# Estimation of the Number of Distinct Values over Data Stream Based on Compound Sliding Window

Yingli Zhong[1, 2]
1School of Computer Science and Technology, Heilongjiang University, Harbin, China, 150080
2Key Laboratory of Database and Parallel Computing Heilongjiang Province, Harbin, China, 150080
Email: zhongylmy@yahoo.com.cn

Jinghua Zhu[1, 2], Meirui Ren[1, 2]*and Yan Yang[1, 2]
1School of Computer Science and Technology, Heilongjiang University, Harbin, China, 150080
2Key Laboratory of Database and Parallel Computing Heilongjiang Province, Harbin, China, 150080
Email: renmeirui1972@sina.com

***Abstract*—Estimating the number of distinct values in a data stream is a vital problem with many applications such as complex join query over multiple data streams. In this paper, we focus on the continuous and periodic distinct values estimation over sliding windows. We propose a compound sliding window model to compute the distinct values over basic sliding windows in an incremental way. *LDV*, *HDV* and *AHDV* are the three algorithms that are based on compound sliding windows. The basic idea behind the compound sliding windows is to organize the basic windows into a Hash table according to distinct values. Whenever a new data arrives at the data stream, it is inserted into a basic window. Once the basic window is full, a scan using distinct values is executed and the distinct values number is updated incrementally. Theoretical analysis and experiment results show that the distinct values estimation algorithms based on compound sliding windows have a great performance benefits.**

***Index Terms*—data stream, basic window, compound sliding window, distinct values estimation**

## I. INTRODUCTION

In recent years, data stream are widely applied in application domains such as sensor network, network monitor and control, correspondence data management and stock analysis. Data stream is a new model of data processing. Data do not take the form of persistent relations, but rather arrive as a continuous, infinite, rapid, and time-varying data stream in the model [1]. Estimating the number of distinct values over data stream is an important problem with many applications such as complex join query over multiple data streams. Due to the infinite characteristic of data stream and the limit storage, it is impossible to maintain the whole data stream. Hence, we count the number of distinct values over data stream by scanning the data in the historical and can only obtain the approximate statistical results. Sliding window model is data sampling technique in nature; it is often used in data stream query operation and approximate result computation. A sliding window over a data stream is a window that is set on a segment of the data stream, and the segment only includes the latest coming data. When the new data arrives, the sliding window moves ahead by replacing the oldest data with the latest arriving one. There are usually two types of sliding window according to the updating granularity: continuously updating sliding window and periodically updating sliding window [2]. Nearly all the existing sliding window algorithms for the estimation of the number of distinct values over data stream are designed for continuously updating sliding window. Further, these algorithms did not take into consideration the effect of data structure on performance. However, in practical applications, we often need to get the query result in a time period. For example, when monitoring the operation of internet, the network supervisors need to know the statistic analysis result of the operation of the network in the last two hours between two integral points. So the data that arrives between the intervals cannot be inserted into the sliding window until the time reaches an integer point. In this paper, we propose a compound sliding window model to compute the distinct values over basic sliding windows in an incremental way. A compound sliding window [3] consists of several equally sized basic windows [4]. When the latest basic window is full, it is inserted into the compound sliding window; the expired basic window in the compound sliding window is deleted. The basic idea behind the compound sliding windows is to organize the basic windows into a Hash table according to distinct values. Whenever a new data arrives at the data stream, it is inserted into a basic window. Once the basic window is full, a scan using distinct values is executed and the distinct values number is

*To whom correspondences should be addressed.renmeirui1972@sina.com
Manuscript received January 25, 2012; revised June 19, 2012; accepted June 23, 2012.

updated incrementally. We also propose three distinct values estimation algorithms based on compound sliding window: LDV, HDV and AHDV. Theoretical analysis and experiment results show that the distinct values estimation algorithms based on compound sliding windows have a great performance benefits.

## II. Structure of the Compound Sliding Window

There are two ways to define a sliding window in data stream [5]: one is to define the sliding window based on sequence order; and the other is to define the sliding window based on time. In this paper, we will investigate the sliding window based on time; the related definitions are as follows.

**Definition 1** (**Data Stream**) A data stream is an infinite time sequence that is arranged in an ascending time order $S=\{<s_1,t_1>,<s_2,t_2>,\ldots,<s_i,t_i>,\ldots\}$,here $s_i$ is the sequence element that appears at time $t_i$.

**Definition 2** (**Sliding Window**) Let $T$ be a time interval, and $t > T$ is a moment of change. We call $S[t\text{-}T: t]$ a sliding window of $S$ within time interval $T$, here t and $T$ have the same units, and $t$ is the time delay with respect to the starting point of $S$.

**Definition 3** (**Basic Window**) Let $BT$ be a time interval, $t_1$, $t_2$ are changing time points, $BT=t_1\text{-}t_2$. We call $S[t_1: t_2]$ a basic window of $S$ within time interval $BT$, here $t_1$, $t_2$ and $BT$ have the same units, and $t_1$, $t_2$ are the time delays with respect to the starting point of $S$.

**Definition 4** (**Compound Sliding Window**) Let $S[t\text{-}ST: t]$ be a sliding window of data stream $S$ at time $t$, and $BT$ is the time interval of a basic window, let $ST=nBT$. If $S[t\text{-}ST: t]$ only changes at the end of each time interval $BT$, and at the end of the $k$th time interval of $BT$, it changes to $S[t+kBT\text{-}ST: t+kBT]$, then we call $S[t\text{-}ST: t]$ a compound sliding window, noted as $S_{BT}[t\text{-}ST: t]$. Here $t$, $ST$ and $BT$ have the same units.

The compound sliding window structure is shown in Figure1.

Generally speaking, $BT$ has two meanings. The first one is time period of the periodic continuous queries. The second one is to identify the upper bound of the time to calculate the query result on the compound sliding window. The users can choose different $BT$ values for specific applications; generally, $BT$ value should be much smaller than the size of the compound sliding window.

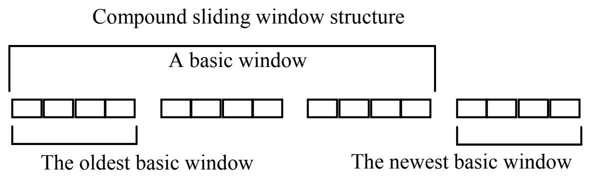


Figure 1.Compound Sliding Window Structure

## III. Estimation Algorithms Based on Compound Sliding Window

In this section, we propose three distinct values estimation algorithms over data stream based on compound sliding window: LDV, HDV and AHDV. LDV algorithm puts the basic windows of compound sliding window into chained lists, HDV algorithm and AHDV algorithm put the basic windows into hash tables. In addition, AHDV algorithm uses the $N$th number of distinct values to calculate the $N$+1th number of distinct values. For the convenience of description, the symbols used in this paper are listed in Table 1.

TABLE I
DEFINITION OF TERMS

| Symbol | definition |
|---|---|
| $T_{cswr}$ | Size of the compound sliding window of data stream $R$ |
| $T_{bw}$ | Size of the basic windows of data stream $R$ |
| $\lambda_r$ | Arriving speed of data stream $R$ |
| $C_n$ | Cost of accessing a tuple in chained algorithm |
| $C_h$ | Cost of accessing a tuple in Hash algorithm |

### A. LDV Algorithm

Since the distinct values estimation algorithm based on the compound sliding window is implemented over basic windows periodically, we can assign the same timestamps to all elements of a basic window. LDV algorithm uses the timestamp of the first-arriving element as the elemental timestamp for all the elements in a basic window. The algorithm uses a global counter as the timestamp of the arrival of an element. The data structure of the compound sliding window is a chain queue. The queue is divided into $T_{csw}/T_{bw}$ blocks. Every block is a basic window. Because the data rate of a data stream is constantly changing, the size of the memory occupied by the basic windows is changing too. Therefore, the data structure of the basic windows is a chained list. When a new element arrives, a new node will be requested and attached to the rear of the chained list. The basic window structure stores a head node which consists of one time stamp and two pointers. One pointer points to the element of that basic window, and the other points to the head node of the next basic window.

The basic idea of LDV algorithm is to store the distinct values over a data stream in a compound sliding window in a chained list leading by Estimate. When the newly arriving element fills up the newest basic window, the basic window is inserted into the current sliding window, and the outdated element in the basic window is deleted from the compound sliding window. Using the distinct values of the elements in the current compound sliding window to scan the chained list led by Estimate, if the value of the elements does not exist in the current compound sliding window, the element value will be inserted into the chained list led by Estimate. If the element value already existed in the chained list, the corresponding element value will be updated. In the end, the chained list led by Estimate is scanned through and the effective number of the distinct values in the compound sliding window is counted out.

Now we will analyze the time complexity of LDV algorithm. $C_n$ indicates the cost of accessing an element. LDV algorithm first inserts the newest basic window into the rear of the compound sliding window, the time cost of this step is the time of an address assignment, and we let it be $C_n$. The algorithm deletes elements in the expired basic windows of the compound sliding window of data stream $R$, we here make an assumption that the data stream has constant data rate, so there are $\lambda_r T_{bw}$ elements to be deleted. Since the compound sliding window is arranged in a time order, the time cost should be $\lambda_r T_{bw} \times C_n$. Then a scan of the compound sliding window is executed to count the number of the distinct values. The time cost of scan depends on the length of the chained list led by Estimate. We assume that the length of the chained list is $n$, the average searching length to determine the exists of the value is $(n+1)/2$, so the time cost is $\lambda_r T_{cswr} \times ((n+1)/2) \times C_n$. The last step is to output the number of the distinct values which has a cost of $n \times C_n$. To sum up, the average time cost of LDV algorithm is

$$C_n+\lambda_r T_{bw} \times C_n+\lambda_r T_{cswr} \times ((n+1)/2) \times C_n+n \times C_n \quad (1).$$

*B. HDV Algorithm*

The basic idea of HDV algorithm is similar to that of LDV algorithm. The data structure of basic window in HDV algorithm is Hash table, but the data structure of compound window in LDV algorithm is chain list which are sorted by distinct values.

Now we will analyze the time complexity of HDV algorithm. $C_h$ indicates the cost of accessing an element in HDV algorithm. HDV algorithm first attaches the newest basic window to the rear of the compound sliding window, the time cost of this step is the time of an address assignment. We assume that the data rate of the data stream is a constant, so there are $\lambda_r T_{bw}$ outdated elements to be deleted in a compound sliding window. Because the compound sliding window is arranged in time order, the time cost is $\lambda_r T_{bw} \times C_h$. In order to calculate the number of distinct values, HDV algorithm needs to scan the compound sliding window of data stream $R$. Here we use Bucket ($T_{bw}$) to indicate the number of Hash buckets in the basic window structure of data stream $R$, so the average number of elements in a bucket is $\lambda_r T_{bw}$/Bucket ($T_{bw}$). There is an element in every Hash bucket in the Hash table led by Estimate. Hence the time cost above is

$$\text{Bucket}(T_{bw}) \times (T_{cswr}/T_{bw}) \times (\lambda_r T_{bw}/\text{Bucket}(T_{bw})) \times C_h= \lambda_r T_{cswr} \times C_h. \quad (2).$$

Then the number of distinct values is output. Because the Hash table with distinct values needs to be scanned, here we assume its length to be $n$, so the time cost is $n \times C_h$. Based on the above analysis, the time cost of HDV algorithm is

$$C_h+\lambda_r T_{bw} \times C_h+\lambda_r T_{cswr} \times C_h+n \times C_h. \quad (3).$$

*C. AHDV Algorithm*

The main idea of AHDV algorithm is to count the number of distinct values in an incremental way. When a new element arrives at data stream $R$, AHDV algorithm first inserts it into the Hash table of the basic window according to its distinct values, and then makes a judgment of whether or not each value in the Hash table led by Estimate is expired. If not, AHDV algorithm compares each value of the new basic window with that of the Hash table led by Estimate. If the value already exists in the Hash table, AHDV algorithm updates its timestamp. If not, AHDV algorithm inserts it into the Hash table and calculates the number of non existing value. After that, AHDV algorithm scans each value of the Hash table led by current Estimate to count the number of outdated distinct values, and adds the number of non existing value to the number of distinct values of the current compound sliding window, besides subtracts the number of the expired distinct values, it then obtains the number of distinct values at the current refresh moment. In the end, AHDV algorithm inserts the newest basic window into the compound sliding window of data stream $R$. In order to determine whether the current element value is outdated, AHDV algorithm assigns the same timestamp to the distinct value of the basic windows. The AHDV algorithm is described as follows.

**AHDV algorithm**

```
Input: Estimate, BWRhead, CSWRfront,CSWRrear
Output: $m$
1.  Temp = CSWRfront;  $m=c$;
2.  WHILE (CurrentTimestamp-(Temp -> timestamp) >$T_{CSW}/T_{BW}$ )
3.      CSWRfront = Temp -> next;
4.      Delete every outdated element in the Hash table of
         basic window led by Temp;
5.      Temp = CSWRfront;
6.  END WHILE
7.  FOR (Every Hash bucket A led by BWRhead)
8.      Compare the Hash table led by Estimate with every
         Hash  bucket A led by BWRhead;
9.      IF   the value exists in the Hash table
10.         updates the timestamp;
11.     ELSE   inserts the value into the Hash table; $m$++;
12.     END IF
13. END FOR
14. FOR (Every Hash bucket A led by Estimate)
15.   IF (CurrentTimestamp-A.timestamp $\geq T_{cswr} / T_{bw}$ )
16.       $m$--;
17.   END IF
18. END FOR
19. CSWRrear -> next = BWRhead;
20. output $m$.
```

**Theorem 1**: The AHDV algorithm can calculate the number of distinct values correctly and terminate in finite steps.

**Proof**

**Correctness**: Step 1 assigns the number of distinct values of the previous moment to a temp variable $m$. In step 2 to 6, WHILE loop deletes the outdated elements of the compound sliding window, and thus there will be no outdated element in the compound sliding window at the current moment. In step 7 to 13, FOR loop calculates the number of distinct values $m$ at the current moment by comparing the Hash table pointed by Estimate with the new elements, and determines whether the number of distinct values changes. Note that the distinct values and timestamp of the previous moment are stored in the Hash table. In step 9 to 12, we check whether the new arrival elements already exist in the Hash table. If so, updates the timestamp. Otherwise, we insert the new arrival elements into the Hash table and update the number of distinct values *m*. Therefore, we can guarantee that the new distinct values are stored in the Hash table, and the number of distinct values is counted correctly. In step 14 to 18, FOR loop checks whether there are outdated distinct values in the Hash table. When some value expires, it will be deleted from the Hash table and the statistical count *m* will be updated. Step 19 inserts the newest basic window into the compound sliding window in order to correctly count the number of the distinct values at the next moment. In a word, the AHDV algorithm can calculate the number of distinct values in the current compound sliding window correctly.

**Termination**: The data arrives at the compound sliding window at a fix time interval and the size of the compound sliding window is finite. Furthermore, our algorithm counts the number of distinct values in the compound sliding window, so the number of distinct values is also limited in this time range. Therefore WHILE loop and FOR loop can be executed in finite steps, that is to say our algorithm can terminate in finite steps.

We analyze the time complexity of AHDV algorithm. Step 1 to 6 in AHDV algorithm aim at deleting all the elements in the outdated basic window of the compound sliding window. Here the data stream is assumed to have a constant data rate, so there are $\lambda_r T_{bw}$ elements to be deleted. Because the compound sliding window is arranged in time order, the time cost in Step 1 to 6 is $\lambda_r T_{bw} \times C_h$. Here we use Bucket ($T_{bw}$) to indicate the number of the Hash buckets that store the distinct values of the elements, there is one element in each bucket. So there are Bucket ($T_{bw}$) loops in Step 7. The step 8 to 13 aim at comparing the Hash table led by Estimate with every Hash bucket A led by BWRhead, so the time cost is one comparison and one assignment, we set it to be 2 Bucket ($T_{bw}$) $\times C_h$. Because the Hash table led by Estimate, which stores distinct values, needs to be scanned, the time cost depends on the length of the Hash table led by Estimate. We assume that the length of the Hash table is *n*, so there are *n* loops. The step 14 to 18 is to make a judgment of whether or not each value in the Hash table led by Estimate is expired, the time cost is one comparison, we set it to be $n \times C_h$. In step 19, the newest basic window is attached to the rear of the compound sliding window, that is implemented by assigning the value of BWRhead to CSWRrear. In the end, the number of distinct values is output, the time cost is only an address assignment, and we set it to be $C_h$. To sum up, the time cost of AHDV algorithm is

$$\lambda_r T_{bw} \times C_h + 2\ Bucket(T_{bw}) \times C_h + n \times C_h + C_h. \quad (4).$$

### *D. Comparison of Time Cost of LDV, HDV and AHDV Algorithms*

By comparing the time cost of the three algorithms, we can conclude that the AHDV algorithm does not need to scan the entire compound sliding window each time, it only needs to count the newest basic window, so its performance is the best among these three algorithms.

## IV. EXPERIMENT RESULTS AND PERFORMANCE ANALYSIS

Because the main influencing factor to the performance of LDV algorithm 、 HDV algorithm and AHDV algorithm is whether there is duplicated elements over data stream and domain of elements over data stream, two type of data are used in the experiments. The first data is non-duplicated data stream; and the other data is duplicated data stream. Non-duplicated data stream refers to the number of distinct values of data stream is equal to the number of elements. If the number of distinct values of data stream is less than the number of elements, we call it duplicated data stream. The hardware setting for the experiments is: T2370 1.73G, main memory 1G.

The elements in data stream *R* are duplicated in Experiment 1, where the size of compound sliding window is 100 seconds, the size of basic window is 10 seconds, and the data rate of the data stream is 1000 elements per second. The domain of elements is (0, 100). The result of experiment is shown in Figure 2. Because the size of compound sliding window is 100 seconds, in the first 100 seconds, the size of compound sliding window keeps growing. It can be seen from Figure 2 that, in the first 100 seconds, the time cost of the algorithm grows rapidly, this is because LDV algorithm needs to scan the whole compound sliding window at every time. Because the size of compound sliding window is fixed after the first 100 seconds, the time cost of the algorithm becomes stable. The basic windows of HDV algorithm and AHDV algorithm are Hash tables formed according to distinct values of elements, so the time costs are relatively low. Because AHDV algorithm doesn't need to scan the whole compound sliding window every time, and in the first 100 seconds, the elements of data stream won't be outdated, it has the lowest time cost among the three algorithms. After the first 100 seconds, because the elements may be outdated, a part of the compound sliding window should be scanned, the time cost of the algorithm increases, but it will never exceed the time cost of HDV algorithm. If there are many elements in the basic window, the elements seldom get outdated. Experiment 1 shows that AHDV algorithm has the best performance among the three algorithms.

The elements in data stream *R* are non-duplicated in Experiment 2. The left parameters are the same as those in Experiment 1; the results are shown in Figure 3. When the elements in data stream *R* are non-duplicated, because of the increase number of outdated elements, the total performance of AHDV algorithm falls down, but it is still better than the other two algorithms.

Because all the three algorithms need to scan the structure of saving the distinct values over data stream, the domain of the elements of data stream also has some influences on the performance of algorithms. The value in data stream *R* in Experiment 3, is also randomly distributed, where the size of compound sliding window is 100 seconds, the size of basic window is 10 seconds, the data rate of the data stream is 1000 elements per second. The result of the experiment is shown in Figure 4.

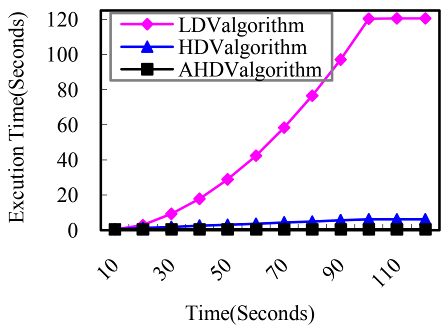


Figure 2. Performance comparison of LDV, HDV and AHDV on duplicated data

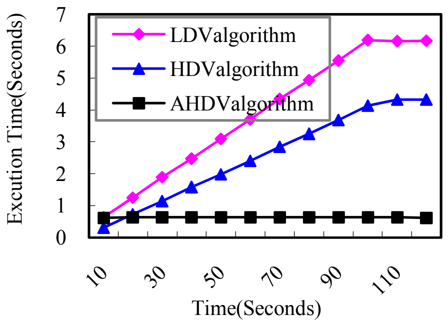


Figure 3. Performance comparison of LDV, HDV and AHDV on non-duplicated data

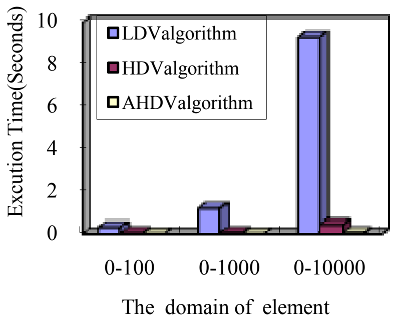


Figure 4. The influence of the domain of element on the performance of the algorithm

As the expansion of the domain of elements in the data stream, the length of the chained lists of LDV algorithm, in which the distinct values are stored, increases rapidly, the time cost of the algorithm increases significantly. HDV algorithm and AHDV algorithm use Hash structure to store the number of the distinct values, and this slightly increased the time cost of the algorithm. For the three algorithms, as the domain of elements of data stream expands, the performance of the algorithm degrades. However, the performance of AHDV algorithm is still better than the other two algorithms.

## V.CONCLUSIONS

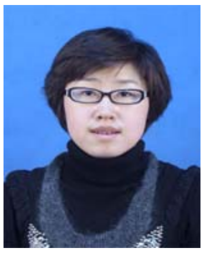
The estimation of the number of distinct values over data stream is vital for a variety of data stream applications, such as complex join query over multiple data stream. In this paper, incremental estimation algorithm of the number of distinct values for periodical updating is presented. The estimating algorithm is based on a compound sliding window model which uses a sliding window as a basic window period of updating. The effect of data structure in the basic window in the compound window on the performance of estimation algorithms is taken into consideration. We propose both non-incremental algorithm and incremental algorithm. The latter uses the $N$th estimation value to calculate the $N$+1th estimation value. Theoretical analysis and experiment results show that the incremental algorithms that organize the basic windows in the compound sliding window into hash tables have a better performance.

## ACKNOWLEDGMENT

This work was supported by The Science and technology research of Heilongjiang Education Office under Grant No. 12511401. It was partly supported by the Natural Science Foundation of Heilongjiang Province of China under Grant No.F201011, the Science Foundation of Heilongjiang University for Young Scholar under Grant No.QL201029, the National Natural Science Foundation of China under grant No. 61100048, the Harbin technological innovation fund under grant No.2011RFQXG028, 2012RFQXG096.

## REFERENCES

[1] LUKASZ G, TAMER M, OZSU. Data Stream Management Issues–A Survey[R]. *University of Waterloo Technical Report* CS–2003–08 April 2003.
[2] Weiping Wang, Jianzhong Li,Dongdong Zhang,Longjiang Guo. Periodically updating sliding window join algorithm s over data streams[J].*JOURNAL OF HARB IN INSTITUTE OF TECHNOLOGY*. 2005，37(6):756-759.
[3] Yingli Zhong,Weiping Wang,Longjiang Guo.Data Streams Join Aggregate Algorithms Based on Compound Sliding Window[C].*International Workshop on Database Technology and Applications*.Wuhan,China,2009:426-430.
[4] ZHU Y, SHASHA D. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time[C]. In Proc.28th Int.Conf.on Very Large Data Bases. *Hong Kong, China*. 2002: 358-369.
[5] BABCOCK B, BABU S, DATAR .M, et al. Models and Issues in Data Stream Systems[C]. *In Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*.2002: 1-16.



**Yingli Zhong** was born in 1976. Received the Master's degree from Heilongjiang University in 2004. She has been associate professor of Heilongjiang University since 2009. Her main research interests are database and wireless sensor network.

**Jinghua Zhu** was born in 1976. Received the Doctor's degree from Harbin Institute of Technology in 2009.She has been associate professor of Heilongjiang University since 2009. Her main research interests are wireless sensor network and Uncertain Database.

**Meirui Ren** was born in HeiLongJiang province of China, on July 3,1972. She achieved M.S. in Computer Software and Theory from HeiLongJiang university in 2002. Now, she is an associate professor of Computer Science and Technology School in HeiLongJiang university. Her research area is database, wireless sensor network.

**Yan Yang**, born in 1973. Received the Doctor's degree from Harbin Institute of Technology in 2005. She has been professor of Heilongjiang University since 2010. She has the membership of China Computer Federation. Her main research interests are database and parallel computing.