RHist: Adaptive Summarization over Continuous Data Streams*

Lin Qiao† Divy Agrawal Amr El Abbadi
Department of Computer Science
University of California, Santa Barbara
{lqiao, agrawal,amr}@cs.ucsb.edu

ABSTRACT

Many applications, such as network monitoring and telecommunication applications, depend critically on the efficient processing of data streams. Maintaining approximate aggregates and summaries over data streams is crucial to handle the OLAP query workload that arises in such applications. Furthermore, since the entire data is not available at all times the maintenance task must be done incrementally. We show that R(elaxed)Hist(ogram) is an appropriate summarization under data stream scenario. In order to reduce query estimation errors, we propose adaptive approaches which not only capture the data distribution, but also integrate independent query patterns. We introduce a workload decay model to efficiently capture global workload information and ensure that query patterns from the recent past are weighted more than queries that are further in the past. An advanced adaptive algorithm based on suitable threshold function is proposed for our workload decay model. The algorithm adapts the histogram to user query workloads for more accurate estimate of the range queries over data streams. We verify experimentally that our approach successfully adapts to changes in the workload as well as continuing changing data streams.

1. INTRODUCTION

Although traditional databases and associated operations are generally designed for static datasets, a large class of applications need database support in which the datasets consist of continuous data streams. For example, monitored information generated continuously from access points to which pocket PCs connect through a wireless network is streamed into analysis tools so that a system administrator is able to monitor and manage network flow among access points in real time. Similarly, for stock market applications, prices and exchange volumes of stocks keep changing with time. In order to analyze the market status, large amounts of stock information needs to be processed by stock exchange analysis software.

In most cases, large volumes of data would be stored and moved on secondary or tertiary storage. However, access to data on those media is quite expensive, and moreover, multiple passes are often not possible due to the large volume of data on such media. Also many applications need immediate answers from the data stream, thus preventing the use of secondary or tertiary storage to answer queries over a data stream. In the context of network management, it is important to track phenomena such as network congestion, network intrusion, or malicious attacks. In the case of stock markets, investors need online statistical analysis of stocks. Most of these applications often involve aggregate queries over data streams for trend analysis, rather than retrieving specific tuples with certain properties. In such applications fast approximate answers for aggregate queries are often more important than exact answers. This results in the need to maintain summary statistics to capture the data distribution of data streams instead of storing the entire data stream.

Continuous data streams are a new research topic that has recently received significant attention. In this paper, we consider the problem arising from applications of data streams that require fast approximate answers for aggregate queries by mainly using histograms as summary representation of data streams. Our histogram based approach maintains summary statistics by using information derived from both the data distribution and query patterns. We propose a dynamic summary representation over data stream, i.e., R(elaxed) Hist(ogram), which is constructed in one pass. RHist is maintained in memory and is used to answer aggregation queries directly. We present an integrated approach to adapt the histogram according to changes in the data distribution as well as changes in the query patterns. This approach successfully combines the refinement process of adapting to query-interest regions with the maintenance process of the data stream. Instead of simply considering the range of a single query as in prior work [5, 1, 3, 27], which cannot capture the distribution of a global query workload that evolves over time. We therefore introduce a workload decay model that represents a weighted summary of the global query workload. Based on the workload decay model, a multi-threshold function is proposed in order

*The work was supported in part by NSF awards IIS98-17432, EIA99-86057, EIA00-80134, and IIS02-09112.
†Supported in part by IBM Corporative Fellowship

CIKM'02, November 4–9, 2002, McLean, Virginia, USA.

to efficiently map workload information to the granularity of buckets of RHist. No prior research exists that combines summary maintenance with global query workload explicitly in the context of data streams.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 introduces the histogram model and discusses histogram maintenance issues in dynamic environments. Section 4 proposes an initially simple algorithm for adapting RHist to individual queries while maintaining the histogram based on the incoming a data stream. In Section 5, we develop the workload decay model, and present an advanced adaptive algorithm. We present an experimental evaluation of the proposed methods in Section 6 and conclude in Section 7.

2. RELATED WORK

Extensive research in the literature has studied static datasets, especially for query selectivity estimation. As our approach uses summary representation of a data stream and provides approximation answers, it is related to histograms [23, 15], sampling [27] and wavelets [19]. However, because these techniques work in static settings, they are not applicable to data streams.

As the need for dynamic datasets arises for an increasing number of applications, much research has been directed to incrementally maintaining summary representations. Babu et al. [2] presented an overview of the problem arising in this field. [11] proposed approaches for the dynamic maintenance of histograms. Incrementally maintainable quantiles have been proposed in [18, 13, ?]. Recent work done by Gilbert et al. [12] considered small space data summarization, i.e., wavelet, to provide approximate answers for aggregation queries over data streams from sketch. They used λ-aging data model to keep track of "recent" behavior of a data set. Gehrke et al. [9] explored the issue of computing correlated aggregates over data streams. Guha et al. [14] proposed a fixed window algorithm to process data streams and allow a tradeoff between accuracy and speed. Datar et al. [6] proposed stream statistics which can be maintained over sliding windows. Dobra et al.[?] proposed the framework to process general aggregate SQL queries over data streams by using randomizing techniques and domain partitioning. Unlike our approach, which adapts to both query workload and the data stream, these techniques maintain the histograms only based on the changes in the data stream.

As has been observed in the database community, precisely capturing data distributions may not be enough for accurately answering queries and hence several proposals exploit the query answer from a DBMS to adjust summary statistics. [5] introduced the concept of using query feedback from the query execution engine. ST-histograms [1] are refined by distributing estimation errors over the domain. STHoles [3] allows nested buckets to capture data regions and drills holes (buckets) selectively from the area where the query feedback intersects with STHoles. A sampling-based approximation [8] proposed weighted tuple sampling adaptive to query workload. Wu et al. [28] proposed adaptive sampling using golden estimation. All the techniques listed above, use query feedback to adjust the summary representation. Our work is motivated by these efforts, in the sense that the summary needs to capture the query patterns in addition to data distribution.

3. HISTOGRAM MODEL AND MAINTENANCE

3.1 Preliminaries

Histograms are widely used as a data summarization tool for selectivity estimation and approximate query processing. *Equidepth histograms* [22] are simple and form the basis of several popular and fairly robust histograms. In an equidepth histogram, all buckets represent the same number of data-items. *Compressed histograms* are a variant of equidepth histograms, which are more appropriate for data-sets with large skew. Consider an equidepth histogram where data values with high frequencies may span multiple buckets; this is wasteful since the sequence of spanned buckets for a data value can be replaced with a single count. A Compressed histogram has a set of such *singleton* buckets and an *equidepth* bucket over the remaining values. This approach is currently in use in IBM's DB2 6000 [23]. Consider a dataset of size N. A compressed histogram with β buckets has β' equidepth buckets and $\beta - \beta'$ singletons, where $1 \leq \beta' \leq \beta$. N' is the total number of tuples in equidepth buckets. For each bucket b_i, two values are stored: the largest value in the bucket $b_i.MaxVal$ and the number of items contained in a bucket $b_i.Count$. Thus b_i includes all values $\in [b_{i-1}.Maxval + 1, b_i.MaxVal]$. A compressed histogram satisfies the following requirements:

`REQ1` No single value "spans" [11] an equidepth bucket, i.e., the set of bucket boundaries are distinct;

`REQ2` Each equidepth bucket has N'/β' tuples, where N' is the total number of tuples in equidepth buckets;

`REQ3` Each singleton bucket has depth $\geq N'/\beta'$;

3.2 Histogram Maintenance and Ancillary Storage

Gibbons et al. [11] proposed a fast incremental maintenance algorithm for approximate histograms. We briefly describe the maintenance process as follows. After initializing the histogram, we derive a threshold $T = (2+\gamma)N'/\beta'$, where γ is a threshold control parameter [11]. The buckets of the histogram are either approximate equidepth buckets or singletons. When a new data item comes, the depth of the corresponding bucket increases by one. If the depth of this bucket exceeds T, the bucket is divided into two and two other buckets are merged in order to keep the number of buckets constant. T changes according to incoming data items and due to changes in the number of singleton buckets. Furthermore, a singleton may no longer be a singleton when T increases, while a new singleton bucket is generated due to a large number of incoming data items with a particular value. This process guarantees that value points with high frequencies are captured by singletons and other values are grouped into approximate equidepth buckets.

As is shown in [11], additional information, i.e. approximate median, is needed to divide a bucket into two. There are other summaries, for example wavelet with sketches [12], which can be incrementally maintained without accessing ancillary storage. However, in the following section, we will show that ancillary storage is indispensable to adapt summarization to query workloads. For many applications, such as telephone system and stock market analysis tools, records of all the data are maintained. Even if this base information is not available, we can maintain *backing information* on disk incrementally. Gibbons et al. [11] have exploited reservoir sampling [26] to generate backing samples and store them on disk in order to compute the median for dividing a bucket.

Other types of backing summarizations can also be used for backing samples. We maintain a backing frequency distribution function(fdf) on disk which summarizes the data streams into distinct values with frequencies. Given data stream DS with attribute X, the *frequency* f_i of value d_i for attribute X is the number of tuples $t \in DS$ with $t.X = d_i$. The backing fdf is an ordered set of value pairs $\{\langle d_0, f_0\rangle, \langle d_1, f_1\rangle, \langle d_2, f_2\rangle, ..., \langle d_{N-1}, f_{N-1}\rangle\}$. There are several techniques that can be applied to lower the overheads of the algorithm with regard to disk accesses, such as *lazy updates*.

3.3 RHist: A Relaxed Histogram

In this section, we define the histogram studied in this paper and evaluate its performance briefly.

Our histogram approximates the exact compressed histogram by modifying the three requirements as follows.

`REQ1'` No single value belongs to more than one bucket.

`REQ2'` The depth of equidepth bucket does not exceed the threshold $T = (2 + \gamma) \times N'/\beta'$ [11];

`REQ3'` The same as `REQ3`.

`REQ1`' strengthens `REQ1`. `REQ1`' not only guarantees that singleton can be captured in RHist, but also requires to reallocate the boundaries of equidepth buckets to include or exclude data items with the same values. Due to `REQ1'`, requirement `REQ2` is *relaxed*. Buckets can have approximately equal depth within the threshold T. We refer the proposed histogram *RHist*, since it relaxes the restriction that all equidepth buckets have the same count.

For performance evaluation, we use the *Normalized Relative Error* (NRE) to measure the accuracy of query results compared with the exact answer: $NRE = \frac{|\mathcal{A}_{est}(\mathcal{Q}) - \mathcal{A}_{ext}(\mathcal{Q})|}{\mathcal{A}_{ext}(\mathcal{Q})}$, where $\mathcal{A}_{est}(\mathcal{Q})$ is the estimated answer provided by the histogram and $\mathcal{A}_{ext}(\mathcal{Q})$ is the exact answer.

MaxDiff(v,f) [25] and *V-Optimal(v,f)* [15] are two histograms known for their good performance in summarizing static datasets. Figure 1 depicts the difference among histograms in approximating skewed data in terms of normalized relative error of queries with selectivity between 10% and 20%. The centers of queries are generated by a Gaussian distribution. The dataset is Zipf with different skewness and the correlation between values and frequencies is random. Figure 1 shows that the performance of RHist is not as good as MaxDiff and V-Optimal histograms. But RHist performs much better than the equidepth histogram. Since it successfully captures high frequencies. In the following sections, we are going to reduce the estimation error to lower than that of MaxDiff and V-Optimal for data streams.

4. THE BASIC ADAPTIVE ALGORITHM

In general, the aim of the adaptive approach is to integrate the information contained in queries into the data summarization so that it can provide more accurate answers for a particular *query-interest region*. The information associated with queries can be the exact answer of a query [5, 3, 27, 8],

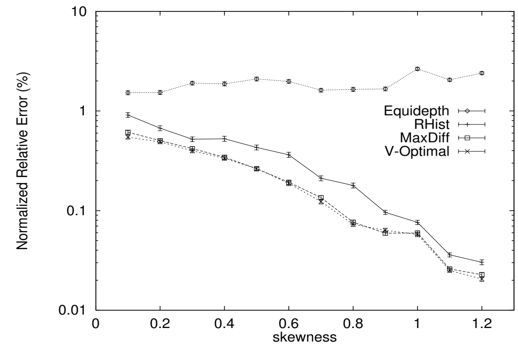


Figure 1: The performance of static histograms

the error in the approximate answer from data summarizations [1], or just a query range. In the database scenario, the exact answer for a query is provided by the DBMS. Thus the former two kinds of information are available, even for free, to the adaptive process. More accurate answers can be provided by refining the query-interest region, especially exploiting the exact query answers. In the data stream scenario, data summarization is the only available source for answering queries. Although the complete dataset may be stored on disk, it is not realistic to be frequently accessed due to its large volume. Hence we cannot exploit traditional approaches to refine RHist. In order to achieve the goal of adaptive maintenance, we need to refine the granularity of buckets within the query-interest region, and make coarser the granularity of buckets out of this region.

4.1 Motivation

To adapt RHist to varying query patterns, our basic idea is to ensure that there are more buckets in the ranges where more queries are expected to occur. The inaccuracies of queries based on RHist (or any other histogram) arise because the boundaries of a query may not completely align with the boundaries of the buckets. In such cases the query result is extrapolated usually by making the *uniform spread assumption* [25] of data distribution within each bucket. More buckets in the range of interest result in narrower buckets and hence reduces the error during estimation [22]. Our approach therefore relies on dividing some buckets corresponding to the query range. However, in order to maintain the overall storage requirements of RHist constant, creating additional buckets must be offset by reducing some buckets that are outside the range. We accomplish this by merging two consecutive buckets, outside of the query range, into one. Although wider buckets will probably result in higher estimation errors, we expect the impact to be small since these buckets are outside the range of query interests.

We establish that if the data in each bucket is monotone (increasing or decreasing), then dividing a bucket in two will always reduce the error for a query that is delimited by this bucket. The metric we use is the normalized relative error. Lemma 4.1 and Theorem 4.2 show the reduction in error for queries within a bucket and the proofs are presented in the appendix.

LEMMA 4.1. *Let b_i be a bucket with monotone frequency distribution in histogram $\mathcal{H}$. b_i is divided to b_i' and b_i''. Let Q be a query with range $(a, c]$ where $a = b_{i-1}.MaxVal$*

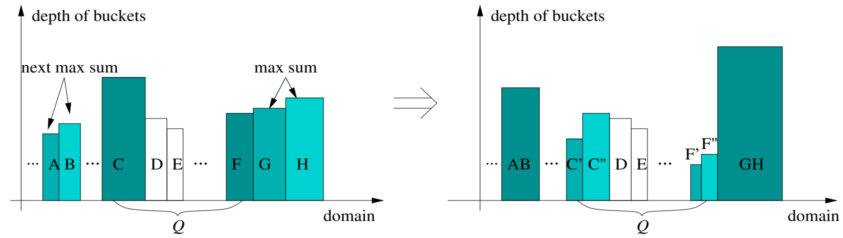


Figure 2: Refine RHist

and $c \leq b_i.MaxVal$. If NRE and NRE' are the respective normalized relative errors before and after division, then $NRE' \leq NRE$.

Based on Lemma 4.1, we generalize the result to a query with arbitrary range.

THEOREM 4.2. *In histogram $\mathcal{H}$ dividing a set of buckets $\mathcal{B} = \{b_i | b_i$ has monotone frequency distribution$\}$ reduces the error introduced by the uniform spread assumption.*

Clearly, the above results establish that by refining the number of buckets in a given query range results in more accurate answers for monotone frequency distribution. Motivated by the above analysis, we develop our basic adaptive algorithm to adjust RHist due to changing query patterns.

4.2 The Basic Approach

Our basic approach is based on adapting RHist for individual queries. The only information a query provides is the query range. Given a query $Q = [a, b]$, there are several ways to use the range of a query to refine the histogram.

A simple method is to divide all the buckets $b_i \in \mathcal{B}$, where $\mathcal{B}$ is the set of buckets covered by the range of Q. Let n be the number of buckets in $\mathcal{B}$ and hence after the division, the depth of those n buckets intersected with Q will decrease. In order to keep the total number of buckets in RHist unchanged, before those n buckets are divided, we need to identify $2n$ buckets which do not intersect with Q that need to be merged. Clearly, if $n > \beta/3$, there will not be enough buckets to merge. Furthermore, if the query load focuses on a specific range, the overhead for refining the histogram will double for each incoming query. In order to overcome these limitations, we propose an alternative method, in which at most the two buckets containing the boundaries of Q, i.e., a and b, are considered as candidates for division. For example, buckets C and F in Figure 2. However, before the buckets can be merged we need to identify the necessary number of buckets (i.e., 2 or 4) outside $[a, b]$ which can be merged in a pairwise manner. Buckets A and B as well as buckets G and H in Figure 2 result in merged buckets AB and GH. Since the number of divided buckets is bounded by two, constant time is needed to refine a histogram as a result of a query.

We consider the maintenance process together with the refinement process, because incoming data and queries both modify RHist in the data stream scenario. The algorithm for bucket refinement is shown in Figure 3.

Before a bucket is divided, several conditions need to be checked. First, if bucket b_i is a singleton, i.e., there is only one attribute value in bucket b_i, bucket b_i does not need to be divided. Second, in order to keep the total number of buckets unchanged, two other buckets should be merged.

```
Refine(bkt[i])
/* 𝓕 is the backing frequency distribution function */
if(!need_divide(bkt[i])) return;  /*singleton or Section 4.3*/
for all j ∈ [1, n-1] /*n is # of buckets*/
   if (bkt[j].Count+bkt[j+1].Count)<T)
      find Max(bkt[j].Count+bkt[j+1].Count);
if(find j) Merge(bkt[j], bkt[j+1]);
else return;
if bkt[j] or bkt[j+1] is singleton
   update N', β';
total = bkt[i].Count;
k = Find_median(bkt[i]);
sum = 𝓕.freq[bkt[j-1].MaxVal+1]+...+𝓕.freq[k];
if((sum-total/2)>(total/2-(sum-𝓕.freq[i])
   k−−; /*Find the quantile closest to median*/
divide bkt[i] at k into bkt[i'] and bkt[i''];
if bkt[i'] or bkt[i''] is singleton
   update N', β';
update N, T;
```

Figure 3: Basic Bucket Refinement Algorithm

If there are no such two buckets available, the process to divide b_i terminates.

A bucket should be divided when its depth exceeds the threshold T or it intersects either boundary of the query. To keep the depth of the two new buckets balanced, we divide the bucket at the median (50% quantile). Recall that in RHist all data items with the same value should be in one bucket. Using the median to divide a bucket may violate this constraint of RHist. In order to solve this problem, we reallocate the boundary of the first new bucket to the value which is the quantile closest to the median using the backing information in ancillary storage.

As discussed above, before dividing a bucket, we should first determine the buckets to merge. If the merge is due to an update which causes bucket b_i to exceed the threshold T, then two buckets in RHist excluding b_i should be merged, i.e., the set of buckets that are candidates for merging are $\mathcal{B} = \{bkt_j |\ bkt_j \neq bkt_i\}$. If we need to merge buckets due to query Q, the set of buckets $\mathcal{B} = \{bkt_i |\ bkt_i.MaxVal \subset [Q.left, Q.right]\ \vee\ bkt_{i-1}.\text{MaxVal+1} \subset [Q.left, Q.right]\ \}$ will not be considered for merging since the buckets that fall in the range of a query should not be merged. Let $\bar{\mathcal{B}} = \{bkt_i |\ bkt_i \notin \mathcal{B}\}$. Candidates to be merged are generated from $\tilde{\mathcal{B}} = \{bkt_i |\ bkt_i.Count{+}b_{i+1}.Count < T\ with\ bkt_i, bkt_{i+1} \in \bar{\mathcal{B}}\}$. We do not want to merge buckets that were just divided due to recent queries and those buckets usually have low frequencies. The algorithm pick two consecutive buckets $\{(bkt_i, bkt_{i+1}) |\ \exists bkt_i,\ bkt_{i+1} \in \tilde{\mathcal{B}}\ , \forall bkt_j,\ bkt_{j+1} \in \tilde{\mathcal{B}}, bkt_j.Count + bkt_{j+1}.Count \leq bkt_i.Count + bkt_{i+1}.Count\}$, i.e., the two buckets resulting in the largest summation are merged. Note that recently divided buckets will usually have a small total number of tuples and hence are less likely to be candidates for merging.

Singletons can participate in merging. Hence, after merging, the number of singletons may decrease, as well as the total number of items within singletons. On the other hand, when a bucket is divided into two buckets, new singletons may be generated. In both cases, we need to update N', β' and recompute threshold T.

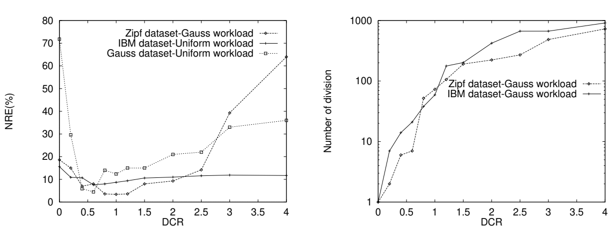
4.3 Controlling over-division

The basic approach adapts RHist by tuning the granularity to be finer for buckets covering the boundaries of queries and coarser for the buckets out of query ranges. This process is performed for every incoming query. However the benefit of continually dividing buckets within a particular region due to a stream of queries will not be significant since the granularity in that area already results in a small relative error. Significant inaccuracy may arise if the query interest region moves out of the refined region. Under these circumstances, the benefits of dividing such fine-grained buckets will be outweighed by the reduced accuracy resulting from the merging of other coarse-grained buckets. We call this case *over-division*. Furthermore, we cannot afford to access the backing information for each query in order to refine buckets. Due to these concerns, we introduce a parameter, referred to as *division control ratio* (DCR) to control the degree of division. In the basic adaptive algorithm, when a bucket b_i is a candidate for division, we introduce the following check to control the granularity of the division. If

$$b_i.MaxVal - b_{i-1}.MaxVal \le \frac{\#\ of\ distinct\ values}{bucket\_number} \times \frac{1}{DCR} \quad (1)$$

then b_i does not need further division. From this constraint, we can see that, if DCR is large, then buckets with small range will not be divided. For example, let the average number of distinct values in a bucket be 20. If DCR is set to 2, then buckets with range smaller than 10 will not be divided. If DCR is set to 0.5, then buckets with range smaller than 40 will not be divided.

Figures 4 depicts the effect of DCR on estimation error and the number of accesses to backing information. Note that the number of divisions is the same as the number of accesses.



(a) Estimate error (b) Number of division

Figure 4: Effect of DCR

The IBM dataset is a set of daily prices of IBM stock starting from 1962 to 2000 with 9567 records. For the *Zipf dataset-Gauss workload* in Figure 4.(a), as DCR is increased up to around 1, the normalized relative error decreases as the number of buckets being divided decreases and thus prevents the case of over-division efficiently. Note that beyond the DCR value of 1, NRE increases because not enough buckets have been divided so that future queries will not benefit from previous queries. The other two dataset-workloads show similar effect of DCR. It is obvious that DCR is an effective parameter to control the percentage of buckets being divided and thus prevents over-division efficiently. However, the value of DCR varies for different data distributions and query workloads. From our experiments a value between 0.5 and 1.2 is reasonable. Figure 4.(b) shows how DCR controls the number of division operations over the whole workload. The number of divisions during maintenance is not included, because DCR only affects the refinement process. If DCR is between 0.5 and 1, the number of divisions is quite low.

5. ADVANCED ADAPTIVE ALGORITHM WITH WORKLOAD DECAY MODEL

In the previous section, we developed a basic algorithm to adapt the granularity of RHist to individual queries. However, there are several problems which cannot be solved using the basic adaptive algorithm. First of all, using a single threshold for all the buckets of the histogram maintains the depth of the buckets approximately equal. This blocks the refinement of the histogram thus preventing different granularity in different query-interest regions. Also the affect of the division control parameter depends largely on the data distribution and thus we cannot set it to an appropriate value in advance. Furthermore, the range of a single query does not represent the interests of most queries. Lacking knowledge of the global workload information, the basic adaptive algorithm is not able to determine the difference between individual interest and group interest. In this section, in order to solve these problems, we introduce a generalized workload model and exploit it in our advanced adaptive algorithm to automatically control the granularity of the histogram buckets according to current workload information.

5.1 Workload Decay Model

The purpose of the adaptive approach is to change the data summaries to reflect the query-interest region and hence reduce the estimation error. Most data summaries are built by assuming that queries will be issued *uniformly* over the dataset and most adaptive methods are developed for the areas of interest to the *individual* queries. However they may not be able to reflect the interests of a group of queries. A global view of the workload should contain a *history* of the queries. However, older queries have less affect on the desired area of interest, while recent queries should have greater influence. In other words, the affect of a query should decay with time. In order to capture this kind of global behavior of the workload, we introduce a novel model, the *Workload Decay Model*.

Our construction of the workload decay model is described as follows. W is a vector which keeps the current workload information. Q_j is a vector of the query information at time j $(-t < j < 0)$. Q_0 is current query vector and Q_{-t} is the initial query vector.

$$W = \lambda\cdot(1-\lambda)^{t-1}Q_{-t} + \lambda\cdot(1-\lambda)^{t-2}Q_{-t+1} + \cdots + \lambda\cdot Q_0 \quad (2)$$

The invariant for the workload decay model is

$$\sum_{i=min}^{max} W[i] = 1 \quad (3)$$

$W[i]$ reflects the query workload on value i. For example, if the incoming queries uniformly cover the whole domain and the number of distinct values is 1000, $W[i] = 0.001$ for all $i \in domain$.

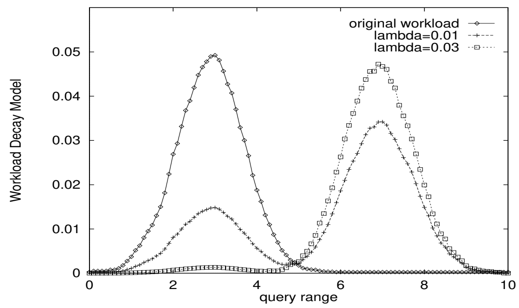


Figure 5: Workload Migration

Parameter λ controls the migration behavior of workload information W. Take two extremes of λ as an example: When $\lambda = 0$, W keeps the past instead of integrating the new query into the workload information. When $\lambda = 1$, W only contains the information of current query and no history is accessible, and hence it reduces to the basic adaptive algorithm. From Figure 5, we can see the affect of λ on W. The original workload consists of queries generated from a Gaussian distribution with mean 3. Then the center point of queries was shifted to Gaussian with mean 7. The number of distinct values is 100. With $\lambda = 0.03$, the workload curve quickly forgets the past and reflect the new trend very well. While with $\lambda = 0.01$, the workload curve still keeps decayed memory of the past, and thus not only reflects the new trend, but also remembers the history quite well.

We maintain an workload histogram, referred to as WHist, with non-integer depth $\in [0, 1]$ to capture the changes in the query workload. Initially, we have no query workload information, so we assume it is uniformly distributed. All the buckets are initialized to the depth $1/n$, where n is the number of buckets. If the query workload is uniformly distributed, each distinct value should have the same query interest weight and due to Equation 3, i.e.$\sum_{i=1}^{n} b_i.count = 1$, the depth of each bucket should be 1/n. An incoming query vector Q has $Q[i] = 0$ for the value i out of query range and $Q[i] = 1$ for the value i within query range. The depth of each bucket is decreased by multiplying the factor $1 - \lambda$. Then the depth of bucket b_i, which is covered by Q, is increased by $\lambda \times (Q.range \cap b_i.range) \times \frac{1}{Q.range}$.

```
WHist(n:  # of buckets)
   for all the bucket bi.count = 1/n;
   Decay(Q.left:the left boundary of current query,
         Q.right:the right boundary of current query)
   {  increment = λ/(Q.right-Q.left+1);
     pos = Q.left;
     for all the bucket bi
        bi.count *= 1-λ;
        while((pos∈ bi.range)&(pos∈[Q.left, Q.right]))
           pos ++;
           bi.count += increment;
   }
```

Figure 6: Incremental workload histogram

5.2 Threshold Function

As discussed in Section 4, the threshold plays a critical role in controlling the depth of buckets in histogram. Because there is only one threshold for the whole histogram, the affect of the threshold is equal for each bucket, namely, it tries to equalize the depth of buckets within a certain flexible range. However, when the histogram begins to adapt to incoming queries, the granularity of each bucket tends to be different due to the query-interest region. Buckets falling in the query range will have finer granularity and thus tend to be have less height than buckets out of the query range. So the shape of RHist changes according to variations in query ranges. It is obvious that a single threshold restricts RHist's ability to adapt to the query workload. It is in fact detrimental to the adaptive process. In this section, we propose a multi-threshold solution by introducing a threshold function. Using this function, we are able to compute the threshold for a single value or a bucket from the workload information and data information. Virtually, we can imagine that for each value there is a threshold and later we introduce the threshold function for a set of values within a bucket.

Because we use a threshold to control the granularity of buckets in different query-interest regions, the threshold function should reflect the shape of the histogram during the adaptive process, i.e. the buckets in the heavily queried region have lower depths while buckets in sparsely queried region have higher depths. The threshold function is based on a query workload function and a data function, that is, $Threshold(v) = f(W(v)) \times g(D(v))$. Intuitively, $f(W(v))$ is inversely related to $W(v)$ and $g(D(v))$ is proportional to the domain range, where $g(D(v)) = (2+\gamma)N'/\beta'$. Thus the more queries focus on v, the lower the threshold and the more data, the higher the threshold.

Now the key to find the threshold function is to derive $f(W(v))$. Let

$$f(W(v)) = \frac{c'}{W(v) + c} \tag{4}$$

We add parameter c to the denominator so that when $W(v) = 0$, $f(W(v))$ is still meaningful and parameter c' is a multiplier. Our goal is to determine the values of c and c'. From Lemma C.1 (see appendix), if the query workload is uniformly distributed, then $W(v) = \frac{1}{d}$, where d is the number of distinct values. In this case, we do not need to refine any bucket in RHist, i.e. $f(W(v)) = 1$. Thus we have

$$\frac{c'}{1/d + c} = 1 \tag{5}$$

Given d, we need another equation to solve c' and c.

By observation, the probability that $W[j]$ has large value, especially close to 1, is fairly small. And the probability that $W[j]$ is between zero and $1/d$ is quite large, i.e., approximately equal to 50%, because $1/d$ is the unbiased estimator for $W[j]$ at any value point j by lemma D.1(see appendix). Normal distribution $N(\mu = 0, \sigma^2)$ can capture this feature on the positive horizontal axis. In order to check the assumption that $W[j]$ at any j is normally distributed, we conduct a *normal probability plot* [21]. The plotted points fall along a straight line, which confirms that the hypothesis model is appropriate. The 75 percentile of $N(0, \sigma^2)$ is very close to the mean of the right half when σ is very small. Later we will show that σ is actually very small. Thus we approximate σ from the following equation.

$$\int_0^{1/d} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}\, dx = 0.25 \qquad (6)$$

Let random variable T stand for $f(W(v))$ at point v. Let X be the random variable for $W(v)$. By equation 4, $T = \frac{c'}{X+c}$, $X \in [0,1]$ and $T \in [\frac{c'}{1+c}, \frac{c'}{c}]$. Let $Z \sim N(0,\sigma^2)$. Because X has the positive half of the $N(0,\sigma^2)$ distribution, we have $f_X(v) = 2f_Z(v)$ for $v \in [0,1]$. By the *random variable transform theory* [4],

$$\begin{aligned} f_T(T=t) &= f_X(X = \frac{c'}{t} - c) \times \left|\frac{\partial x}{\partial t}\right| \\ &= 2f_Z(\frac{c'}{t} - c) \times \frac{c'}{t^2} \\ &= \frac{2}{\sqrt{2\pi}\sigma} e^{-(\frac{c'}{t}-c)^2/2\sigma^2} \times \frac{c'}{t^2} \end{aligned}$$

Because T is a random variable for $f(W(v))$, we want the values of T to be centered around 1 so that RHist is balanced, i.e., no bias.

$$\begin{aligned} E(T) &= \int_{\frac{c'}{1+c}}^{\frac{c'}{c}} f_T(t)t\, dt \\ &= \int_{\frac{c'}{1+c}}^{\frac{c'}{c}} \sqrt{\frac{2}{\pi}} \frac{1}{\sigma} e^{-(\frac{c'}{t}-c)^2/2\sigma^2} \frac{c'}{t}\, dt \\ &= 1 \end{aligned} \qquad (7)$$

Now we are able to solve for c' and c. From equation 6, we obtain σ for different values of d. Given d and σ, equations 5 and 7 can provide solutions for c' and c. The following table shows the respective solutions for different d.

d	σ	c	c'
10	0.148	0.263	0.363
100	0.0145	0.031	0.041
1000	0.0015	0.0024	0.0034

Hence, given d, the threshold is computed by

$$Threshold(v) = \frac{c'}{W(v)+c} \times (2+\lambda)N'/\beta'$$

5.3 The Advanced Adaptive Algorithm

The workload decay model is able to capture the interest of a group of queries. The threshold function successfully maps information in the workload decay model and data information into controlling parameters for different values in the domain. Now we present the complete algorithm by integrating all the techniques developed in this section with the basic adaptive algorithm.

The granularity of data in RHist is buckets. We cannot use the threshold function directly on RHist, since it is a mapping from a particular value to a threshold. In RHist, the threshold controls the depth of a bucket. Hence a new threshold function for each bucket is needed. For a bucket b_i in RHist, we have the boundary $b_i.left$ and $b_i.right$, where $b_i.left = RHist.bkt[i-1].MaxVal + 1$ and $b_i.right = RHist.bkt[i].MaxVal$. Since each value $v \in [b_i.left, b_i.right]$ has its own $W(v)$ in WHist, the average of $W(v)$ represents the average query interests in $[b_i.left, b_i.right]$. Thus the threshold value for a bucket is function based on the average of $W(v)$ for values within this bucket. The formula of the threshold function becomes

$$Threshold(b_i) = \frac{c'}{Avg_{v\in[b_i.left,b_i.right]}(W(v)) + c} \times (2+\gamma)N'/\beta' \qquad (8)$$

In WHist, the summation of $W(v)$ for all $v \in [b_i.left, b_i.right]$ can be computed by issuing a query on WHist and the query range is $[b_i.left, b_i.right]$. The average of $W(v)$ is the summation divided by the number of values within this bucket. We now have two summarizations, i.e. WHist for the workload decay model and RHist for the data. Updates on either one change the threshold, because the threshold function is related to both workload and data. Thus when a new query is issued or a data item arrives, we need to check the threshold and see if the corresponding RHist bucket depth exceeds the threshold. If a particular RHist bucket exceeds the threshold, it needs to be refined. The process of refining a bucket is basically the same as the basic adaptive algorithm except that each bucket has its own threshold in the advanced adaptive algorithm. The advanced adaptive algorithm does not use DCR which is essential in the basic adaptive algorithm. This is because the refinement is controlled by the workload decay model. The algorithm to check the threshold is presented in Figure 7.

```
Check_threshold(WHist, RHist)
   //triggered by incoming data
   if RHist.bkt[i].isupdated()
      if RHist.bkt[i].count>threshold(RHist.bkt[i])
         Refine(RHist.bkt[i]);
   //triggered by incoming queries
   if WHist.bkt[j].isincreased()
      for all RHist.bkt[i] ∩ WHist.bkt[j] ≠ ∅
         if RHist.bkt[i].Count>threshold(RHist.bkt[i])
           Refine(RHist.bkt[i]);
```

Figure 7: Threshold Checking Algorithm

6. EXPERIMENTAL RESULTS

In this section, we describe a set of experiments that evaluate the effectiveness and efficiency of our workload-aware histogram maintenance techniques. First we describe the experimental setup. Then we describe a set of experiments to demonstrate the efficiency of using the two adaptive RHist algorithms. After that, we study the performance of our adaptive algorithms with mixed query workloads. Finally, we evaluate their performance with mixed data updates and query workloads.

6.1 Experimental Setup

Data streams The incoming tuples of a data stream are generated from an extensive set of both *synthetic* and *real* datasets. The real dataset we consider is the Alexandria Digital Library(ADL) dataset, which includes relatively large datasets derived from a spatial domain. The dataset contains 2,339,771 digital spatial objects. We project the centers of the spatial objects on the longitude to derive the ADL dataset, as shown in Figure 8.

We also generated synthetic datasets for our experiments from different data distributions, including *Zipf-ian* [30], *Gaussian* and *Uniform* distribution. In the Zipf distribution,

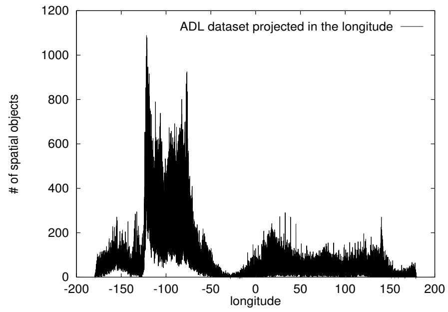


Figure 8: ADL projection in the longitude

the values and frequencies are randomly correlated (uncorrelated) and z varies from 0.1 to 1 for different data skewness. The Gaussian distribution is generated from $domain \times Gauss(\mu, \sigma^2)$. The values and frequencies are also randomly correlated. In the uniform distribution, the number of tuples in a data stream varies from 10,000 to 1,000,000. The default value for the synthetic dataset parameters are described as below.

Dataset	Parameter	Value
All	R: # of distinct values	1,000
	N: Length of data streams	1,000,000
	DCR: divide control ratio	1
Gauss	μ: mean	0.5
	σ: standard deviation	0.005
Zipf	z: Skewness	0.4

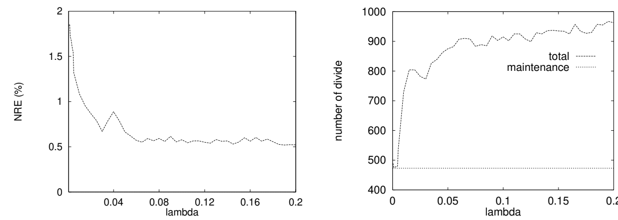
In our current experimental framework, we restrict data streams to those with tuple insertions, which is generally the case with most data streams.

Histograms We studied both *static* and *dynamic* histograms. For static histograms, e.g., MaxDiff, V-Optimal and EquiDepth histograms, we rebuild them for every incoming data item, thus giving them extra benefit, which in practice is unrealistic, while we maintain and refine RHist incrementally. The number of tuples from the data stream to initialize RHist varies from 1,000 to 10,000. The default number of buckets of a histogram is set to 100. Threshold T, which controls the depth of buckets in RHist, equals $N'/\beta' \times (2+\gamma)$, where $\gamma = 0.5$, as [11] has claimed that this is a reasonable value for limiting the number of computations, i.e., the number of access to disk, as well as for decreasing errors. We use RHist, BRHist and ARHist as the abbreviations for non-adaptive RHist, RHist with the basic adaptive approach and RHist with the advanced adaptive approach respectively. In the following experiments, we allocate the same memory space for all types of histograms studied. Since the equidepth histogram only stores the bucket boundary, the number of buckets are twice as much as RHist, MaxDiff and V-Optimal. For ARHist, we count the additional buckets WHist consumes for the workload decay model.

Query workload We used 3 kinds of query workloads to generate a range query: 1. *Gaussian*, the center of a query is generated by $domain \times Gauss(\mu, \sigma^2)$; 2. $Uniform$, the center of a query is calculated from a uniform distribution over the whole domain; 3. *Mixed workload*, in which queries are created by mixed distribution from Gauss and uniform, rather than a single distribution. To simplify the representation of the query workload, the workload with Gaussian distribution is represented as $Gauss(\mu)$, short for $domain \times Gauss(\mu, 0.005)$. The default selectivity is 10%.

Metrics To measure the accuracy of answers to aggregation queries, we used two error metrics: *Normalized Relative Error* (NRE) and *Confidence Interval for Mean of Error* (CIME). As introduce in Section 3.3, $NRE = \frac{|\mathcal{A}_{est}(\mathcal{Q}) - \mathcal{A}_{ext}(\mathcal{Q})|}{\mathcal{A}_{ext}(\mathcal{Q})}$; CIME represents the interval where the mean of the error will vary with 90% confidence.

6.2 Effect of λ



(a) Estimate error (b) Number of division

Figure 9: Effect of *lambda*

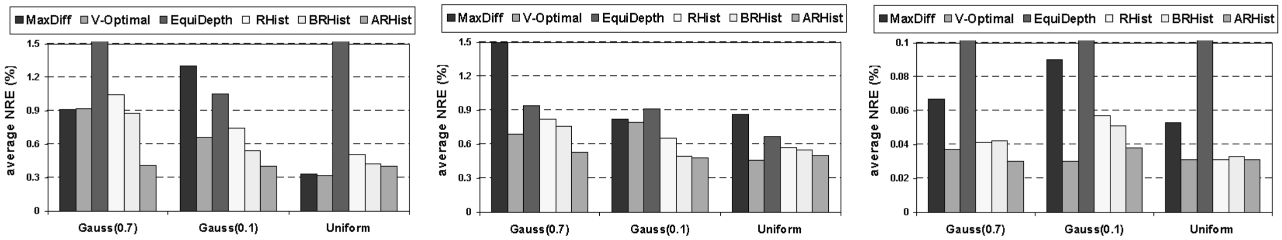
In order to find a reasonable value for λ for ARHist, we conducted this experiment to observe the performance and disk access when λ changes. Unlike DCR, λ controls the number of bucket divisions in both maintenance and refinement. Hence, we measure the number of divisions due to both processes. We use a *Training workload* to set up the workload decay model, instead of using the default setting, i.e. uniform distribution. Hence, we are able to observe how λ controls the granularity of buckets under the incoming data stream. The training workload is $Gauss(0.7)$, then we generate a Zipf dataset with z=0.4. The query workload is $Gauss(0.3)$ and size is 1000.

Figure 9.(a) depicts the dramatic improvement in performance when λ increases from 0 to 0.06. Further increasing the value of λ does not help to reduce the estimation error. In Figure 9.(b), a rapid increase in the number of divisions is observed when λ is varied from 0 to 0.7. As depicted by the lower dotted plot, the number of divisions during maintenance is a constant. In conclusion, the choosing of λ represents a tradeoff between disk accesses and performance, and 0.01 is reasonable for both concerns.

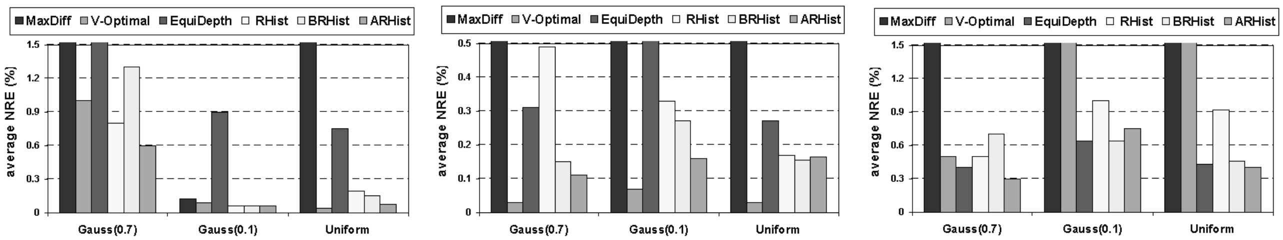
6.3 Accuracy of histograms

We now compare the effectiveness of various techniques in approximating histograms under different data streams and workloads conditions. In this experiment, we subject the histogram, first to a data stream, and then to a series of queries. At the end of the data stream, we build the static histograms, i.e., MaxDiff, V-Optimal and Equi-depth. RHist is incrementally maintained over the data stream. BRHist and ARHist incorporate the incoming stream and then adapt to the query workload.

We implemented 3 workloads. The first two are $Gauss(0.7)$ and $Gauss(0.1)$. The third is a uniform query workload. We impose queries on 6 different datasets. In addition to uncorrelated Zipf and Gauss, the tested datasets also include positively correlated Zipf(Figure 11.(d)) and Gauss (Figure 11.(e)). Figure 10(a-f) shows that the adaptive RHist approaches, in general, greatly improve the performance compared to non-adaptive RHist. In Figure 10.(a)(b)(f)

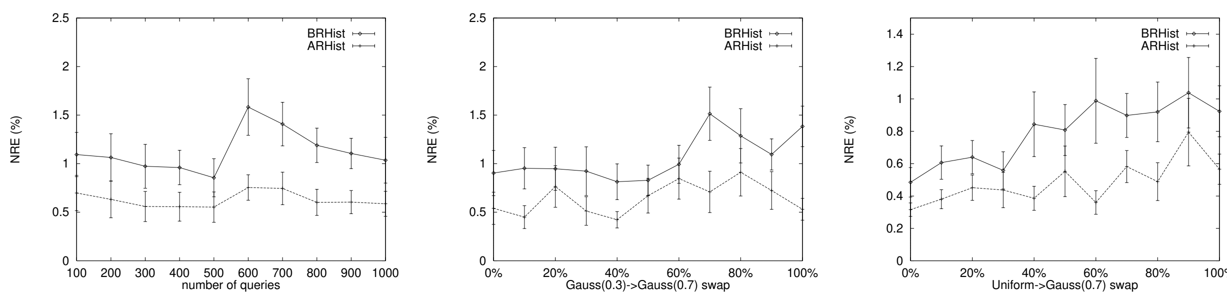


(a) Zif(0.4) (b) Gaussian dataset (c) Uniform dataset



(d) correlated Zif(0.4) dataset (e) correlated Gaussian dataset (f) ADL dataset

Figure 10: The performance of histograms for different dataset and workload



(a) abruptly shifting (b) gradually changing (c) gradually changing

Figure 11: Adapt to changing workload

ARHist is even better than MaxDiff and V-Optimal histograms, which are built from the whole dataset. The results establish that the workload decay model successfully captures the query distribution and that the multi-threshold function effectively maps workload information to thresholds, which lead to an efficient adaptive algorithm. For non-uniformly distributed workloads, ARHist is always better than BRHist and much better than RHist. For uniformly distributed workloads, the threshold at every data value is $(2+\gamma)N'/\beta'$ for ARHist, which is the same as in BRHist or RHist. Thus the performance of those histograms are identical due to the same granularity distributions. Although RHist allocates some buckets with equal depth, it outperforms equidepth histograms because singleton buckets are able to capture high frequencies separately. Note that in Figure 10.(d), for Gaussian(0.7), the NRE of BRHist is quite higher than RHist. This is because the effect of DCR depends on the dataset and workload distributions.In conclusion, our adaptive algorithms for incrementally maintained RHist, especially ARHist, compete with static histograms which are well-known for their good estimation performance, but would incur unrealistic overheads in a data stream setting.

6.4 Mixed workload

Queries may not always be generated from the same distribution. It is very common that the query workload evolves. In order to test the robustness of our adaptive approaches, we used mixed workloads, which are processed after the stream of data. We mix the workload by either abruptly or gradually changing from one workload to another.

In Figure 11.(a) we shift the workload abruptly from Gauss(0.3) to Gauss(0.7) after 500 queries. The NRE keeps decreasing in the same workload as later queries benefit from dividing and merging buckets due to earlier queries. When the query workload shifts suddenly to another region, the NRE increases accordingly. ARHist has very small variation compared to BRHist, which benefits from the effectiveness of the workload decay model and hence prevents unnecessary over-divisions. Figure 11.(b) and Figure 11.(c) show the gradual transformation between two different Gaussian distributions, and from a uniform distribution to a Gaussian distribution respectively. In both cases, the x axis represents the percentage of Gauss(0.7) with the remainder percentage representing the other workload. Both BRHist and ARHist show robustness across workloads with stable NRE and CIME. ARHist outperforms BRHist, as shown in the figures, the former has less NRE and smaller CIME.

6.5 Issuing queries over data streams

In order to observe the performance of adapting RHist during a continuous stream of data, we mixed query streams with data streams. The data stream is interrupted, and a set of queries is issued, then the data stream is resumed. This process goes on until the end of the data stream. At each time unit, $50,000$ data items arrive from a synthetic dataset (default Zipf and default Guass). For the ADL dataset, the stream length is set to 2,090,000 and at each time unit, 110,000 data items are processed. The size of the query set is 100. And the query workload is Gauss(0.7). For clarity, we do not plot CIME in Figure 12.(a) and (b).

Figure 12.(a-c) shows that ARHist is quite stable over the entire data streams. Both BRHist and ARHist perform very well. In Figure 12.(a), BRHist is comparable to MaxDiff and V-Optimal histograms while ARHist outperforms these two static histograms, which are rebuilt at each time unit. Figure 12.(b) depicts the similar situation, except that V-Optimal histogram is slightly better than ARHist. In Figure 12.(c), both BRHist and ARHist outperform MaxDiff and V-Optimal most of the time. Note that since the y axis is logscale, the CIME for BRHist and V-Optimal histogram is much bigger than ARHist. In all the figures, ARHist improves the performance by 20% to 70% compared to BRHist and is quite robust in the presence of update.

7. CONCLUSION

In this paper we presented a new approach for maintaining summary information over data streams. Our approach is novel in that we adapt the summary information not only to the changes in the data distribution, but also to the changes in the query workload characteristics. We start by developing a variant of compressed histogram, referred as RHist. We first developed a basic adaptive algorithm in which RHist is refined for each incoming query. Since per query refinement can become prohibitively expensive and to gracefully incorporate workload changes, we introduced the concept of a workload decay model. Using this model, we developed the advanced adaptive algorithm for maintaining RHist. We conducted an extensive evaluation of these two approaches and compared them with other well-known histograms. Our results indicate that the adaptive approaches based on both workload and data distributions are robust and result in very low estimation errors.

8. REFERENCES

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning Histograms: Building Histograms Without Looking at Data. In *Proceedings of ACM SIGMOD Conference*, pages 181–192, 1999.

[2] S. Babu and J. Widom. Continuous Queries over Data Streams. Technical report, Stanford University, 2001.

[3] N. Bruno, S. Chaudhuri, and Luis Gravano. STHoles: A Multidimensional Workload-Aware Histogram. In *Proceedings of the ACM SIGMOD Conference*, pages 211–222, 2001.

[4] G. Casella and R. L. Berger. *Statistical Inference.* Wadsworth Pub, 2nd edition, 2001.

[5] C.M. Chen and N. Roussopoulos. Adaptive Selectivity Estimation Using Query Feedback. In *Proceedings of the ACM SIGMOD Conference*, pages 161–172, 1994.

[6] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining Stream Statistics over Sliding Windows. In *ACM-SIAM Symposium on Discrete Algorithms(SODA)*, 2002.

[7] D. Donjerkovic, Y.E. Ioannidis, and R. Ramakrishnan. Dynamic Histograms: Capturing Evolving Data Sets. In *Proceedings of the 16th ICDE Conference*, 2000.

[8] V. Ganti, M.L. Lee, and R. Ramakrishnan. ICICLES: Self-Tuning Samples for Approximate Query Answering. In *Proceedings of 26th VLDB Conference*, pages 176–187, 2000.

[9] J. Gehrke, F. Korn, and D. Srivastava. On Computing Correlated Aggregates Over Continual Data Stream. In *Proceedings of ACM SIGMOD Conference*, pages 13–24, May 2001.

[10] P. B. Gibbons and Y. Matias. Synopsis Data Structures for Massive Data Sets. Technical report, Bell Labs, 1998.

[11] P.B. Gibbons, Y. Matias, and V. Poosala. Fast Incremental Maintenance of Approximate Histograms. In *Proceedings of 23rd VLDB Conference*, pages 466–475, 1997.

[12] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M.J. Strauss. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries. In *Proceedings of the 27th VLDB Conference*, Italy, September 2001.

[13] M. Greenwald and S. Khanna. Space-Efficient Online computation of Quantile Summaries. In *Proceedings of ACM SIGMOD Conference*, pages 58–66, May 2001.

[14] S. Guha and N. Koudas. Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation. In *Proceedings of the 16th ICDE Conference*, 2002.

[15] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantee. In *Proceeding of 24th VLDB Conference*, New York, USA, 1998.

[16] N. Koudas, S. Muthukrishnan, and D. Srivastava. Optimal Histograms for Hierarchical Range Queries. In *Proceedings of the 20th ACM PODS*, 2000.

[17] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate Medians and other Quantiles in One Pass and with Limited Memory. In *Proceedings of ACM SIGMOD Conference*, 1998.

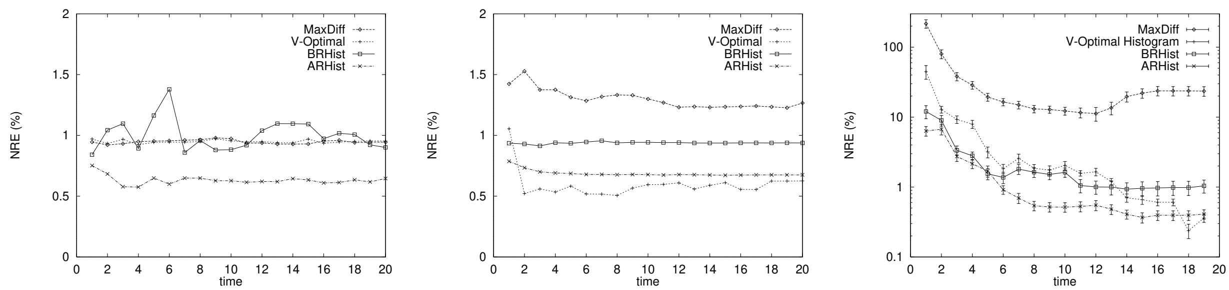
[18] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Datasets. In *Proceedings of ACM SIGMOD Conference*, 1999.

[19] Y. Matias, J.S. Vitter, and M. Wang. Wavelet-Based histograms for Selectivity Estimation. In *Proceeding of the ACM SIGMOD Conference*, 1998.

[20] Y. Matias, J.S. Vitter, and M. Wang. Dynamic Maintenance of Wavelet-Based histograms. In *Proceeding of 26th VLDB Conference*, 2000.

[21] Douglas C. Montgomery. *Design and Analysis of Experiments.* John Wiley & Sons INC, 5th edition, 2000.

[22] G. Piatetsky-Shapiro and C. Connell. Accurate Estimation of the number of Tuples Satisfying a Condition. In *Proceedings of the ACM SIGMOD*



(a) Zipf (b) Gauss (c) ADL dataset

Figure 12: Issuing queries over data stream

Conference, pages 256–276, 1984.

[23] V. Poosala. *Histogram-Based Estimation Techniques in Databases.* PhD thesis, Univ. of Wisconsin-Madison, 1997.

[24] V. Poosala and Y.E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proceedings of the 23rd VLDB Conference*, pages 486–495, 1997.

[25] V. Poosala, Y.E. Ioannidis, P. Haas, and E. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. In *Proceedings of the ACM SIGMOD Conference*, May 1996.

[26] J.S. Vitter. Random sampling with a reservoir. *ACM Transaction Mathematical Software*, 11(1):37 – 57, March 1985.

[27] Y.L. Wu, D. Agrawal, and A. El Abbadi. Applying the Golden Rule of Sampling for Query Estimation. In *Proceedings of ACM SIGMOD Conference*, pages 449–460, 2001.

[28] Y.L. Wu, D. Agrawal, and A. El Abbadi. Query Estimation by Adaptive Sampling. In *Proceedings of the ICDE Conference*, 2002.

[29] D. Zhang, V.J. Tsotras, D. Gunopulos, and B. Seeger. Temporal aggregation over data streams using multiple granularities. In *Proc. Int. Conf. on Extending Database Technology (EDBT)*, Feburary 2002.

[30] G. K. Zipf. *Human Behaviour and the Principle of Least Effort.* Addison-Wesley, Reading, MA, 1949.

APPENDIX

A. PROOF FOR LEMMA 4.1

Proof. Let $\mathcal{A}_{ext}(\mathcal{Q})$ be the exact answer of query Q. The estimated answer generated from b_i is $\mathcal{A}_{est}(\mathcal{Q}) = b_i.Count \times \frac{c-a}{b_i.MaxVal-a}$, $NRE = \frac{|\mathcal{A}_{est}(\mathcal{Q})-\mathcal{A}_{ext}(\mathcal{Q})|}{\mathcal{A}_{ext}(\mathcal{Q})}$.

The estimated answer from buckets b_i' and b_i'' is

$$\mathcal{A}'_{est}(\mathcal{Q}) = \begin{cases} b_i'.Count \times \frac{c-a}{b_i'.MaxVal-a} \\ \text{for } c < b_i'.\text{MaxVal} \\ b_i''.Count \times \frac{c-b_i'.MaxVal}{b_i''.MaxVal-b_i'.MaxVal} + b_i'.Count \\ \text{for } c \geq b_i'.\text{MaxVal} \end{cases}$$

and $NRE' = \frac{|\mathcal{A}'_{est}(\mathcal{Q})-\mathcal{A}_{ext}(\mathcal{Q})|}{\mathcal{A}_{ext}(\mathcal{Q})}$.

1) Considering the case when $c < b_i'.MaxVal$

a) Considering the case when the frequencies of values in this bucket are monotonically non-decreasing.

In bucket b_i the frequency starts from f_{min}, which is less than or equal to $\frac{b_i.Count}{b_i.MaxVal-a}$, i.e., the average frequency, and grows up to f_{max}, which is greater than or equal to the average frequency. Obviously, using the average frequency to estimate the distribution will result in over-estimating the answer. So $\mathcal{A}_{est}(\mathcal{Q}) \geq \mathcal{A}_{ext}(\mathcal{Q})$. For the same reason, $\mathcal{A}'_{est}(\mathcal{Q}) \geq \mathcal{A}_{ext}(\mathcal{Q})$.

Since the frequencies of values in the bucket are monotonically non-decreasing, the average frequency in b_i' is less than the average frequency in b_i. So $\frac{b_i'.Count}{b_i'.MaxVal-a} \leq \frac{b_i.Count}{b_i.MaxVal-a}$, where $\frac{b_i.Count}{b_i.MaxVal-a}$ is the average frequency in b_i and $\frac{b_i'.Count}{b_i'.MaxVal-a}$ is the average frequency in b_i'.

$\Rightarrow b_i'.Count \times \frac{c-a}{b_i'.MaxVal-a} \leq b_i.Count \times \frac{c-a}{b_i.MaxVal-a}$

$\Rightarrow \mathcal{A}_{ext}(\mathcal{Q}) \leq \mathcal{A}'_{est}(\mathcal{Q}) \leq \mathcal{A}_{est}(\mathcal{Q})$

$\Rightarrow \frac{\mathcal{A}'_{est}(\mathcal{Q})-\mathcal{A}_{ext}(\mathcal{Q})}{\mathcal{A}_{ext}(\mathcal{Q})} \leq \frac{\mathcal{A}_{est}(\mathcal{Q})-\mathcal{A}_{ext}(\mathcal{Q})}{\mathcal{A}_{ext}(\mathcal{Q})}$

$\Rightarrow NRE' \leq NRE$

b) Considering the case when the frequency of values in this bucket are monotonically non-increasing.

As opposed to case (1.a), using the average frequency results in under-estimating the answer. So $\mathcal{A}_{est}(\mathcal{Q}) \leq \mathcal{A}_{ext}(\mathcal{Q})$ and $\mathcal{A}'_{est}(\mathcal{Q}) \leq \mathcal{A}_{ext}(\mathcal{Q})$.

Since the frequencies of values in the bucket are monotone decreasing, the average frequency in b_i' is greater than the average frequency in b_i. So $\frac{b_i'.Count}{b_i'.MaxVal-a} \geq \frac{b_i.Count}{b_i.MaxVal-a}$,

The rest proof for this case is similar to that in case (1.a).

2) Considering the case when $c \geq b_i'.MaxVal$

a) Considering the case when the frequencies of values in this bucket are monotonically non-decreasing.

As we have proved in (1.a), $\mathcal{A}_{est}(\mathcal{Q}) \geq \mathcal{A}_{ext}(\mathcal{Q})$ and $\mathcal{A}'_{est}(\mathcal{Q}) \geq \mathcal{A}_{ext}(\mathcal{Q})$.

Let $\tilde{\mathcal{Q}}$ be a query with range (c, d], where d equals $b_i.MaxVal$.
$\Rightarrow \mathcal{A}_{est}(\mathcal{Q}) = b_i.Count - \mathcal{A}_{est}(\tilde{\mathcal{Q}})$ and $\mathcal{A}'_{est}(\mathcal{Q}) = b_i.Count - \mathcal{A}'_{est}(\tilde{\mathcal{Q}})$.
This case is symmetric to case(1.b) where the frequencies are monotonically non-increasing instead of non-decreasing and the query boundaries are reversed. From (1.b), $\mathcal{A}_{est}(\tilde{\mathcal{Q}}) \leq \mathcal{A}'_{est}(\tilde{\mathcal{Q}})$
$\Rightarrow \mathcal{A}_{ext}(\mathcal{Q}) \leq \mathcal{A}'_{est}(\mathcal{Q}) \leq \mathcal{A}_{est}(\mathcal{Q})$
$\Rightarrow \frac{\mathcal{A}'_{est}(\mathcal{Q}) - \mathcal{A}_{ext}(\mathcal{Q})}{\mathcal{A}_{ext}(\mathcal{Q})} \leq \frac{\mathcal{A}_{est}(\mathcal{Q}) - \mathcal{A}_{ext}(\mathcal{Q})}{\mathcal{A}_{ext}(\mathcal{Q})}$
$\Rightarrow NRE' \leq NRE$
b) Considering the case when the frequency of values in this bucket are monotonically non-increasing.
The proof for this case is similar to that for case(2.a).
Thus we prove that under all cases NRE' is less or equal to NRE. □

B. LEMMA B.1

LEMMA B.1. *WHist with a decimal fraction depth belongs to the workload decay model.*

PROOF. The granularity of workload information in WHist is bucket with equal width. In WHist $Q_0[j] = (Q.range \cap b_j.range) \times \frac{1}{Q.range}$. From the initialization, we know, $W[j] = 1/n$ and $\sum_{j=1}^{n} W[j] = 1$. Here $W[j]$ is the depth of b_j. During the decay process, $W[j] = (1-\lambda) \times W[j] + \lambda \times Q_0[j]$. By induction, $\sum_{j=1}^{n} W[j] = \sum_{j=1}^{n}(1-\lambda) \times W[j] + \sum_{j=1}^{n} \lambda \times Q_0[j] = (1-\lambda) \times 1 + \lambda \times \sum_{j=1}^{n} Q_0[j] = 1 - \lambda + \lambda \times \frac{1}{Q.range} \sum_{j=1}^{n}(Q.range \cap b_j.range) = 1$. Thus the definition and invariant of workload decay model are satisfied by RHist. □

C. LEMMA C.1

LEMMA C.1. *If query workload is uniformly distributed, $W[j] \rightsquigarrow \frac{1}{d}$ for all $j \in domain$, when the number of queries tends to infinity, where d is the number of distinct values.*

D. LEMMA D.1

LEMMA D.1. *Let X be a random variable of workload W at any value, $\frac{1}{d}$ is an unbiased estimator for the expectation of X, where d is the number of distinct values.*

PROOF. Let Y be a random mapping function from value j to $W[j]$ for all $j \in domain$. No matter what kind of mapping Y is from domain to W, we have invariant $\sum_{j=minval}^{maxval} W[j] = 1$, thus $\bar{X} = \frac{1}{d}$. Because sample mean is the unbiased estimator of expectation, $E(X|Y) = \bar{X}|Y = \frac{1}{d}$ for all Y when number of samples tends to infinity. EX $= E_Y E_{X|Y}(x|y) = E_Y \frac{1}{d} = \frac{1}{d}$. □

E. VERIFICATION OF NORMALITY ASSUMPTION

```
                   Normal Probability Plot
 0.325+                                                    *
      |                                               ******
      |                                            ******
      |                                         *****
      |                                     +****
      |                                ********
      |                         ******
      |                    ******
      |             ********
      |           *****
      |       +****
      |   ++++++
      |******
-0.325+*
       +----+----+----+----+----+----+----+----+----+----+
           -2        -1         0        +1        +2
```

Generally, probability plotting is a graphical technique for determining whether sample data conform to a hypothesized distribution based on a visual examination of the data. To obtain sample points, we first feed 2 sets of 100 queries, i.e. Zipf and Gaussian distributed queries, into workload decay model. Because $W(j)$ at each value point j follows the same distribution. After each query, we collect $W(j)$ at all value points as observations. To construct a probability plot, samples are randomly assigned to be negative or positive. We conduct this experiment using SAS. And the plotted points fall along a straight line, which means our hypothesized model–normality assumption is reasonable.