Probabilistic Counting, Adding, and Dividing

rikhav.shah@berkeley.edu

October 2020

1 Introduction

The problem of approximate counting requires a data structure D which supports the following methods:

```
D.init():
    initialize n to 0

D.update():
    update n ← n + 1

D.query():
    return an estimate n
```

It is not difficult to show that if we require D.query() $= n$ exactly, then D will need $\log n$ bits. Traditionally, this problem has been studied in the setting where D.query() $= n$ is required to have fixed ϵ relative error with fixed probability $1-\delta$. The optimal data structure in that setting is the Morris counter, and is equivalent to the following: let $a = O(\epsilon^2/\log(1/\delta))$ and define the sequence

$$s_i = \begin{cases} i & i \leq 1/a \\ (1+a)s_{i-1} & \text{o.w.} \end{cases}.$$

Then the data structure methods are implemented as

```
D.init():
    initialize i to 0

D.update():
    update i ← i + 1 with probability (s_{i+1} − s_i)^{−1}

D.query():
    return s_i
```

The number of bits needed by D at any point in time is simply $\log i$, which in this case is the minimum of $\log n$ and $\log\log n + \log(1/a)$. It can be shown that this the optimal number of bits up to a constant factor.

We extend this counter in three ways. First, we provide a generic way to construct a counter satisfying whatever kind of error bounds the user desires, not just fixed relative error. Second, a user may want to increase the counter by a large number all at once rather than repeatedly calling the update function; the structure should support this in $O(1)$ time. Third, there is a limited sense in which decrements will be supported.

We tackle the first challenge first. Our data structure closely resembles the Morris counter; the only change is in the choice of s_i. We call the structure a 'probabilistic counter with range s_i'.

Intuitively, if s_i is fast growing, then s_i can reach n even for small i, so the bit requirement is small. On the other hand, the variance in number of updates before the data structure increments from i to $i+1$ is $(s_{i+1}-s_i)^2$, which is large when s_i is fast growing, so the variance of D.query() will be larger.

We will show for arbitrary s_i that D.query() is an unbiased estimator of n, and we will furthermore give conditions on s_i under which D.query() is close to correct with high probability.

2 Preliminaries

We will make use of the following tail bound.

Lemma 1. *Let X be the sum of independent (but not necessarily identically distributed) geometric random variables with finite means. Let m be the largest of those means. Let $\mathbb{E}[X]=\mu$. Then for any $\epsilon \geq 0$ we have*

$$\Pr\left[X \leq \lambda\mu\right] \leq \exp\left(-\frac{\mu}{m}\left(\lambda - 1 - \log\lambda\right)\right).$$

for $\lambda \leq 1$ and

$$\Pr\left[X \geq \lambda\mu\right] \leq \exp\left(-\frac{\mu}{m}\left(\lambda - 1 - \log\lambda\right)\right).$$

for $\lambda \geq 1$.

Proof. Combine theorems 2.1 and 3.1 here `http://www2.math.uu.se/~svante/papers/sj328.pdf`. □

The following observations will make it easier to get a handle on the exponent $\frac{\mu}{m}(\lambda - 1 - \log\lambda)$ in the context of this problem.

Observation 2. *For constants $C_1 < C_2$, the functions f_1, f_2 given by*

$$f_1(x) = (C_2 - x)\left(\frac{C_1}{C_2 - x} - 1 - \log\left(\frac{C_1}{C_2 - x}\right)\right),$$

$$f_2(x) = (C_1 + x)\left(\frac{C_2}{C_1 + x} - 1 - \log\left(\frac{C_2}{C_1 + x}\right)\right)$$

are strictly decreasing on $[0, C_2 - C_1)$ and lower bounded by the linear functions

$$f_1(x) \geq -\log(C_2/C_1)x + (C_1 - C_2 + C_2\log(C_2/C_1)),$$

$$f_2(x) \geq -\log(C_2/C_1)x + (C_2 - C_1 + C_1\log(C_2/C_1)),$$

Proof. The first two derivatives of f_1, f_2 are

$$f_1'(x) = -\log\left(\frac{C_2 - x}{C_1}\right) \quad \text{and} \quad f_1''(x) = \frac{1}{C_2 - x},$$
$$f_2'(x) = -\log\left(\frac{C_2}{C_1 + x}\right) \quad \text{and} \quad f_2''(x) = \frac{1}{C_1 + x},$$

so f_1, f_2 are convex and strictly decreasing for $x < C_2 - C_1$. So, f_1, f_2 are lower bounded by the linear functions tangent to them at 0. $\square$

The following definition will help in the application of Lemma 1.

Definition 1. *Given an increasing sequence s_i of numbers, define their max-gap to be the monotone function*

$$g(n) = \max(\max_{s_i \le n} s_i - s_{i-1}, \min_{s_i \le n} n - s_i).$$

3 Probabilistic Counter

In the following statements, let i_n be the value of i in an instance of D after n updates. Note that i_n is a random variable and that s_{i_n} is the result of calling D.query() at that point in time.

Proposition 3 (Correct expectation)**.**

$$\mathbb{E}[s_{i_n}] = n.$$

Proof. Note that $s_{i_0} = s_0 = 0$ exactly. Inductively,

$$\begin{aligned}
\mathbb{E}[s_{i_n}] &= \sum_{j=0}^{\infty} \Pr[i_{n-1} = j]\, \mathbb{E}[s_{i_n} \mid i_{n-1} = j] \\
&= \sum_{j=0}^{\infty} \Pr[i_{n-1} = j] \left(\left(1 - \frac{1}{s_{j+1} - s_j}\right) s_j + \frac{1}{s_{j+1} - s_j} s_{j+1}\right) \\
&= \sum_{j=0}^{\infty} \Pr[i_{n-1} = j]\, (s_j + 1) \\
&= \mathbb{E}[s_{i_{n-1}}] + 1.
\end{aligned}$$

Thus $\mathbb{E}[s_{i_n}] = n$ as desired. $\square$

Theorem 4 (Concentration)**.** *Given a natural number n and an arbitrary interval $[n_{\text{lo}}, n_{\text{hi}}]$ containing n, if the max-gap of s_i satisfies*

$$g(n_{\text{hi}}) < n \min\left(\frac{(n_{\text{hi}}/n)\log(n_{\text{hi}}/n) - n_{\text{hi}}/n + 1}{\log(2/\delta) + \log(n_{\text{hi}}/n)},\, n_{\text{hi}}/n - 1\right)$$

and

$$g(n) < n \min\left(\frac{(n_{\text{lo}}/n)\log(n_{\text{lo}}/n) - n_{\text{lo}}/n + 1}{\log(2/\delta) + \log(n/n_{\text{lo}})},\, 1 - n_{\text{lo}}/n\right)$$

then $s_{i_n} \in [n_{\text{lo}}, n_{\text{hi}}]$ with probability $1 - \delta$.

Proof. Since $g(n_{\text{hi}}) < n_{\text{hi}} - n$ and $g(n) < n - n_{\text{lo}}$, we are guaranteed for some i, j that $s_i \in [n_{\text{lo}}, n)$ and $s_j \in (n, n_{\text{hi}}]$. Let i_{hi} be the largest index such that $s_{i_{\text{hi}}} \le n_{\text{hi}}$ and i_{lo} the smallest index such that $n_{\text{lo}} \le s_{i_{\text{lo}}}$. We have

$$n_{\text{lo}} \le s_{i_{\text{lo}}} \le n_{\text{lo}} + g(n) < n < n_{\text{hi}} - g(n_{\text{hi}}) \le s_{i_{\text{hi}}} \le n_{\text{hi}}.$$

Let $N(i)$ be the smallest n' for which $i_{n'} \ge i$. Note the event $s_{i_n} \ge s_i$ is equivalent to the event $N(i) \le n$. Also note that $N(i)$ is a sum of geometric random variables. Specifically, $N(i+1) - N(i)$ is the number of updates before the counter increments from i to $i+1$, so is a geometric random variable with mean $s_{i+1} - s_i$. By linearity of expectation, we have $\mathbb{E}[N(i)] = s_i$. The largest mean of any of the geometric random variables comprising $N(i_{\text{hi}})$ is $g(s_{i_{\text{hi}}})$, which is upper bounded by $g(n_{\text{hi}})$. Similarly the largest mean of any of the geometric random variables comprising $N(i_{\text{lo}})$ is $g(s_{i_{\text{lo}}})$, which is upper bounded by $g(n)$. Lemma 1 allows us to conclude $N(i)$ is highly concentrated. We apply the bound for the upper and lower tails separately using Lemma 1 for each tail, and using Observation 2 twice for each tail.

$$\begin{aligned}
\Pr[s_{i_n} \ge n_{\text{hi}}] &\le \Pr[s_{i_n} \ge s_{i_{\text{hi}}}] \\
&= \Pr[N(i_{\text{hi}}) \le n] \\
&= \Pr[N(i_{\text{hi}}) \le \frac{n}{s_{i_{\text{hi}}}} s_{i_{\text{hi}}}] \\
&\le \exp\left(-\frac{s_{i_{\text{hi}}}}{g(n_{\text{hi}})}\left(\frac{n}{s_{i_{\text{hi}}}} - 1 - \log\frac{n}{s_{i_{\text{hi}}}}\right)\right) \\
&\le \exp\left(-\frac{n_{\text{hi}} - g(n_{\text{hi}})}{g(n_{\text{hi}})}\left(\frac{n}{n_{\text{hi}} - g(n_{\text{hi}})} - 1 - \log\frac{n}{n_{\text{hi}} - g(n_{\text{hi}})}\right)\right) \\
&\le \exp\left(\log(n_{\text{hi}}/n) - \frac{(n_{\text{hi}}/n)\log(n_{\text{hi}}/n) - n_{\text{hi}}/n + 1}{g(n_{\text{hi}})} n\right) \\
&\le \delta/2
\end{aligned}$$

$$\begin{aligned}
\Pr[s_{i_n} \le n_{\text{lo}}] &\le \Pr[s_{i_n} \le s_{i_{\text{lo}}}] \\
&= \Pr[N(i_{\text{lo}}) \ge n] \\
&= \Pr[N(i_{\text{lo}}) \ge \frac{n}{s_{i_{\text{lo}}}} s_{i_{\text{lo}}}] \\
&\le \exp\left(-\frac{s_{i_{\text{lo}}}}{g(n)}\left(\frac{n}{s_{i_{\text{lo}}}} - 1 - \log\frac{n}{s_{i_{\text{lo}}}}\right)\right) \\
&\le \exp\left(-\frac{n_{\text{lo}} + g(n)}{g(n)}\left(\frac{n}{n_{\text{lo}} + g(n)} - 1 - \log\frac{n}{n_{\text{lo}} + g(n)}\right)\right) \\
&\le \exp\left(\log(n/n_{\text{lo}}) - \frac{(n_{\text{lo}}/n)\log(n_{\text{lo}}/n) - n_{\text{lo}}/n + 1}{g(n)} n\right) \\
&\le \delta/2.
\end{aligned}$$

By union bound, the probability s_{i_n} misses $[n_{\text{hi}}, n_{\text{lo}}]$ is thus at most δ, as desired. □

Corollary 5. *For small ϵ, the standard Morris counter with $a = \frac{\epsilon^2/\log(2/\delta)}{2+\epsilon}$ satisfies for every n that $|s_{i_n} - n| \le \epsilon n$ with probability $1 - \delta$.*

Proof. Note that D is deterministically correct for $n \leq 1/a$, and s_i satisfies the hypothesis of Theorem 4 with $n_{\text{lo}} = (1-\epsilon)n$ and $n_{\text{hi}} = (1+\epsilon)n$ for every $n \geq 1/a$. $\square$

Corollary 6. *The probabilistic counter implemented by Redis, for particular parameters, satisfies for every* n *that* $|s_{i_n} - n| \leq cn^{0.75}$ *with probability* $1-\delta$*. It uses* $(0.5+o(1))(\log n + \log\log(1/\delta) - \log c)$ *bits.*

Proof. Redis takes
$$s_i = \frac{a}{2}(i^2+i),$$
which gives a max-gap of
$$g(n) = O(\sqrt{2an}).$$
We can therefore take $n_{\text{lo}} = n(1-cn^{-0.25})$ and $n_{\text{hi}} = n(1+cn^{-0.25})$, and set the parameter $a = c^4/(8\log^2(2/\delta))$. These choices satisfies the hypothesis of Theorem 4. Note that taking logs of $i \approx \sqrt{2n/a}$ gives the bit requirement. $\square$

Remark 7. *Experimentally, the bounds in the above two corollaries appear to be tight up to a constant factor in the sense that if theory guarantees* $|s_{i_n}/n - 1| < \epsilon$ *with probability* $1-\delta$*, then for some* C *we will have* $|s_{i_n}/n-1| > C\epsilon$ *with probability at least* δ *in practice. With a very limited memory budget of* 8 *bits, Morris provides decent error on a huge range of* n *(roughly from* $n = 2^{10}$ *to* $n = 2^{23}$*) at the expense of worse 'peak' performance on a narrower range of* n*. Redis, for carefully selected parameter* a*, gives about twice as good relative error as Morris for a small range of* n*, (like* 2^k *to* 2^{k+2}*), and absolutely atrocious error elsewhere.*

4 Beyond counting

Say one wants to support large increments. One possibility is on increment by U, to simulate U steps of the standard probabilistic counter. However, this will take $O(U)$ time, which is huge. We can support $O(1)$ time.

Let D be the probabilistic counter with range s_i that satisfies the bound $s_{i_n} \in [n-l(n), n+u(n)]$ with probability $1-\delta$ for some monotonically increasing functions l, u. We construct D' supporting large updates that satisfies the same bound, where now n is the total value of the sum, not the number of increments. We require that the bound is satisfied for any sequence of updates summing to n, for any n.

D'`.update(U):`

  `Pick` j `to be the largest index such that` $s_j - s_i \leq U$.

  `Let` $U_1 = s_j - s_i$ `and` $U_2 = U - U_1$. `Update` $i \leftarrow j$.

  `If` $U_2 = 0$`, end.`

  `Else sample a geometric random variable` r `with parameter` $(s_{i+1} - s_i)^{-1}$.

    `If` $r > U_2$ `end.`

    `Else update` $i \leftarrow i+1$ `and call` D'`.update(`$U_2 - r$`).`

Theorem 8. *The above update method satisfies the desired bound.*

Proof. Say the current state of the data structure is i. Say we are given an update U which happens to be $U = s_j - s_i$ for some $j \geq i$. Then since $s_i \in [n - l(n), n + u(n)]$, we automatically have $s_j \in [n+U-l(n), n+U+u(n)] \subset [n+U-l(n+U), n+U+u(n+U)]$, so we can simply increase i to j. For $U \leq s_{i+1} - s_i$, we sample a geometric random variable r with mean $s_{i+1} - s_i$. The sequence of r-s sampled in each recursive call mimic the $N(i+1) - N(i)$ in the proof of Theorem 4, so correctness follows from the correctness of D. □

An alternative formulation of this problem is this: given a stream of positive numbers, approximate their sum. The standard solution to that problem is to use floats. An float f is represented as three non-negative integers $b < 2^1, e < 2^8, F < 2^{23}$ with

$$f = (-1)^b \cdot 2^{e-127} \cdot \left(1 + \frac{F}{2^{23}}\right).$$

Floats satisfy the bound that if $a + b = c$ exactly, then $a + b$ according to float arithmetic will be c rounded to the nearest float, which is at most a factor of 1 ± 2^{-23} off of the true value. As a consequence, the data structure that simply adds a stream of floats to an accumulating sum can have relative error as high as $T/2^{23}$ where T is the number of items in the steam. Our probabilistic counter can remove the dependence on T. Note that

$$2^{127+23} f = 2^{23+e} + 2^e F$$

is an integer ranging from 2^{23} (for $e = F = 0$) to $2^{23+255} + 2^{255}(2^{23} - 1) = 2^{279} - 2^{255}$ (for $e = 2^8 - 1$ and $F = 2^{23} - 1$). We thus set

$$s_i = \begin{cases} i & i \leq 2^{23} \\ 2^{150} f_{i-2^{23}} & \text{o.w.} \end{cases}$$

where f_i is the ith smallest positive float. The max-gap satisfies

$$g(n) \leq \max(1, n/2^{23}).$$

Our structure can scale inputs by 2^{150}, perform an update, then scale the output of query by 2^{-150}. As a consequence, the probabilistic counter with range s_i will have $< 0.1\%$ error with probability $> 99.9\%$. This will be true regardless of n or T. Contrast this with standard floating-point arithmetic. There, the accuracy degrades linearly with the number of operations performed. Machine precision is 2^{-23} for floats, so after roughly 2^{23} additions, the error may be huge.

5 Application to Caching

If one wants to implement a least-frequently-used (LFU) caching policy for a particular database, then one needs to estimate how frequently each item in the database is requested. We can use a probabilistic counter to do this with very little extra memory. We consider two models of requests. In the first model, we are given a stream of random requests, and the probability that each request is for a particular item obeys a fixed power-law distribution. In the second model, requests arrive as a Poisson point process whose rate may change over time. In both cases, one needs to figure out the distribution on the item that the next request will be for; the first model serves as kind of a warm-up, and the second model seems more realistic in practice.

5.1 Model one

Say we have U items in a database and the probability of querying the xth item is proportional to $1/x$. An ideal cache of size $k+1$ stores first k, giving a cache hit rate of approximately $\log(k)/\log(U)$. In practice, the probabilities may not be known in advance. So we must estimate them using empirical frequency. If the system correctly identifies the top $k^{0.9}$ items and keeps them in cache, then the hit rate will be at least $0.9\log(k)/\log(U)$. So the system need to differentiate between items that occur with probability more than $1/k^{0.9}$ versus less than $1/k$. For this task, a probabilistic counter can tolerate huge ($k^{0.1}$) relative error and still be successful. If one lets $s_i = (\log k)^i$, then one only needs $\log\log n - \log\log\log k$ bits.

Unfortunately, it's unclear if this model is very realistic. In reality, the probability of querying each item can change over time.

5.2 Model two

Requests for a particular item come in as a Poisson point process with rate λ, which can depend on t, time. If one makes some assumption about how quickly λ can change, then it may be reasonable to estimate λ based on the the number of requests made in the last T seconds, for some parameter T. One can do the following with a deterministic counter:

1. Count the requests made in the first $T+1$ seconds.
2. Scale that number down by $\frac{T}{T+1}$
3. Add the number of requests made in the next second and go to step 2.

If x_t is the value after step 2 after t seconds, then we have the recurrence

$$x_{t+1} = \frac{T}{T+1}(x_t + b_t)$$

where b_t, the number of items observed during $[t, t+1)$, is a Poisson random variable with rate λ. Rearranging gives

$$x_{t+1} - \lambda T = \frac{T}{T+1}(x_t - \lambda T) + (b_t - \lambda).$$

Note that $b_t - \lambda$ has mean zero and variance λ, so with high probability $|x_t/T - \lambda|$ should exponentially quickly decrease to below $O(\sqrt{\lambda}/T)$.

Translating each of these steps to their probabilistic versions, steps 1 and 3 are fine. What about 2? If we use a Morris counter and it happens to be that $(T+1)/T = (1+a)^d$ for some d, then one can decrement $i \leftarrow i - d$. Then $s_i \in [(1-\epsilon)n, (1+\epsilon)n] \implies s_{i-d} \in [(1-\epsilon)n\frac{T}{T+1}, (1+\epsilon)n\frac{T}{T+1}]$ deterministically. If not, one can perturb the value of T slightly so that $(T+1)/T = (1+a)^d$ exactly for some d. Because we only care about estimating λ, the exact value of T can be adjusted and x_t/T will still quickly approximate λ.