Technical Summary: "An Improved Data Stream Summary: The Count-Min Sketch and its Applications" by Cormode and Muthukrishnan

Ravi Gaddipati, Matthew Ige, and Emily Wagner

December 8, 2016

1 Problem Statement and Overview

Consider a vector $\vec{a}(t) = [a_1(t), \ldots, a_i(t), \ldots a_n(t)]$ which evolves with time. Initally, $\vec{a}$ is the zero vector $\vec{0}$. We represent the tth update as (i_t, c_t), which modifies the vector as follows:

$$a_{i_t}(t) = a_{i_t}(t-1) + c_t, \tag{1}$$

$$a_{i'}(t) = a_{i'}(t-1),\ i' \neq i_t \tag{2}$$

The update (i_t, c_t) modifies the ith element by adding c_t to it. For all other $a_{i'}$, the vector remains unchanged.

This vector and its updates represent some stream of data that evolves with time. There are 2 main models for such a stream:

1. **cash-register case:** $c_t > 0$, so every vector element is monotonically increasing.
2. **turnstile case:** c_t can also be negative. There are two subcases:
   (a) **non-negative turnstile:** (i_t, c_t) will never cause vector elements a_{i_t} to dip below zero. This guarantee can be provided by the application.
   (b) **general turnstile:** vector elements a_{i_t} may become negative.

The basic problem this work addresses is to summarize or calculate certain characteristics about the stream. There are 2 main constraints. First, the space used by such algorithms should be small; at most polylogarithmic in n. This compressed version of the data is called a **sketch**. Second, updates to the sketch should be processed quickly.

Since we are sublinear in input space, we will have to approximate almost any function we want to compute over $\vec{a}$, but we still want to specify an approximation parameter ε and bound the probability of error by δ.

The **count-min sketch** addresses these requirements and allows us to calculate several characteristics of a stream using a single type of data structure. It can be used to approximate three types of **queries**:

1. **point query:** $Q_i(i)$ is an approximation of the vector element $a_i(t)$
2. **range query:** $Q_i(l, r)$ is an approximation of $\sum_{i=l}^{r} a_i$.
3. **inner product query:** $Q_i(\vec{a}, \vec{b})$ approximates $\vec{a} \odot \vec{b} = \sum_{i=1}^{n} a_i b_i$

These queries have a variety of applications: point and range queries are used in summarizing the data distribution and the inner product query can be used to estimate the join size of relations.

In addition, these queries can be used for more complex data stream functions:

1. **ϕ-quantiles:** The ϕ-quantiles of the cardinality

$$||\vec{a}||_1 = \sum_{i=1}^{n} |a_i(t)| \tag{3}$$

   are the values within a multiset of finite size that split it into ϕ groups of equal size based on their rank R. For example, the median of a set of data is the 2-quantile.

   To obtain the ϕ-quantiles, the elements are sorted and split as follows. The kth quantile is the element(s) with rank $R = k\phi||\vec{a}||_1$ for $k = 0, \ldots, 1/\phi$. The ε-approximation for the ϕ-quantiles accepts as the k-th quantile any of value with rank $(k\phi - \varepsilon)||\vec{a}||_1 \leq R \leq (k\phi + \varepsilon)||\vec{a}||_1$ for a given $\varepsilon < \phi$.

2. **Heavy hitters:**

2 State of the art

Sketches have been commonly used to help analyze large streams of data. These sketches allow for the following types of queries: L_1 and L_2 norms, number of distinct items in a sequence, join size of relations, range sum queries, and more. Although these data structures have proved to be very powerful, there are a few key drawbacks that limit the effectiveness of these sketches. A few of the drawbacks are as follows:

1. Common sketches typically have a $\Omega(\frac{1}{\epsilon^2})$ multiplicative factor. For common uses of ϵ, like 0.1 or 0.01, this scaling can quickly become too expensive.
2. Many sketches take linear time (in the size of the sketch) to do a single update.
3. Many sketches require p-wise independent hash functions. This is not trivial, and is especially difficult in hardware applications.
4. Some sketches can only handle one particular query.
5. Many sketches use analysis that hides large constants.

The count-min sketch reduces these problems in the following ways:

1. CMS uses space proportional to $\frac{1}{\epsilon}$
2. Update time is sublinear with respect to the size

3. CMS requires only pairwise independent hash functions
4. The sketch can answer several queries and has numerous applications
5. All constants are explicit and small.

In particular, notice that the space has been reduce from $\frac{1}{\epsilon^2}$ to $\frac{1}{\epsilon}$ using a CMS, and the time for an update has been reduced from $\frac{1}{\epsilon^2}$ to 1.

3 How to make a count-min sketch

Note e is the base of the natural logarithm function, ln. The count-min sketch is an array with w columns and d rows where:

$$w = \lceil e/\varepsilon \rceil \tag{4}$$

$$d = \lceil \ln 1/\delta \rceil \tag{5}$$

for a given accuracy parameter ε and probability guarantee δ. We need d pair-wise independent hash functions

$$h_1 \dots h_d : \{1 \dots n\} \to \{1 \dots w\} \tag{6}$$

The sketch starts with every array entry being 0. The count-min sketch *count* is updated when the tth update $\{a_{i_t}, c_t\}$ arrives as follows

$$count[j, h_j(i_t)] \leftarrow count[j, h_j(i_t)] + c_t \tag{7}$$

In other words, upon receiving an update (i_t, c_t):

```
Update_CM_Sketch(count, $(i_t, c_t)$)
    for each row j, 1 $\leq$ j $\leq$ d:
        count[j, $h_j(i)$] $\leftarrow$ count[j, $h_j(i)$] + $c_t$
    return count
```

4 Approximate Point Queries using CM sketches

4.1 Non-Negative Case

4.1.1 Algorithm

For the non-negative case (either the non-negative turnstile or cash-register case), the algorithm to estimate $\hat{a}_i(t) = Q_i(i)$ is as follows:

$$Q_i(i) = \hat{a}_i = \min_j count[j, h_j(i)] \tag{8}$$

In other words for a given $\hat{a}_i(t) =$ `Estimate_Point_Query(count,` i_t`)`.

```
Estimate_Point_Query(count, $i_t$)
    s ← ∞
    for each row j, 1 ≤ j ≤ d:
        if count[j, $h_j(i)$] < s:
            s ← count[j, $h_j$(i)]
    return s;
```

Theorem 1: The estimate $\hat{a}_i$ has the following guarantees:

(1) $a_i \le \hat{a}_i$

(2) with probability at least $1 - \delta$,

$$\hat{a}_i \le a_i + \varepsilon ||\vec{a}||_1 \tag{9}$$

4.1.2 Proof of Theorem 1

First, we prove 1.a.

Define an indicator variable $I_{i,j,k}$ as follows:

$$I_{i,j,k} = \begin{cases} 1, & \text{if } h_j(i) = h_j(k) \text{ and } i \neq k \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

$I_{i,j,k}$ is 1 if i and k, $i \neq k$, have a collision under hash function h_j.

Since, to begin with, we chose all h_j from a 2-universal hash family with range $1 \le h_j \le d = \lceil \frac{e}{\varepsilon} \rceil$ we know that, for $i \neq k$,

$$Pr(h_j(i) = h_j(k)) \le \frac{1}{\text{range}(h_j)} = 1/\lceil \frac{e}{\varepsilon} \rceil = \lceil \frac{\varepsilon}{e} \rceil \tag{11}$$

by the properties of a 2-universal has function and by our choice of hash function range.

Define $X_{i,j}$ as follows:

$$X_{i,j} = \sum_{k=1}^{n} I_{i,j,k} a_k \tag{12}$$

In other words, $X_{i,j}$ is the sum of all corresponding vector values whose indices collide with the index of a_i under h_j. Trivially, we can then state that

$$count[j, h_j(i)] = a_i + X_{i,j} \tag{13}$$

This is clearly true as the column that i hashes to under h_j contains the real value a_i after update t, combined with all of the other a_k, $k \neq i$ after update t that hash to that same spot in that row j under h_j. Now, Theorem 1.a follows because we are only considering the case where vector elements are non-negative and thus

$$a_i \le \hat{a}_i = a_i + X_{i,j} \tag{14}$$

for any possible $X_{i,j}$

Now we show 1.b.

For future use in the analysis, we evaluate:

$$E(X_{i,j}) = E\left(\sum_{k=1}^{n} I_{i,j,k} a_k\right) \tag{15}$$

by definition of $X_{i,j}$. The linearity of expectation states that

$$E(\sum_{i=1}^{n} c_i X_i) = \sum_{i=1}^{n} c_i E(X_i) \tag{16}$$

where X_is may be dependent. Applying linearity of expectations here, we have:

$$E(\sum_{k=1}^{n} a_k I_{i,j,k}) = \sum_{k=1}^{n} a_k E(I_{i,j,k}) \tag{17}$$

We calculated $E(I_{i,j,k}) \leq \lceil\frac{\varepsilon}{e}\rceil$ above, so we have

$$\sum_{k=1}^{n} a_k E(I_{i,j,k}) \leq \sum_{k=1}^{n} a_k \lceil\frac{\varepsilon}{e}\rceil \tag{18}$$

and finally, since $||\vec{a}||_1 = \sum_{k=1}^{n} |a_k| = \sum_{k=1}^{n} a_k$ (all a_k are assumed to be non-negative)

$$\sum_{k=1}^{n} a_k \lceil\frac{\varepsilon}{e}\rceil = \lceil\frac{\varepsilon}{e}\rceil||\vec{a}|||_1$$

$$\implies E(X_{i,j}) \leq \lceil\frac{\varepsilon}{e}\rceil||\vec{a}||_1 \tag{19}$$

We will use this result in the following proof.

We want to show that $Pr[\hat{a}_i > a_i + \varepsilon||a||_1] < \delta$. We can see that

$$Pr[\hat{a}_i > a_i + \varepsilon||a||_1] = Pr[\forall_j.count[j, h_j(i)] > a_i + \varepsilon||\vec{a}||_1 \tag{20}$$

since the probability that all values in a set are greater than $a_i + \varepsilon||\vec{a}||_1$ is equivalent to the probability that the minimum of all the values in that set is greater than $a_i + \varepsilon||\vec{a}||_1$. (Recall that $\hat{a}_i$ is defined as the minimum of all of the array elements that i hashes to in each row.) Then, by the definition stated in equation 13, we have

$$Pr[\forall_j.count[j, h_j(i)] > a_i + \varepsilon||\vec{a}||_1] = Pr[\forall_j.a_i + X_{i,j} > a_i + \varepsilon||\vec{a}||_1] \tag{21}$$

Then, subtracting a_i from both sides, and using equation 19 on the $\varepsilon||\vec{a}||_1$ term, we have:

$$Pr[\forall_j.a_i + X_{i,j} > a_i + \varepsilon||\vec{a}||_1] \geq Pr[\forall_j.X_{i,j} > eE(X_{i,j})] \tag{22}$$

Finally, consider the Markov inequality:

$$Pr[X > aE[X]] \leq \frac{1}{a} \tag{23}$$

Since this is the non-negative case and thus $X_{i,j}$ is non-negative, we can apply it here:

$$Pr[X_{i,j} > eE(X_{i,j})] \leq \frac{1}{e} \tag{24}$$

$$\implies \bigcap_{1\leq j\leq d} Pr[X_{i,j} > eE(X_{i,j})] \leq \left(\frac{1}{e}\right)^d \tag{25}$$

$$\implies Pr[\forall_j . X_{i,j} > eE(X_{i,j})] < e^{-d} < \delta \tag{26}$$

This estimate is calculated in $O(\ln \frac{1}{\delta})$ as the minimum of a multiset can be taken in linear time, and we are taking it across $d = O(\ln \frac{1}{\delta})$ rows for the point query. The update time to maintain the data structure after each update is also $O(\ln \frac{1}{\delta})$, since we hash a single value in each row of the data structure.

The space complexity is $\left(2 + \lceil \frac{e}{\varepsilon} \rceil\right) \lceil \ln \frac{1}{\delta} \rceil$ words, as the dimensions of the matrix are $w \times d = \lceil \frac{e}{\varepsilon} \rceil \times \lceil \ln \frac{1}{\delta} \rceil$.

4.2 General Case

4.2.1 Algorithm

The algorithm is identical to the non-negative case, except that we take the median of the multiset:

$$Q_i(i) = \hat{a}_i = \text{median}_j count[j, h_j(i)] \tag{27}$$

Theorem 2: With probability $1 - \delta^{1/4}$,

$$a_i - 3\varepsilon||\vec{a}||_1 \leq \hat{a}_i \leq a_i + 3\varepsilon||\vec{a}||_1. \tag{28}$$

Here, since the array elements can be negative, we can no longer use Markov, and instead must apply Chernoff bounds. TODO more details

The time and space complexity are the same as the non-negative case: $O(\ln \frac{1}{\delta})$ time and $(2 + \frac{e}{\varepsilon}) \ln \frac{1}{\delta}$ words, respectively.

5 Range query:

A range query, denoted as: $\mathcal{Q}(l, r)$, requests the value:

$$a[l, r] = \Sigma_{i=l}^{r} a_i$$

And we denote our estimator based on the CM sketch as: $\hat{a}[l, r]$.

Procedure for estimation:
Keep $log_2 n$ CM sketches. A single range query can be converted into $2log_2 n$ dyadic range queries, and each dyadic range query can be converted into a single point query. A CM sketch is kept for every dyadic range, and we do a single point query for every dyadic range. The sum of all of the point queries is the result, $\hat{a}[l, r]$.

k TODO better explain dyadic range stuff? ASK LIN

Theorem 4: $a[l, r] \leq \hat{a}[l, r]$. With probability at least $(1 - \delta)$, $\hat{a}[l, r] \leq a[l, r] + 2\epsilon logn||\boldsymbol{a}||_1$.

Proof:
TODO explain why we have 2log2n dyadic ranges, which are equivalently a point query for each. Then:
This proof is a simple extension of the point query proof. Because a single range query can be transformed to $2\log_2 n$ point queries, we can extend the error bounds of many point queries. Recall that a single point query errors by at most $\epsilon||\boldsymbol{a}||_1$, with probability δ. Because each of these point queries that we used are taken from a separate CM sketch, they are all independent. Thus, we can simply add the errors together, and therefore, we error by at most $2\log_2 n(\epsilon||\boldsymbol{a}||_1)$, and with the same probability, δ.

6 Applications

6.1 Quantiles in the turnstile model

The count-min sketch can be used to calculate the ϕ-quantiles in the turnstile model. In this case, updates where $c_t < 0$ are called deletions, and $c_t > 0$ are called insertions. The value for each vector element a_i is actually the number of instances of the elements i.

Prior work (21) shows that ϕ-quantiles in this model can be approximated using range sums. This is done as follows:

1. For each $k \in 1, 2, \ldots, 1/\phi$
   (a) Find a range sum such that $a[1, r] = k\phi||\vec{a}||_1$. Find r using a binary search on the possible range sums $a[1, \hat{r}]$, where $\hat{r} \in \{1, 2, \ldots, n\}$. r is the ε-approximation for the kth ϕ-quantile.

The method in [21] uses Random Subset Sums to approximate range sums. If instead a count-min sketch is used to approximate the range sums, better results follow.

To use a count-min sketch to approximate quantiles in the turnstile model, $\log n$ sketches are kept, one for each dyadic range. Each sketch gets accuracy parameter $\varepsilon/\log n$ for an overall accuracy bounded by ε. Each sketch gets probability guarantee $\delta\phi/\log n$, for an overall probability guarantee of δ for all $1/\phi$ quantiles. Then, we have the following:

Theorem 5: ε-approximate ϕ-quantiles can be found with probability at least $1-\delta$ by keeping a data structure with space $O\left(\frac{1}{\varepsilon}\log^2(n)\log\left(\frac{\log n}{\phi\delta}\right)\right)$ The time for each insert or delete operation is $O\left(\log(n)\log\left(\frac{\log n}{\phi\delta}\right)\right)$, and the time to find each quantile on demand is $O\left(\log(n)\log\left(\frac{\log n}{\phi\delta}\right)\right)$

This improves the existing query time and update by a factor of more than $\frac{34}{\varepsilon^2}\log n$ The space requirements are improved by a factor of at least $\frac{34}{\varepsilon}$.

6.2 Heavy hitters for the cash register case

Recall that the cash register case is for positive updates only. We can easily obtain $||\boldsymbol{a}||$ at any given point in time because it is simply: $\Sigma_{i=1}^{t} c_i$.We define a ϕ heavy hitter if the estimation of the point query, $Q(i_t) = \hat{a}_{it} \geq \phi||\boldsymbol{a}(t))||$. We can do this by maintaining a heap to store the items above the $phi||\boldsymbol{a}(t)||$ threshold. On any update, we check the lowest value in the heap, and if the update would be greater than the lowest item, we replace it in the heap. When we are finished with the stream, we do a final scan of all items in the heap, and return the ones which have a value over $\phi||\boldsymbol{a}||_1$.

Theorem 6: We can identify the heavy hitters of a sequence of length $||\boldsymbol{a_i}||_1$ using space $O(\frac{1}{\epsilon}log(\frac{||\boldsymbol{a}||}{\delta}))$ and time $O(log(\frac{||\boldsymbol{a}||}{\delta}))$. Every item which occurs with count $\phi||\boldsymbol{a}||_1$ is output, and with probability $(1-\delta)$, no items whose count is less than $(\phi-\epsilon)||\boldsymbol{a}||_1$ is output.

Proof idea:
Because we have only positive updates, and a CM sketch will only over estimate, it is not possible to miss any heavy hitter. Thus, we will never omit any heavy hitters. The bound on the error of outputting a non heavy hitter comes from the fact that a single point query outputs the estimate count of an item within ϵ of the actual value, with probabiltiy $(1-\delta)$. Because the heavy hitter relies purely on the point query, the error bound is the same, and thus, is as above.