Incremental Quantization for Aging Data Streams

Fatih Altiparmak, David Chiu, Hakan Ferhatosmanoglu[†]
Department of Computer Science and Engineering
The Ohio State University
{altiparm, chiud, hakan}@cse.ohio-state.edu

Abstract

A growing number of applications have become reliant or can benefit from monitoring data streams. Data streams are potentially unbounded in size, hence, Data Stream Management Systems generally maintain a "sliding window" containing the N most recent *elements. In an environment where the number of stream sources can vary, the amount of storage available to hold the sliding window can reduce dramatically. However, it has already been noted that as data becomes older their relevance tends to diminish until they are ultimately discarded from the sliding window. Based on this assumption, we propose to "wound" older data elements by relaxing their storage requirements as an effort constantly free enough space to keep pace with accurate representation of incoming elements in a process that we call aging. We propose two incremental quantization techniques that enable aging in an efficient manner. We will show that, by relaxing storage utilization of the summary created by our quantizers, the older data elements are not rendered useless. In fact, we will show that their accuracy is only lessened by a sustainable amount.*

1 Introduction

Data Stream Management Systems (DSMS) have gained significant interest in the database community due to a growing number of applications that deal with continuous data, e.g., environmental monitoring and network intrusion detection. The general challenge that comes coupled with the data stream model is storage management. Once an element from a data stream has been processed it is discarded or archived - it cannot be retrieved easily unless it is explicitly stored in memory, which typically is small relative to the size of the data streams [1]. Because data streams are potentially unbounded in size, storing the complete summary in memory is prohibitive. To overcome this issue, DSMS's generally maintain a constant size "sliding window" containing the N most *recent* data stream elements. By restricting the stream application to finite bounds, it becomes feasible to store and access the most recent elements of streaming data efficiently, assuming that real-time applications mostly find recent data more relevant.

Consider multi-streaming models where M streams feed into a central DSMS server, which must maintain a sliding window over these streams. When M becomes large, system resources (e.g., CPU and storage) are consumed at a much more vigorous rate, becoming increasingly difficult to maintain Quality of Service constraints such as timeliness and accuracy. In order to alleviate CPU load under these strenuous conditions, recent development in DSMS design employs techniques such as accuracy adjustment and load shedding [3, 4, 15, 16]. The burden on storage has also received attention. Recall that the sliding window must be constant size in order to fit in memory for efficient processing. Summarization techniques including histograms [6, 7, 10], sketching [8] and sampling [2] pave the way to data reduction.

We propose an approach to mitigate the stress on storage in multi-stream systems by adding the element of *age* to the sliding window. Aging data streams is not a novel concept as Jagadish et al. [10] first proposed to take into account the age of the elements. Their approach aims to provide fast approximate point queries by updating a histogram summary. Ours differ in the way that we offer an online compression technique to hold the actual values (not just statistics) of the sliding window with minimal loss of accuracy, thus enabling a wide array of data mining tasks including clustering and such as K-Nearest Neighors queries. The intuition is established on the observation that stream applications carry natural tendencies to favor recent elements over older elements [13]. As data grows older, its relevance diminishes until it is ultimately discarded from the sliding window. Our technique "wounds" older data elements by relaxing their storage requirements. This regained space is then further utilized to capture higher accuracy for incoming elements.

We make several contributions in this paper. First,

[†] This research is partially supported by US National Science Foundation grants III-054713, CNS-0403342, and OCI-0619041.

we propose 2 novel online quantization techniques, *Incremental V-Optimal Quantization* (IVQ), and *Distortion Based Quantization* (DBQ) that achieve efficient on-demand compression. Correspondingly, we will also show that not only do IVQ and DBQ find near-optimal solutions, i.e., minimal loss in quantized data representation, they are also polynomially faster than current approaches to the optimal solution. Finally, we discuss the *Ladder Approach* for sensible storage distribution as a means to apply age to data elements.

The rest of this paper is organized as follows. In Section 2, we describe the role of quantization with regards to supporting aging on streams. We then propose IVQ and DBQ respectively in Sections 3 and 4. Next, Section 5 discusses the Ladder Approach towards aging data stream elements, followed by a performance analysis of our approach in Section 6. Finally, we conclude and discuss future directions in Section 7.

2 Background

Because quantization lies deep in our aging procedure, we first present a brief review of this technique. The goal of quantization is to constrain a given vector F of continuous values into B discrete intervals (with $B-1$ boundaries). Each discrete interval can be uniquely identified by a label consisting of $\lceil log_2(B) \rceil$ bits. Thus, the actual values in F are replaced with a $\lceil log_2(B) \rceil$ bit sequence which indicates the the interval with its closest representation. Albeit a loss in accuracy of the actual elements, this technique has been proven useful in many disciplines including image processing and signal encodings. Moreover, as the number of intervals increases the information lost due to the quantization is reduced. We illustrate this procedure with a simple example.

$F =$	0	2	3	6	7	15	16	20
Interval	0	0	0	0	0	1	1	1

Using $b = 1$ bit for quantization, the domain, $D = [0 \ldots 20]$, of the 8 given numbers is divided into $B = 2$ intervals, I_0 and I_1, separated by $B-1 = 1$ boundary. For sake of simplicity, we let the boundary $\psi = 10$. Then all given values $f \in F$ that are less than or equal to 10 are assigned to I_0 and the rest are assigned to I_1. Notice that only a single bit is needed for mapping all values in F to the closest interval.

Statistics can also be applied per interval to increase the information denoted by the bit representation. Typically, the mean or median of all values within each interval is used for this purpose. In order to maintain an accurately quantized synopsis, now the goal is to minimize the sum of the square of the differences (SSE) between all values within an interval and the statistic of the interval to which they are assigned. In this example, 3.6 and 17 are the means of I_0 and I_1 respectively, and $SSE_0 = 33.2$ and $SSE_1 = 14$. The problem becomes finding the interval boundaries such that they minimize $SSE_{total} = \sum SSE_k$.

Although our boundary was trivially assigned to 10 in this example, effective boundary identification is a non-trivial task on real number domains. For instance, if the boundary had been selected as $\psi = 5$, then the points 6 and 7 would have fallen into I_1. Because $0, 2, and 3$ are very close to each other, this would expectedly decrease SSE_0 to 4.67. However, it increases SSE_1 to 178.2, and $SSE_{total} = 182.78$. With regards to storage, assuming that it takes 4 bytes to represent integers and floating point numbers, without quantization the given values in F would allot 32 bytes. On the other hand, quantization observes the following attributes on storage:

- $4B$ bytes to store the B interval statistics.
- bM bits for the M b-bit representations used for interval mapping.

Quantizing F generates only 8 bytes $+8$ bits $= 9$ bytes to hold the lossy representation of the given values. Considering the fact that we must sustain some loss in data accuracy, recall that the main goal of quantization is to use minimal space to maintain a maximally accurate summary. It should now be clear that optimally minimizing SSE_{total} translates to this solution.

2.1 Enabling Aging

In our system aging refers to the process of wounding older data elements by relaxing their storage and then using this storage to capture the details of the synopsis belonging to the younger incoming elements. Hence, the design of the quantizer should be made appropriate not only to support such an operation, but to support it efficiently. We summarize the requirements that should be met by the quantizer as follows: (1) Efficient Aging — the final quantization function should be built from an incremental manner in such a way that wounding a bit from the synopsis takes minimal time. (2) Preserving High Accuracy — boundaries should be selected in such a way that SSE_{total} is minimized.

While the first requirement will be discussed in the next section, the second requirement refers to solving the V-Optimal Histogram Problem, which states, given a vector of M numbers and the number of desired intervals, B, find the $B-1$ boundaries such that SSE_{total} is minimized. While optimal solutions to this problem are polynomial in nature [9, 10, 6], there exists a notion that, when M is large, approximate solutions become more practical [10, 7, 12]. Unfortunately, none of these solutions fit our first requirement of an incremental quantizer construction. We show the need for an incremental design through an example with quantizing multi-stream data.

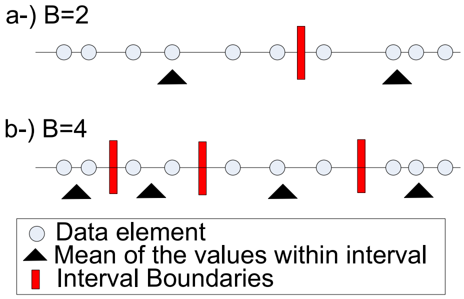

Figure 1. Challenge of the 1^{st} Criterion

In Figure 1, assume that our system handles $M = 10$ streams, and we let each point denote a single element arriving from a data stream. Suppose that these elements are currently represented in our sliding window with a $b = 2$ bit quantization, as depicted in Figure 1b. After some time, our system recommends that these data are now "older" and should be compressed to make space for more incoming elements. In order to relax a bit from the 2-bit representation, our system must store a corresponding 1-bit representation as shown in Figure 1a. Without a directly accessible 1-bit representation, it would become necessary to again quantize the 2-bit data using one less bit — an undesirable and seemingly redundant process. Furthermore, we would also expect too much error to be introduced by the re-quantization of already quantized (lossy) data. Another approach is to save both representations independently. Although it would guarantee a statically accessible representations, the algorithm would have to be run b times to obtain each. Moreover, its increased storage requirements for each set of data makes this option even more so unlikely for data stream systems. It is clear that a quantization algorithm is needed such that the boundaries of the quantization for b bits are preserved during the quantization for $b + 1$ bits. That is, a b bit quantization should be constructed iteratively such that new boundaries are added on top of existing boundaries. We note that the current solutions to the V-Optimal Histogram Problem are not incremental, and therefore unable to apply aging efficiently.

Incremental quantization techniques do, however, exist. Most notably, equal-length and equal-depth quantization are heuristics that naturally observe incremental behavior [10]. Their interval boundary selection criteria are resident in their names: equal-length iteratively partitions an interval into 2 equal halves while equal-depth divides intervals into halves containing an equal number of elements. While both techniques are mechanically efficient and incremental, the trivial boundary selection criteria may produce poor data accuracy on practical distributions. In light of this issue we propose a novel V-Optimal Histogram heuristic that identifies near optimal *incremental* intervals while achieving the same linear time complexity as equal-length and equal-depth.

3 Iterative V-Optimal Quantization

The proposed method, Iterative V-Optimal Quantization (IVQ), exploits a special case in the solution for V-Optimal Histogram. The major issue with directly employing the exact optimal solution [6] is its polynomial time complexity $O(M^2)$. However, we prove that for the special case of $B = 2$, finding an optimal boundary, ψ, for 2 intervals can be done in $O(M)$-time on a sorted set of M numbers. The premise of IVQ starts with a 0-bit summary, then iteratively using this $O(M)$-time subroutine, we add 1 bit to our summary at a time until the number of total bits is satisfied. The incremental construction of the quantized summary allows us the structure to remove bits efficiently. But first, we prove that when $B = 2$, the optimal solution can be found in $O(M)$-time.

We call this special case of $B = 2$ the *V-Optimal_B2* Problem. Given a sorted vector F of length M, we can state this problem as follows: find i such that

$$\sum_{k=1}^{i}(F[k]-Avg(F[1...k]))^2+\sum_{k=i+1}^{n}(F[k]-avg(F[i+1...n]))^2 \tag{1}$$

is minimized and $\psi = \frac{F[i]+F[i+1]}{2}$. We can rewrite equation 1 to obtain

$$\sum_{k=1}^{n}(F[k])^2-(i*avg(F[1...i])^2+(n-i)*avg(F[i+1...n])^2). \tag{2}$$

Now we see that minimizing equation 2 is equivalent to maximizing

$$i*avg(F[1...i])^2+(n-i)*avg(F[i+1...n])^2. \tag{3}$$

Note that if we define an array P, as in [10], of length M with $P[j] = \sum_{1\leq k\leq j} F[k]$ then we can rewrite equation 3 as

$$\frac{P[i]^2}{i}+\frac{(P[M]-P[i])^2}{M-i}. \tag{4}$$

After reformulation, we can see that finding the i that maximizes this equation is an $O(M)$-time routine.

The IVQ algorithm, shown in 1, is summarized as follows. First, we divide the quantization process into b incremental steps, where b again denotes the desired number of bits used in our final representation. Assume that for each step, we use only 1 bit per data element for a given set of numbers. Recall that using 1 bit results in $B = 2$ intervals. This implies that at each step j, we need only to solve the *V-Optimal_B2* Problem to find the boundary, ψ_j. For the

Algorithm 1 The IVQ Algorithm

```
$b$ // number of bits to be used in final quantization
$psi[1 \ldots B]$ // boundaries
$F[1 \ldots M]$ // input vector
list[] $\longleftarrow$ sort($F$)

IVQ(list[], start, end, j, psi_index)
        if $j <= b$ then
        {
            //call solution for v-optimal-b2 to get boundary
            psi[psi_index] $\longleftarrow$ v-optimal-b2(list, start, end)

            //call recursively on both sides of lists

            //first half
            IVQ(list, start, psi[psi_index], j+1, 2*psi_index)

            //second half
            IVQ(list, psi[psi_index] + 1, end, j+1, 2*psi_index+1)
        }
```

numbers in F on the left of the boundary, i.e., $< \psi$, the corresponding bit is assigned 0 and the rest 1.

To illustrate, when $j = 1$, we add the first bit to the synopsis. In do this, we first employ the above solution for V-Optimal_B2 Problem and find ψ_1 and the statistics of the two intervals. Then when $j = 2$, we add the second bit by apply our algorithm recursively to find ψ for each half. This process is continued until we reach $j = b$ whereby all $2^b - 1$ ψ's (and $B = 2^b$ interval statistics) are computed. The complexity of IVQ is straightforward. For a list of size M, we run the $O(M)$-time subroutine for *V-Optimal_B2*, then the list is split into 2 halves, and the algorithm is applied recursively on both halves. The number of splits is directly dependent on b. Therefore, cost of this algorithm is $O(bM)$, while keeping in mind that b is typically constant. The procedure is illustrated for $b = 3$ in Figure 2.

Recall that the original need for an incremental quantizer involved Requirement (1): efficient aging. Returning to Figure 2, when adding bits to the quantization summary, notice that the boundaries of the intervals from the previous iteration are preserved and reused. To visualize the process of aging, assume that we have a 3-bit representation of the data elements belonging to a particular time unit and we want to remove one bit from this representation. Because our summary was built in an incremental manner, we can simply remove the least significant bit from each stream in order to obtain the exact representation for the 2-bit summary. The removal of this bit only costs $O(1)$-time. To support this low cost, we must maintain all statistics $Avg_{1,1}$, $Avg_{1,2}$, $Avg_{2,1}$, $\ldots$, $Avg_{3,8}$, to map to the intervals. This implies saving $\sum_{i=1}^{k} 2^i$ means for some k-bit representation. The cost of this storage is amortized for a massive amount of streams (i.e., when M is large), but it should be noted that while deleting the k^{th} bit, we also can delete all averages belonging to the quantizer of that bit: $Avg_{k,1}$, $\ldots$, $Avg_{k,2^k}$, thus regaining that space.

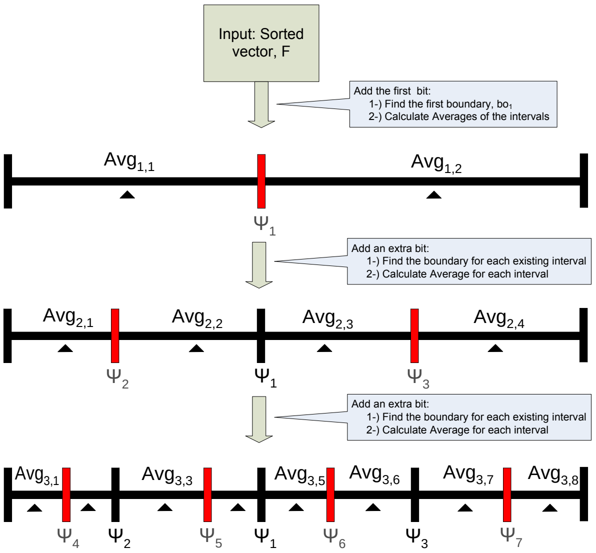
Input: Sorted vector, F
Add the first bit:
1-) Find the first boundary, bo1
2-) Calculate Averages of the intervals
Avg1,1
Avg1,2
Ψ1
Add an extra bit:
1-) Find the boundary for each existing interval
2-) Calculate Average for each interval
Avg2,1
Avg2,2
Avg2,3
Avg2,4
Ψ2
Ψ1
Ψ3
Add an extra bit:
1-) Find the boundary for each existing interval
2-) Calculate Average for each interval
Avg3,1
Avg3,3
Avg3,5
Avg3,6
Avg3,7
Avg3,8
Ψ4
Ψ2
Ψ5
Ψ1
Ψ6
Ψ3
Ψ7

Figure 2. Example Run of IVQ for $b = 3$

As for achieving Requirement (2): accuracy, we address IVQ's efficacy in data representation as compared to the optimal solution in Section 6.

4 Distortion Based Quantization

Shen and Hasgawa previously proposed a distortion based quantization method, an adaptive incremental LBG, which is a K-Means clustering based method [14]. This algorithm increases the number of quantization cells one after another. At each iteration, it finds the quantization cell that introduces the maximum error, and sets a random data point within the cell as the representative of the new quantization cell. After this, the LBG clustering algorithm is executed in order to find the new clusters and the new represenetives. As the boundaries can be altered in the clustering step, this technique does not guarantee the preservation of the old boundaries.

In this section we will introduce our second incremental quantization technique, Distortion Based Quantization (DBQ), which was inspired by this approach. This technique is akin to IVQ in that we still exploit the $O(M)$-time solution for the V-Optimal_B2 Problem. The only difference lies in where this subroutine is applied. IVQ applies directly on both halves of the list, thereby giving all existing intervals a fair share of details. In contrast, at each iteration, DBQ applies to the interval observing the maximum SSE. This allows for the formation of finer-grained sub-intervals when variance between the points within an interval is high, leading to more accurate quantization for highly skewed data sets. This feature is not without cost, as DBQ is $O(M^2)$ in the worst case, and can only offer

$O(M)$-time bit removal, both polynomially larger than its IVQ counterpart. The algorithm is shown below in Algorithm 2.

Algorithm 2 The DBQ Algorithm

```
b // number of bits to be used in final quantization
$psi[1 \dots B]$ // boundaries
$F[1 \dots M]$ // input vector
list[] ← sort($F$)

DBQ(list[])
  $psi[1]$ ← min(list)
  $psi[2]$ ← max(list)
  j = 1;
  used = 2;
  while $j <= b$
  {
      while $used <= 2^j + 1$
      {
          //find the Interval with the maximum SSE
          max ← IntervalWithMaximumSSE(list, psi, used)

          //shift the boundaries right to insert the new boundary
          $psi[max + 2 : used + 1] \leftarrow psi[max + 1 : used]$

          //call solution for v-optimal-b2 to get boundary
          $psi[max + 1]$ ← v-optimal-b2(list, $psi[max]$, $psi[max + 2]$)
          used = used+1
      }

      if j =1,
              OLD_Boundaries = psi[1:used]
      else,
              CURRENT_Boundaries = psi[1:used]
              MAP(OLD_Boundaries, CURRENT_Boundaries)
              OLD_Boundaries = CURRENT_Boundaries
      j = j+1
  }
```

We now show that this algorithm also satisfies both of our criteria. To satisfy Requirement (1), the enabling of efficient aging, we digress to an example run of this technique, as illustrated in Figure 3. The first bit is added so that ψ_1 divides the entire dataset into two halves. The second bit allows us to add two additional boundaries ψ_2 and ψ_3. After determining that the righ half contains the most distortion, we place ψ_2 in this interval. The same procedure applies to placing ψ_3, giving us a total of 4 intervals.

Aging on this synopsis is not as efficient as IVQ. It is easy to see that simply removing the least significant bit from the synopsis is no longer sufficient. In our example, the elements within the “01”, “10”, and “11” intervals should be replaced with “1” after removal of a bit. No simple bit removal rule exists for this case. Consequently, an additional mapping is required to point “01”, “10”, and “11” to “1”. Because this mapping can be saved for each interval, so there is not an additional storage cost for each data element in this technique. Unfortunately, deleting a bit now takes $O(M)$-time because each data element needs to be observed and remapped. This is the tradeoff for properly handling highly skewed data sets.

For Requirement (2), the accuracy of the quantization relative to the optimal solution is shown through experimental results are provided in Section 6.

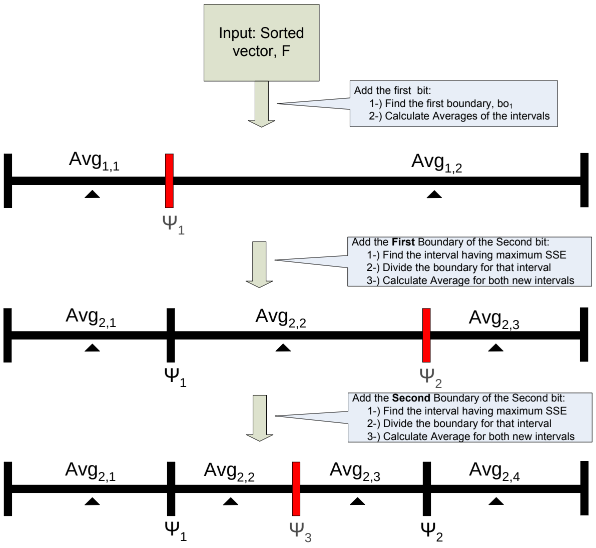
Input: Sorted vector, F
Add the first bit:
1-) Find the first boundary, bo_1
2-) Calculate Averages of the intervals
$Avg_{1,1}$
$Avg_{1,2}$
Ψ_1
Add the **First** Boundary of the Second bit:
1-) Find the interval having maximum SSE
2-) Divide the boundary for that interval
3-) Calculate Average for both new intervals
$Avg_{2,1}$
$Avg_{2,2}$
$Avg_{2,3}$
Ψ_1
Ψ_2
Add the **Second** Boundary of the Second bit:
1-) Find the interval having maximum SSE
2-) Divide the boundary for that interval
3-) Calculate Average for both new intervals
$Avg_{2,1}$
$Avg_{2,2}$
$Avg_{2,3}$
$Avg_{2,4}$
Ψ_1
Ψ_3
Ψ_2

Figure 3. Example Run of DBQ for $b = 2$

5 Overall Approach for Aging Data Streams

The concept of using *aging* on data streams to address the storage issue is born from the observation that newer streaming data is typically more relevant than the older [13]. This naturally corresponds to the idea of capturing more detail on the younger elements by sacrificing some detail from older elements. Our overall approach of supporting this is through storage manipulation: to utilize more storage for the data belonging to newer time units and less storage for older data, and devise a technique where more storage translates to higher accuracy.

We illustrate with an example. Assuming that the length of the sliding window is 5 time units, and additionally, the available memory for each stream is 15 units of storage. A simple way of distributing storage units would be to simply assign them uniformly across the sliding window: 3 bits per time unit. However, the interest of the queries is not quite as uniform. If most queries involve the first few time units, then we could sustain some depreciation of older elements. The storage can alternatively be assigned to the time units based on their order of the query interest. For instance, the youngest time unit obtains 5 units of storage, the second youngest gets 4 units, and so on.

We call this storage distribution method the *Ladder Approach*. At time $t + 1$, an element from each stream arrives at our system. Focusing on a single stream, under the ladder model, being these youngest elements should receive 5 units of storage. The question now is where to obtain the needed space. We salvage 1 unit of storage from the element belonging to time $t-4$ because it is now discarded from the sliding window. The data belonging at time t is no longer the youngest element in the synopsis, so it “steps down” the ladder and relaxes 1 bit of storage. The same holds for elements of $t-1, t-2$, and $t-3$. The outcome of this aging operation are that there remains 5 units of available storage for the new element at time $t+1$.

In general, the length of the sliding window, N, is much longer than the one given in our example. Assume that the total amount of storage available per stream is $\lceil(\omega+1)/2\rceil * N$ units, where ω units of storage is presumably enough to capture nearly all details of the data belonging to a time unit after quantization. Then the synopsis can be maintained in such a way that the ladder contains ω steps. Based on the above assumptions, the time units can be assigned to the steps of the ladder uniformly. So, in the case of N is a dividend of ω, the number of time units using some u units of storage is equal to that using $u+1$ units of storage. Otherwise, the N can be selected as the closest smaller multiplicand of ω. The starting time unit of the step s, for elements of which we are using s units of storage, can be calculated by $(t-N+1)+(s-1)*N/\omega$. As time passes from t to $t+1$, the oldest time units in each step of the ladder are shed one unit of storage, which corresponds to deleting the synopsis itself for the oldest element. The ladder is then shifted towards the future by the aging operation discussed above. This is summarized in Figure 4.

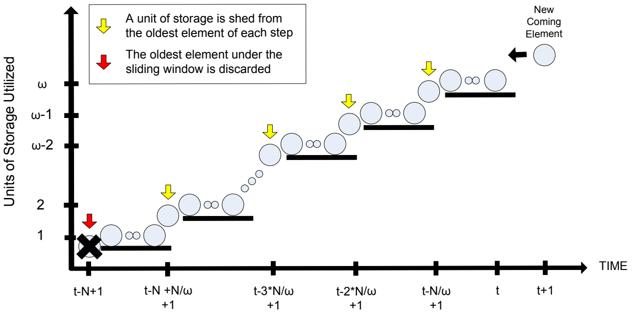
A unit of storage is shed from the oldest element of each step
The oldest element under the sliding window is discarded
New Coming Element
Units of Storage Utilized
ω
ω-1
ω-2
2
1
TIME
t-N+1
t-N +N/ω +1
t-3*N/ω +1
t-2*N/ω +1
t-N/ω +1
t
t+1

Figure 4. The Ladder Approach for Storage Distribution

It should now be clear as to why an incremental quantization technique such as those presented in the previous sections are suitable underlying structures for enabling our aging approach. We have described how IVQ is purposely built in such a way that its incremental data structure supports a constant time routine to age the entire sliding window, whereas DBQ is suitable for highly skewed data at a higher cost.

Focusing on our M stream system, our sliding window can be implemented by using nested linked lists. The outer list is a size N list where each node contains the summary of a specific time unit. This includes the boundary statistics and the bit mappings. The mappings itself is another list where each node contains M bits at some jth position. Aging on IVQ simply involves traversing each outer node and removing its first inner node (the most significant bit). DBQ aging, as we mentioned before, is not as straightforward. After removal of the most significant bit, we must reassign its value to the saved mapping as discussed in Section 4.

6 Experimental Results

In this section, we will compare the performance of all previously discussed quantization techniques with the optimal solution for 8 datasets of 10000 data points. The first 7 datasets follow the chi-square, exponential, laplace, loglogistic, gaussian, triangular, and uniform distributions respectively. The last dataset is the union of the equal-sized samples, without replacement, driven from the previous 7 distributions. These 8 datasets are created by using the Minitab Toolkit [5] and their respective histograms are shown in Figure 5.

In Figure 6, the x-axis denotes the number of used bits, and the y-axis denotes the Mean Squared Error (MSE), which is simply the ratio SSE_{total}/M. The values closer to 0 signifies better quantization in terms of data accuracy. We compared the exact solution [10] for the V-Optimal Histogram Problem with the incremental techniques equal-depth, equal-length, IVQ, and DBQ. As can be seen in all graphs shown in Figure 6, IVQ and DBQ lead to smaller error ratios than equal-depth and equal-length. While the plots belonging to IVQ and DBQ ostensibly overlay that of the optimal solution in all 8 distributions, marginal errors do exist between these two techniques and the optimal. To present this with finer detail, we extracted the exact numerical results from the *chi-square* dataset, and it is shown in Table 1. We see that only a minute amount of variance between V-Optimal and IVQ, and between V-Optimal and DBQ can be observed regardless of the number of bits used for that step in the quantization. The major difference is that IVQ and DBQ are incremental, and fit our purpose for supporting aging, while V-Optimal does not (it would be too inefficient in that the wounding process cannot be done efficiently). Although Table 1 only displays the *chi-square* dataset, it should be noted that all other distributions follow similar numerical results.

As can be seen from the graphs in Figure 6, for 5 bits, the MSE for the IVQ and the DBQ techniques are very close to 0 for all datasets. The conclusion we can derive from these graphs is that a 5-bit quantization scheme seems to be enough to represent a given vector of size $M=10000$ with these techniques with minimal noise.

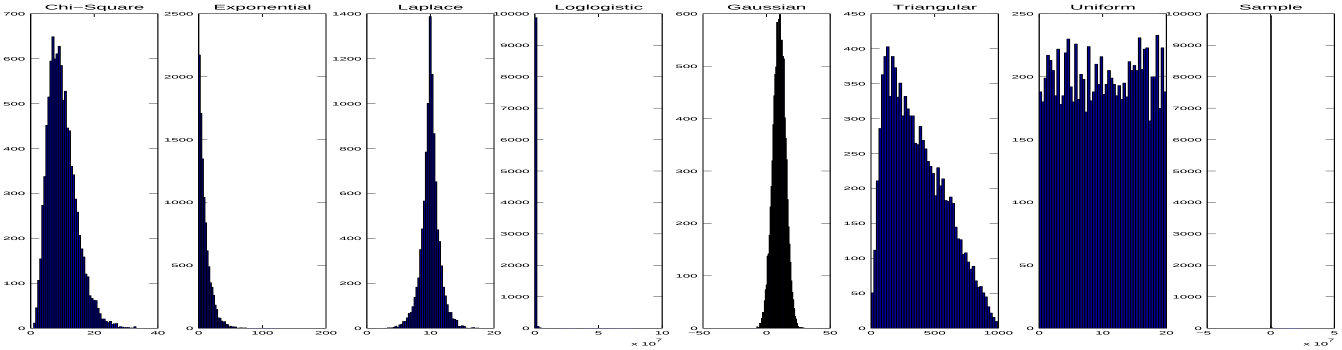

Figure 5. Distributions of the datasets

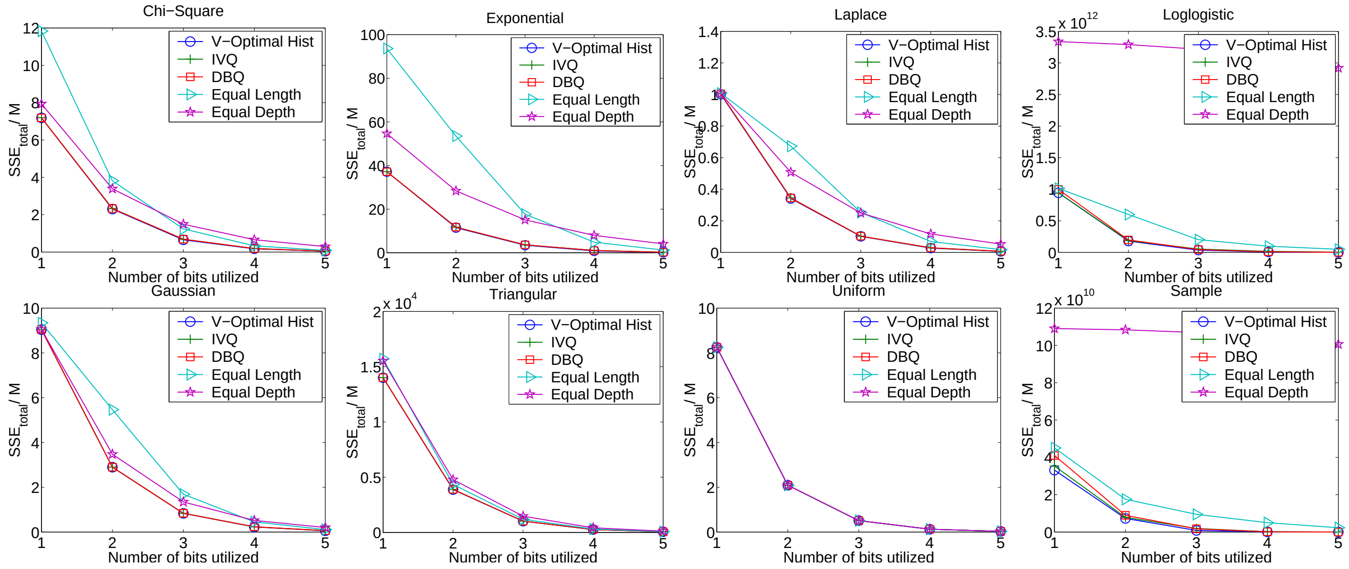

Figure 6. Comparison of different quantization techniques with respect to SSE_{total}/M

	1-Bit	2-Bit	3-Bit	4-Bit	5-Bit
V-Optimal	7.191	2.299	.653	.171	.043
IVQ	7.191	2.327	.684	.185	.046
DBQ	7.191	2.326	.685	.186	.047
Equal-Length	11.83	3.813	1.238	.331	.085
Equal-Depth	7.936	3.386	1.489	.6565	.291

Table 1. Exact values of the MSE ratio SSE_{total}/M for chi-square Distribution with 10000 Elements

In light of the above claim, we examined the effect of increasing the number of elements in these 8 datasets on the performance of the IVQ technique. Due to space restrictions, the remaining experimental results are shown only for IVQ because it is more efficient for aging and its performance is as good as DBQ. From each distribution we obtain a sample of 4 different sizes, $\{5000, 10000, 50000, \text{and } 100000\}$. Table 2 shows the maximum ratio between $SSE_{total}/M * Variance(Data)$ for these datasets by using $\leq$ 6 bits for quantization. We use this normalization, $SSE_{total}/M * Variance(Data)$, in aim of comparing quantization techniques such that they can be qualified independently apart the actual data set. The concept is that SSE_{total} for 0-bit representation is equal to M times the variance of the data. By measuring the ratio $SSE_{total}/M * Variance(Data)$, we can now claim that the values closer to 0 signifies better quantization in terms of data accuracy, and those closer to 1 corresponds to noisier representation. As shown in Table 2, the maximum of this ratio increases only slightly when we increase the size of the datasets. This supports our claim that a 5-bit quantizer is adequate such that SSE_{total} is minimized.

Size of Datasets	1-Bit	2-Bit	3-Bit	4-Bit	5-Bit	6-Bit
5000	.4970	.1754	.0568	.0169	.0039	.0009
10000	.4922	.1689	.0503	.0142	.0034	.0009
50000	.4970	.1750	.0550	.0163	.0045	.0012
100000	.5002	.1766	.0545	.0160	.0044	.0011

Table 2. Max ratio between SSE_{total}/$M * Variance(Data)$ for datasets from 7 distributions

6.1 An Optimization for Improving Quantization

Up to now, it should be clear that the data being considered for quantization are the elements from all M streams that arrived during some time t. That is, the given data at time t is the vector $F_t = \{f_{1,t}, f_{2,t}, \ldots, f_{M,t}\}$ where $f_{i,t}$ denotes the data element corresponding to the ith stream at time t. At first, it seems natural to quantize F_t directly, (which is similar to Vector Approximation (VA-file) [17, 11] employed by multidimensional indexing techniques). But since data elements across all M streams need not to be correlated, the errors introduced by quantization may become large. We use a simple example to expose this problem. Observe the stream manager in Figure 7. Here we see that while stream data is generally correlated across time, no correlation can be expected from the independent stream sources. The result is that the elements in F_t can be quite devious from each other, causing us to have little control over the growth of SSE_{total} if M is large. This is due solely to the fact that the independent sources cause an *unbounded* range of F_t.

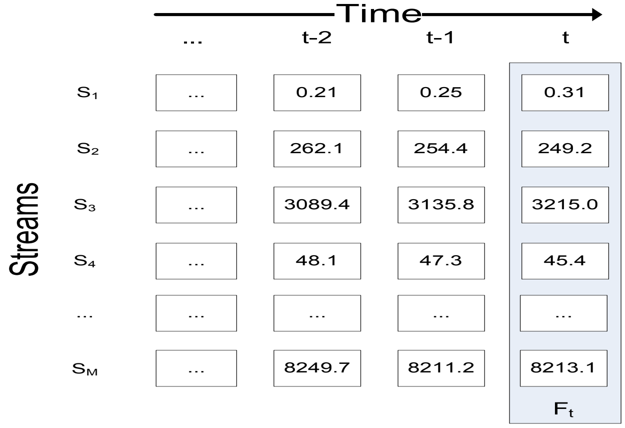

Figure 7. Data range before optimization

We must somehow normalize or compact the values in F_t to a more manageable range. Our compaction method involves mapping F_t to F'_t by storing the mean, μ_i, for each stream S_i, and replacing each $f_{i,t}$ with its difference from its corresponding mean. That is, $f'_{i,t} = (f_{i,t} - \mu_i)$. The hope is that the means of each stream are constantly representative of all data elements across the length of the sliding window. Thus, by using these differences instead of the actual values, we can expect a much smaller range of values in F'_t. This process is shown in Figure 8. Reconstructing a quantized data element involves the extra step of adding itself to the corresponding mean: $\hat{f}_{i,t} = (\hat{f}'_{i,t} + \mu_i)$, where $\hat{f}_{i,t}$, and $\hat{f}'_{i,t}$ are the reconstructed versions of $f_{i,t}$, and $f'_{i,t}$ respectively. $\hat{f}'_{i,t}$ is the mean of the interval that $f'_{i,t}$ falls into. Hence, its value changes when we wound a bit from time unit t.

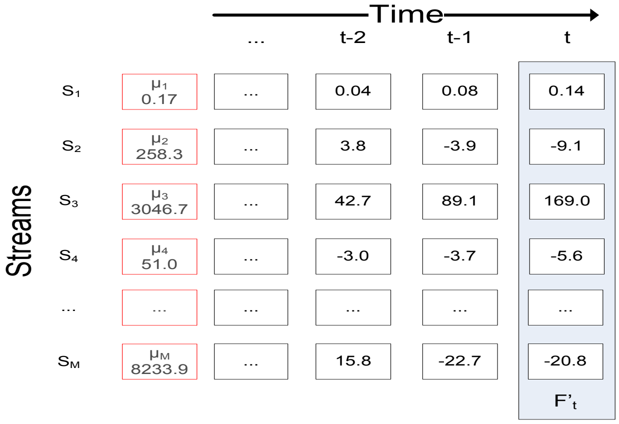

Figure 8. Data range after optimization

Figures 9 and 10 illustrate the effects of applying this optimization to actual weather and stock datasets respectively. The weather dataset is obtained from the National Climatic Data Center and consists of the temperature measurements for the year 2000 (366 days), sent from $5,009$ weather stations worldwide. The second one consists of daily stock values of $6,470$ companies for 360 days. In both datasets, the mean of each individual stream the first 128 time units are calculated and used for the remaining time units. The figures depict the average SSE per stream for each time unit after applying IVQ technique on the actual data and the deviation from the mean separately. In each figure, the top graph belongs to the actual data, whereas the bottom graph is depicting the average SSE_{total} after getting the difference. As one can see, the SSE_{total}/M of the quantization is greatly affected when b is smaller. This translates to better data accuracy for elements belonging to the "older" portions of our ladder.

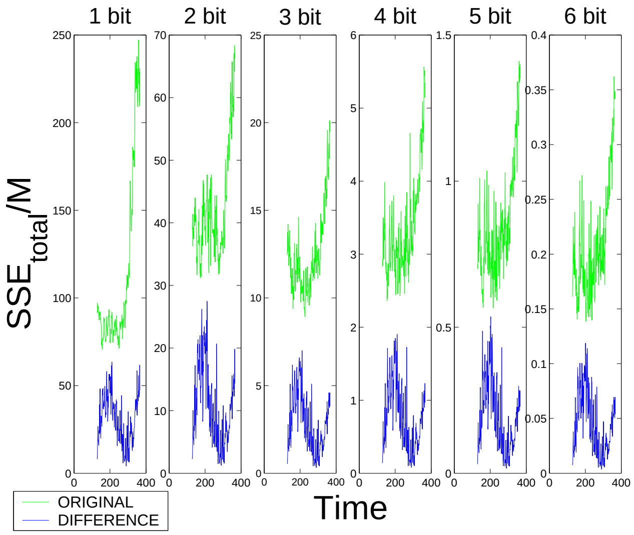

Figure 9. Comparison of average SSE_{total} between the original and the difference from mean for the Weather Data

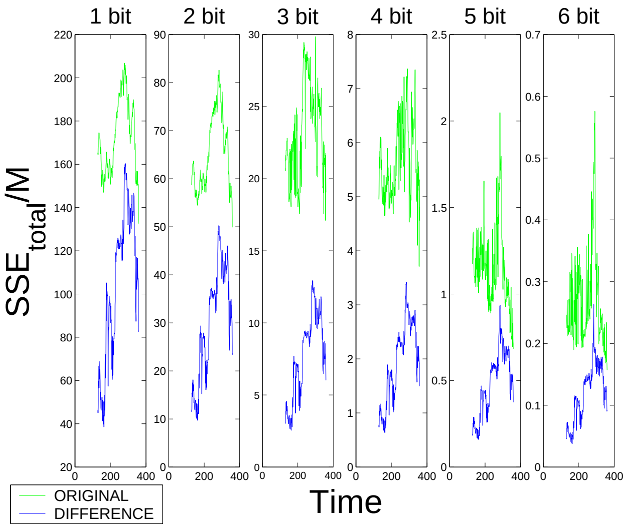

Figure 10. Comparison of average SSE_{total} between the original and the difference from mean for the Stock Data

7 Conclusion and Future Work

As an effort to address storage issues resident in data stream applications, this paper added the element of age to sliding window data. Our contributions include two incremental quantization techniques, IVQ and DBQ. These linear heuristics, while more desirable due to the real-time constraints of most stream applications, have been shown to suffer only minor losses in data representation accuracy compared to the optimal solution. Finally, we described the *Ladder Approach* to apply the correct ages to specific portions of data, allowing our system to maintain a more robust sliding window.

Admittedly, the *Ladder Approach* makes a general assumption that user queries are mostly interested in the most recent data elements. Possibilities of problems caused by this assumption include:

1. The query interest may not be an increasing function.
2. The interest of the queries on a particular time point, $t - K$ where t is the current time and K is the age of the element, may change as time passes.
3. The number of steps in the ladder due to the distribution of the interest may be different from ω.

During the quantization phase, the actual data is accessed once and then forgotten, since we only retain its error from the mean of its stream. Therefore at any point of time, after shedding a portion of the synopsis we cannot regenerate that part again. Hence, any possible storage distribution scheme should obey the following rule: “the amount of storage utilized for a time unit cannot exceed the one used for a younger time unit.”

As a result, the first step in the storage distribution based on the query interest function, I, should be converting I to an increasing function, IL. Then, the available storage can be assigned to the time units proportional to the corresponding values in IL. The input of I is the age of the time unit and its output is the number of queries interested in the data elements having that age. This conversion to IL can easily be accomplished as follows: $IL[i] = max(I([i \dots N]))$, where N is the size of the sliding window. Since IL can be dominated by a single value, a method that *smooths* the given I in a better manner is needed.

The methods discussed in this paper have also larged relied on the assumption that data arrival rates are constant. But as we well know, this situation is hardly typical, and the problems they cause should not be overlooked. For instance, a sharp increase in data arrival rates will saturate the sliding window with all young elements, while older elements will already have been dropped from the synopsis. Depending on the application, a sliding window that hypothetically only contains elements arriving in the past k milliseconds is probably not all that useful. We believe that our aging method can also be applied here for supporting real-time based sliding windows under dynamic environments. Future work on this topic will certainly be under this direction.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, pages 1–16, New York, NY, USA, 2002. ACM Press.

[2] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *ACM-SIAM Symposium on Discrete Algorithms(SODA)*, 2002.

[3] B. Babcock, M. Datar, and R. Motwani. Load shedding for aggregation queries over data streams. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 350, Washington, DC, USA, 2004. IEEE Computer Society.

[4] L. Chen and G. Agrawal. Supporting self-adaptation in streaming data mining applications. In *IPDPS 2006*. IEEE.

[5] C. DU FEU. Minitab 14. *Teaching Statistics*, 27(1):30–32, 2005.

[6] S. Guha. Space efficiency in synopsis construction algorithms. In *Very Large Databases(VLDB)*, pages 409–420, 2005.

[7] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 471–475, New York, NY, USA, 2001. ACM Press.

[8] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *In Proc. of the 26th Int. Conf. on Very Large Data Bases(VLDB)*, September, 2000.

[9] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 233–244, New York, NY, USA, 1995. ACM Press.

[10] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 275–286, 24–27 1998.

[11] X. Liu and H. Ferhatosmanoglu. Efficient k-NN search on streaming data series. In *Symposium on Spatio-Temporal Databases (SSTD), Santorini, Greece*, July 2003.

[12] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:127–135, Mar. 1982.

[13] Y. Ogras and H. Ferhatosmanoglu. Online summarization of dynamic time series data. *The VLDB Journal*, 15(1):84–98, 2006.

[14] F. Shen and O. Hasegawa. An adaptive incremental lbg for vector quantization. *Neural Networks*, 19(5):694 – 704, 2006.

[15] N. Tatbul, U. Cetintemel, S. Zdonik, M. Chemiack, and M. Stonebraker. Load shedding in a data stream manager. In *Very Large Data Bases (VLDB)*, Berlin, Germany, 2003.

[16] Y.-C. Tu and S. Prabhakar. Control-based load shedding in data stream management systems. In *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06)*, page 144, Washington, DC, USA, 2006. IEEE Computer Society.

[17] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *International Conference on Very Large Data Bases (VLDB)*, pages 194–205, New York City, New York, 1998.