Abstract

In this report a potential solution to the Count Tracking problem will be outlined. Specifically it will explore the Count-Min Sketch, a modern data structure that counts the approximate frequencies of items in sub-linear space. It will analyze the error margin and the probability of getting this error margin.

Introduction

Algorithmic problems of tracking the contents of a set arise frequently in all kinds of software. Choosing the appropriate data structure for the specified task of each program is crucial in building efficient and effective software. The main objective of data structures were to maintain set S under any update operation and quickly respond to queries. They tried to use the least amount of space possible (usually linear) while maintaining perfect copy of S.

However, modern applications now face the problem of dealing with massive data sets. These problems calls for the need of new types of data structures. These data structures need to be relatively small (at least sub-linear in size of input) but can provide approximate responses to various queries as applications can work without precise answers. They want to end up being a sketch of the entire data set hence the term “Sketches.”

Count Tracking Problem

Given a large number of items estimate frequency of each item which changes with regard to time. The crucial part of the Count Tracking Problem is that N the total frequency of all items is incredibly large as well as the number of items. Another is that queries can tolerate some imprecision. So a data structure for the problem must trade off precision for space. The two main methods for any data structure for Count Tracking is update(i,c) which updates the frequency of item i by count c and estimate(i) which returns an approximate frequency for item i.

The problem with using a normal data structure such as a hash table is that the amount of memory used increases as more items are added to the table. Since the size is linear, access to the structure will be slow as it will have to reside in slow memory or
virtual memory and due to the steady increase in the size of the table it needs to be resized periodically.

**The Count-Min Sketch**

The Count-Min Sketch is unique in the fact that it only requires a fixed amount of space for however many items. Which means that it does not increase in size over time so there is no need to resize it periodically. Let $N$ be the sum of all counts. Any distortion caused by over counting is a very small fraction of $N$. The fraction is controlled by the parameters of the Count-Min Sketch, the smaller the fraction the larger the sketch.

The core of the Count-Min Sketch is a 2D array that is of width $w$ and depth $d$. It is also initialized with a prime number $p$ that is greater than the largest $i$. Each row has an associated hash function usually of the form $(ai + b \% p) \% w$ where $a, b$ are integers between 1 and $p - 1$. Each row must have a different pair of $a, b$. The array is initialized to all 0 values. $N$ is also 0 at the start.

The update($i, c$) method is rather simple. For each $j$ from 0 to $d - 1$ apply corresponding hash function to $i$ and add $c$ to corresponding column (the result of the hash function). It is easy to see that there are $d$ many updates per a single update($i, c$) call. The estimate($i$) method is also simple and very similar to the update($i, c$) method. For each row use the hash function to find the correct column and return the lowest found frequency.

Since $w$ is much smaller than $I$ (the total set of items) there will be hash collisions in each row. Since the hash functions spread the items uniformly around the row, we expect that a uniform fraction of items will collide with any given $i$. The expected fraction of weight colliding with $i$ is $O(\frac{N}{w})$. There will be some cases where we will be lucky and get less and some cases where we will be unlucky and get more. The colliding weight is unlikely to be much larger than the expected amount: the probability of seeing more than twice the expected amount is at most $\frac{1}{2}$ according to Markov’s Inequality (where $t=2$) So the value of the counter in a row is at $O(\frac{2N}{w})$ more than the actual frequency of $i$ with probability $\frac{1}{2}$.

Since we take the minimum of the counter in each row for us to get an error of more than $\frac{2N}{w}$ every row must have that much of an error. That would occur with probability $(\frac{1}{2})^d$. Hence we can see that the Count-Min Sketch has error of at $O(\frac{2N}{w})$ with probability of $(\frac{1}{2})^d$.

With this analysis we can find values for $w, d$ that satisfy an error margin of at most 0.1% of $N$ and with certainty 99.9%. $\frac{2}{w} = 0.1, w = 2000$ and $(\frac{1}{2})^d = 0.001, d \approx 10$. With that using 32bit counters, the space required by the array is $w \times d \times 4 = 80KB$. 


Conclusion

We saw an overview of the Count Tracking Problem and why older data structures are unsuited in dealing with the problem. We also went through the Count-Min Sketch starting from the initialization and its methods to approximation analysis. In the end we managed to find sufficient values for $w$ and $d$ that we can be assured 99.9% certainty with an error margin of only 0.1%. With these values we found the space required for such an array.

References


