
2500	3.13%	3.23%	12.15%	19.29%	21.87%	49.13%
3000	2.19%	2.12%	11.25%	23.28%	20.35%	44.10%
4000	1.29%	2.20%	10.05%	21.16%	20.12%	52.39%

Accuracy. We evaluate the accuracy of our technique with increasing training size. For testing, we start with B samples and obtain an initial histogram. Then each new query refines the histograms. After the histogram is refined with the given workload, a test workload is used to estimate the overall error. As the training size increases, there is more feedback to learn the data distribution and so the accuracy of any workload-aware technique should improve with increase in training size. We see in Table 1 that this is true for `LS` but not for `STGrid`. In fact extra feedback often increases the error in case of 2 and 3-d range queries. Additionally queries which have higher cardinality amount to high relative estimation errors for the `STGrid` technique in the test workload. This

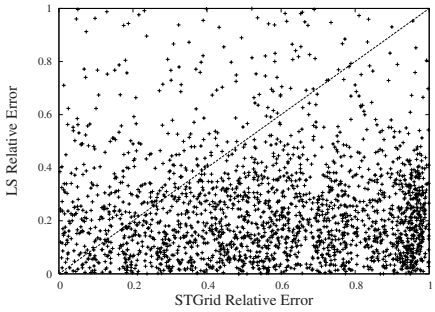

Fig. 2. Scatter Plot: Each point represents a query. Points below the diagonal line correspond to queries on which LS performs better than STGrid and vice-versa.

Model	Network Cost (GB)	Savings (GB)	% difference from optimal
No Caching	100.46		
Prescient	13.51	86.95	-0.00
LS	17.53	82.93	-4.02
STGrid	49.47	50.99	-35.96

Fig. 3. Impact of cardinality estimation on bypass-yield caching performance

is because `STGrid` looks at reducing the error of the current query but not the overall error. Thus it improves accuracy over a few queries but not all queries. This is clearly reflected in figure 2 in which we compare the error on a query-by-query basis. Figure 2 shows that for a few queries, STGrid has very low error, but as more queries come over the same region there is no guarantee that the error will be lower as the additional feedback produces inconsistent results.

Performance of BY caches. Our primary result (Figure 3) is to show the impact of an accurate workload-aware technique on the performance of BY caches. For this experiment we initialized a cache whose size varied from 1-5% of the database size. Our workload consists of range queries on several spatial and intensity attributes. Based on the yield of the incoming workload, the objective is to cache attributes such that network traffic is minimized. the workload consists of a mixture of single and multidimensional queries, each with equal probability of occurrence. In this experiment we compare the impact of the two approaches with a *prescient* estimator. A prescient estimator gives us an upper-bound on how well a caching algorithm could possibly perform when the cardinalities are known a-priori.

As a yield estimator for BY caching, `LS` outperforms `STGrid` dramatically and approaches the ideal performance of the prescient estimator. Even though LS has an accuracy range of 20-30% on the entire workload, for the caching system it performs close to the prescient estimator. This is because some of the queries on which `LS` performed poorly are sent to the server. The `STGrid` did not perform that poorly on those queries and thus severely affected cache performance. This experiment remarkably shows that a technique which improves the accuracy of a few queries can severely affect the performance of the BY cache.

The LS technique is computationally slower than STGrid. STGrid takes practically no time to update a histogram. In the worst case it has to refine the frequency value in B buckets. For the 1, 2, and 3 dimensional synthetic data sets LS takes an average of 5 secs, 65 and 115 secs over the entire training workload of 2000 queries. This is because with every feedback LS has to update an $O(B^2)$ matrix. More efficient approaches [14] for computing matrix transpose can be used. It is to be noted that the accuracy of LS comes at the expense of computational time. However, the considerable increase in accuracy is attractive, and the time to update is tolerable, making it a light-weight, accurate approach for remote applications.

6 Conclusions and Future Work

In this paper we have proposed a novel technique for learning workload-aware histograms that uses queries and their cardinality feedbacks only. The learned histograms are guaranteed to have the minimum sum of squared errors over the entire query sequence. Experiments show that minimizing the sum of squared errors provides far more accurate histograms than histograms learned by distributing error through heuristics. Further, our technique is neither tied to a query execution plan nor requires any artificial feedback. Therefore it is suitable for distributed applications that are remote from databases and need a light-weight, accurate self-learning cardinality estimation technique.

References

1. Olston, C., Widom, J.: Best-effort cache synchronization with source cooperation. In: ACM SIGMOD, pp. 73–84 (2002)
2. Malik, T., Burns, R.C., Chaudhary, A.: Bypass caching: Making scientific databases good network citizens. In: Intl' Conference on Data Engineering, pp. 94–105 (2005)
3. Ambite, J.L., Knoblock, C.A.: Flexible and scalable query planning in distributed and heterogeneous environments. In: Conference on Artificial Intelligence Planning Systems, pp. 3–10 (1998)
4. Poosala, V., Ioannidis, Y.E.: Selectivity estimation without the attribute value independence assumption. In: VLDB, 486–495 (1997)
5. Gibbons, P.B., Matias, Y., Poosala, V.: Fast incremental maintenance of approximate histograms. ACM Transactions on Database Systems 27, 261–298 (2002)
6. Malik, T., Burns, R., Chawla, N., Szalay, A.: Estimating query result sizes for proxy caching in scientific database federations. In: SuperComputing (2006)
7. Bruno, N., Chaudhuri, S., Gravano, L.: STHoles: A multidimensional workload-aware histogram. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (2001)
8. Srivastava, U., Haas, P.J., Markl, V., Kutsch, M., Tran, T.M.: Isomer: Consistent histogram construction using query feedback. In: 22nd International Conference on Data Engineering, p. 39. IEEE Computer Society, Los Alamitos (2006)
9. Aboulnaga, A., Chaudhuri, S.: Self-tuning histograms: Building histograms without looking at data. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 181–192 (1999)

10. Malik, T., Burns, R., Chawla, N.: A black-box approach to query cardinality estimation. In: Conference on Innovative Database System Research (2007)
11. Chen, C.M., Roussopoulos, N.: Adaptive selectivity estimation using query feedback. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 161–172 (1994)
12. Fujiwara, H., Iwama, K.: Average-case competitive analyses for ski-rental problems. In: Intl. Symposium on Algorithms and Computation (2002)
13. Young, P.: Recursive Estimation and Time Series Analysis. Springer, New York (1984)
14. Ari, M.: On transposing large$2^n \times 2^n$matrices. IEEE Trans. Computers 28, 72–75 (1979)