Adding Approximate Counters

Guy L. Steele Jr.
Oracle Labs
guy.steele@oracle.com

Jean-Baptiste Tristan
Oracle Labs
jean.baptiste.tristan@oracle.com

Abstract

We describe a general framework for adding the values of two approximate counters to produce a new approximate counter value whose expected estimated value is equal to the sum of the expected estimated values of the given approximate counters. (To the best of our knowledge, this is the first published description of any algorithm for adding two approximate counters.) We then work out implementation details for five different kinds of approximate counter and provide optimized pseudocode. For three of them, we present proofs that the variance of a counter value produced by adding two counter values in this way is bounded, and in fact is no worse, or not much worse, than the variance of the value of a single counter to which the same total number of increment operations have been applied. Addition of approximate counters is useful in massively parallel divide-and-conquer algorithms that use a distributed representation for large arrays of counters. We describe two machine-learning algorithms for topic modeling that use millions of integer counters, and confirm that replacing the integer counters with approximate counters is effective, speeding up a GPU-based implementation by over 65% and a CPU-based by nearly 50%, as well as reducing memory requirements, without degrading their statistical effectiveness.

Categories and Subject Descriptors D.1.3 [*Programming Techniques*]: Concurrent programming; E.2 [*Data Storage Representations*]; G.3 [*Probability and Statistics*]: Probabilistic algorithms

Keywords approximate counters, distributed computing, divide and conquer, multithreading, parallel computing, statistical counters

1. Introduction, Background, and Related Work

We will say that a *counter* is an integer-valued variable whose value is initially 0 and to which two operations can be freely applied: *increment*, which replaces the current value k with $k+1$, and *read*, which returns the current value of the variable. We assume that in a concurrent environment all the operations on a given counter are observed by all threads to have been performed as if in some specific sequential order, and so each operation may be regarded as atomic. It is then easy to see that the result of any *read* operation on a given counter will be the number of *increment* operations performed on that counter prior to the *read* operation.

Morris [8] introduced the notion of a *probabilistic counter*. His idea was to be able to use w bits to count more than 2^w things by actually increasing the counter value in response to only some *increment* operations rather than all of them; in one specific case, his algorithm increases a counter value of k with probability 2^{-k}, and a *read* operation that observes a counter value k returns $2^k - 1$ as a statistical estimate of the actual number of times the *increment* operation has been performed.

Morris furthermore provided a generalization of this algorithm as well as a statistical analysis. The probabilistic decision made by the *increment* operation can rely on the output of a random (or pseudorandom) number generator, and as Morris observes, "The random number generator can be of the simplest sort and no great demands are made on its properties." Flajolet [5] provided a detailed statistical analysis of the Morris algorithms.

Cvetkovski [3] describes Sampled-Log Approximate Counting, a variant that has better accuracy over most of the counting range and a lower expected number of actual advances of the counter. Csűrös [2] provides a framework and analysis applicable to a more general class of counter representations and algorithms than those of Morris. Mitchell and Day [7] introduce "flexible approximate counters" in which the probability of increasing the counter value may be controlled by a user-supplied table. Dice, Lev, and Moir (hereafter "DLM") [4] examine the use of probabilistic counters in a concurrent environment and explore alternate representations for the counter state.

Tristan, Tassarotti, and Steele [13] report that replacing standard counters with approximate counters allows a machine-learning algorithm (topic modeling) using LDA Gibbs (Latent Dirichlet Allocation with Gibbs sampling) on a GPU to make more efficient use of the limited memory on a GPU and therefore process larger data sets. They also report that the statistical performance of the algorithm remains essentially unchanged, and that, surprisingly, the version using approximate counters runs *faster*, despite the fact that incrementing an approximate counter requires generation of a pseudorandom number as well as other calculations. (Their Figure 6 indicates an overall speed improvement of approximately 9%.) They suggest two possible reasons for the increase in speed: (1) probabilistic incrementation performs many fewer actual writes to memory, and (2) less memory bandwidth is needed when performing bulk reads of the counters.

We have now extended this prior work by implementing distributed versions of the same algorithm (using either normal integer counters or approximate counters), on eight GPU cards with an Ethernet interconnect. The arrays of counters are replicated so that each GPU has a complete set of counters, but each GPU otherwise has just $\frac{1}{8}$ of the total dataset to be analyzed. On each iteration, each GPU clears its counters, processes its fraction of the dataset, and then uses a standard (hypercube-like) message-exchange pattern to sum elementwise the eight arrays of counter values, in such a way that every GPU receives all sums (as for the MPI operation `MPI_ALLREDUCE` [11, §4.11.4]); the version that uses approximate counters adds approximate counters by using the algorithms that we present in this paper. The behavior is quite similar to that of the single-GPU version reported by Tristan *et al.*: replacing integer counters with approximate counters does not affect the statistical performance of the algorithm, but allows the same hardware to process larger datasets. In one test, either 32-bit integer coun-

PPoPP '16, March 12-16, 2016, Barcelona, Spain
Copyright © 2016 ACM 978-1-4503-4092-2/16/03…$15.00
DOI: http://dx.doi.org/10.1145/2851141.2851147

ters were used, or 8-bit approximate counters; using approximate counters improved the overall speed of each iteration of the distributed application by over 65%. We have also tested a distributed multple-CPU implementation of another algorithm, organized as a stochastic cellular automaton, that addresses the same topic modeling problem. We believe that approximate counters will be increasingly useful in applications that perform statistical analysis of "Big Data" and that large-scale distributed implementations of these applications will require the ability to add approximate counters.

The novel contributions of this paper are: (1) a notational framework that encompasses many kinds of approximate counter representation and reconciles some conflicting notation in the literature; (2) the presentation, using this framework, of a general algorithm for adding approximate counters so that expected estimated value of the sum is equal to the sum of the expected estimated values of the given approximate counters (and we believe that this is the first algorithm for adding approximate counters ever to be published); (3) the derivation of optimized implementations of this general addition algorithm for five specific kinds of approximate counter; (4) the presentation of "cookbook" versions of the algorithms (highlighted by framing boxes) that also check for overflow and therefore are suitable for transcription into actual programs; (5) proofs for three kinds of counters that the variance remains bounded when counter values are added; and (6) measurements of distributed implementations of two machine-learning applications whose performance is greatly improved through the use of approximate counters and the ability to add approximate counters.

2. A General Framework

Following Csűrös [2], but generalizing his framework slightly to accommodate the alternate representations of DLM, we characterize a probabilistic counter that uses representation T, transition function $\tau : T \to T$, and transition probability function $Q : T \to [0, 1]$ as a variable that can contain values of type T whose value is initially S_0 (a specific value of type T). Two operations may be freely applied to such a counter: *increment*, which with probability $Q(s)$ replaces the current value s with $\tau(s)$ (and with probability $1-Q(s)$ does not change the value of the variable), and *read*, which returns a value $f(s)$ that may be regarded (that is, modeled) as a random variable whose expected value is the number of *increment* operations performed on that counter prior to the *read* operation. The function f is called the *unbiased estimator function*; Csűrös shows that f may be uniquely derived from Q.[1] We require the transition function τ never to cycle; that is, it satisfies the property that $\tau^i(S_0) \neq \tau^j(S_0)$ if $i \neq j$.[2] As a result, the possible states of a probabilistic counter produced by *increment* operations from the starting value S_0 are in one-to-one correspondence with the natural numbers, and the only reason to choose a representation T other than the natural numbers $\mathbb{N}$ is for "engineering purposes." For notational convenience, we will define $q_k = Q(\tau^k(S_0))$; this matches the meaning of "q_k" as used by Csűrös. We will also define $f_k = f(\tau^k(S_0))$, which corresponds to "$f(k)$" as used by Csűrös.

If we assume that the pseudo-function $random()$ chooses a real number randomly (or pseudorandomly) from the half-open real interval $[0, 1)$, then the *increment* and *read* operations may be implemented as follows:

[1] Mitchell and Day [7] take the opposite approach: their method represents the estimator function f (which they call "ϕ") as a monotonically increasing table and uses a solver to derive the transition probability function Q (which they call "p"), also represented as a table.

[2] In practice, it may be acceptable for an implementation of τ to signal an "overflow error" if it would otherwise be compelled to repeat a counter state. Some of the pseudocode we present explicitly checks for and signals overflow errors. One way to handle such a signal is to allow the counter to "saturate" once it reaches its highest possible value; our pseudocode shows the appropriate assignment statement (if needed) for this purpose.

```
1: procedure $increment$(var $X$: $T$)
2:     if $random() < Q(X)$ then $X \leftarrow \tau(X)$
```

```
1: procedure $read$($X$: $T$)
2:     return $f(X)$
```

That is the essence of it; all the rest is engineering and the statistical analysis behind it. We may choose T and τ and Q so that τ is easy to calculate, or Q is easy to calculate, or f is easy to calculate, or all three. It may be desirable to choose Q so as to guarantee certain statistical properties, for example so that the variance of values returned by f can be bounded (and this can be done in various ways), and perhaps also to choose Q so that the dynamic range of the counter will be appropriate for a specific application.

In addition to the pseudofunction $random()$ already described, which returns a real value (or floating-point approximation) in the range $[0, 1)$, we will also find it convenient to assume the availability of two additional pseudofunctions. First, $randomBits(j)$ takes a nonnegative integer j and returns an integer value chosen uniformly randomly from the 2^j integers in the range $[0, 2^j)$ (which is equivalent to independently and uniformly choosing j bits at random and using them as binary digits to represent an integer value). Second, $allZeroRandomBits(j)$ takes a nonnegative integer j and returns *true* with probability 2^{-j} and *false* with probability $1 - 2^{-j}$ (which is equivalent to independently and uniformly choosing j bits at random and testing whether they all happen to be 0). Note that $randomBits(0)$ always returns 0 and $allZeroRandomBits(0)$ always returns *true*.

3. Adding Approximate Counters

Generalizing the observation of Csűrös to our framework, f may be derived from Q as follows:

$$f_k = f(\tau^k(S_0)) = \sum_{0 \leq i < k} \frac{1}{Q(\tau^i(S_0))} = \sum_{0 \leq i < k} \frac{1}{q_i}$$

Note that $f_0 = 0$. Because each probability q_i lies in the range $[0, 1]$, we have $\frac{1}{q_i} \geq 1$, and therefore the values $f_k = f(\tau^k(S_0))$ are strictly monotonic in k, and then some: $f_k + 1 \leq f_{k+1}$. Therefore we can uniquely define a representation-finding function $\phi : [0, +\infty) \to T$ that given any nonnegative real number v returns $\tau^K(S_0)$ where K is the unique integer such that $f_K \leq v < f_{K+1}$. Because the function ϕ produces a "quantized" result, it is a left inverse to f but not a right inverse: $\phi(f(\tau^K(S_0))) = \tau^K(S_0)$ for all $k \geq 0$, but in general it is not true that $f(\phi(v)) = v$; the best we can say is that $f(\phi(v)) \leq v < f(\tau(\phi(v)))$. A general implementation of ϕ is as follows:

```
1: procedure $\phi(v)$
2:     let $S \leftarrow S_0$
3:     loop
4:         let $S' \leftarrow \tau(S)$
5:         if $v < f(S')$ then return $S$
6:         $S \leftarrow S'$
7:     end loop
```

However, specific counter representations typically permit a much faster (non-iterative) implementation. Just as τ may in practice signal an overflow error, a specialized implementation of ϕ may likewise signal an overflow error if its argument is too large.

Given an implementation of ϕ for a specific sort of counter, we can "add" two counter representation values x and z so that the expected estimated value of the sum $x \oplus z$ equals the sum of their individual expected estimated values (provided that the processes that produced x and z are statistically independent):

$$x \oplus z = \begin{cases} \tau(\phi(f(x)+f(z))) & \text{with probability } \Delta \\ \phi(f(x)+f(z)) & \text{with probability } (1-\Delta) \end{cases}$$

$$\text{where } \Delta = \frac{(f(x)+f(z)) - f(\phi(f(x)+f(z)))}{f(\tau(\phi(f(x)+f(z)))) - f(\phi(f(x)+f(z)))}$$

The expected value of $f(x \oplus z)$ will then be

$$\begin{aligned} &(1-\Delta)f(\phi(f(x)+f(z))) + \Delta f(\tau(\phi(f(x)+f(z)))) \\ &= f(\phi(f(x)+f(z))) + \\ &\quad \Delta\big(f(\tau(\phi(f(x)+f(z)))) - f(\phi(f(x)+f(z)))\big) \\ &= f(\phi(f(x)+f(z))) + \\ &\quad (f(x)+f(z)) - f(\phi(f(x)+f(z))) \\ &= f(x)+f(z) \end{aligned}$$

as desired.

We can express this addition operation as an algorithm that modifies a counter X by adding into it the estimated value of a statistically independent counter Z (we do this so that it will be similar in form to the *increment* operation):

```
1: procedure add(var $X$: $T$, $Z$: $T$)
2:     let $v \leftarrow f(X)$
3:     let $w \leftarrow f(Z)$
4:     let $S \leftarrow v + w$
5:     let $K \leftarrow \phi(S)$
6:     let $V \leftarrow f(K)$
7:     let $W \leftarrow f(\tau(K))$
8:     let $\Delta \leftarrow \frac{S-V}{W-V}$
9:     if $random() < \Delta$ then $X \leftarrow \tau(K)$ else $X \leftarrow K$
```

This algorithm is not terribly mysterious: it adds two counters X and Z by computing their expected values v and w, adding those values to get a sum S, then mapping that sum back to one of the two counter states whose expected values straddle S, in such a way that the expected value of the result is S. The real point is that we will use this general algorithm as a template for addition algorithms specialized to particular counter representations by inlining specific definitions of f and τ and ϕ, and then optimizing. This strategy guarantees that appropriate expected values are maintained. However, it is necessary to provide a separate proof for each kind of counter that the variance is bounded.

4. General Morris Counters

For the general probabilistic counter defined by Morris,

$$\text{type } T \text{ is } \mathbb{N} \qquad S_0 = 0$$

$$\tau(x) = x+1 \qquad Q(x) = q^{-x}$$

(where q is a fixed constant equal to $1+\frac{1}{a}$ where a is the parameter used by Morris); it follows that

$$f(x) = \frac{q^x - 1}{q-1}$$

(which is equivalent to the formula $a((1+\frac{1}{a})^x - 1)$ given by Morris) and

$$\phi(v) = \left\lfloor \log_q((q-1)v+1) \right\rfloor$$

Then *increment* and *read* operations may be implemented thusly:

```
1: procedure increment(var $X$: $\mathbb{N}$)
2:     if $random() < q^{-X}$ then $X \leftarrow X + 1$

1: procedure read(var $X$: $\mathbb{N}$)
2:     return $\frac{q^X - 1}{\langle q-1 \rangle}$
```

We use angle brackets $\langle \cdots \rangle$ to indicate a constant ($q-1$ in this case) whose value can be computed at compile time.

If instead we let the counter representation type T be the set of values $\mathbb{Z}_{2^b} = \{0, 1, 2, \ldots, 2^b - 1\}$ representable in a b-bit word as unsigned binary integers, the *increment* operation may perform overflow checking:

```
1: procedure increment(var $X$: $\mathbb{Z}_{2^b}$)
2:     if $random() < q^{-X}$ then
3:         if $X \neq \langle 2^b - 1 \rangle$ then $X \leftarrow X + 1$
4:         else overflow error
```

In some implementation contexts, when b is relatively small it may be worthwhile to tabulate the functions Q and f—that is, to preconstruct two arrays Q' and f' containing the values of $Q(X)$ and $f(X)$ for all $0 \leq X < 2^b$, but let $Q'[2^b - 1] = 0$ so as to serve as a sentinel that will effectively enforce saturation on overflow. Such use of tabulations is certainly worth considering when $b = 8$, and perhaps even when b is as large as 16. Then the *increment* and *read* operations are simply:

```
1: procedure increment(var $X$: $\mathbb{Z}_{2^b}$)
2:     if $random() < Q'[X]$ then $X \leftarrow X + 1$

1: procedure read(var $X$: $\mathbb{Z}_{2^b}$)
2:     return $f'[X]$
```

Counter Z may be added into counter X as follows:

```
1: procedure add(var $X$: $\mathbb{N}$, $Z$: $\mathbb{N}$)
2:     let $v \leftarrow \frac{q^X - 1}{\langle q-1 \rangle}$
3:     let $w \leftarrow \frac{q^Z - 1}{\langle q-1 \rangle}$
4:     let $S \leftarrow v + w$
5:     let $K \leftarrow \lfloor \log_q(\langle q-1 \rangle S + 1) \rfloor$
6:     let $V \leftarrow \frac{q^K - 1}{\langle q-1 \rangle}$
7:     let $W \leftarrow \frac{q^{K+1} - 1}{\langle q-1 \rangle}$
8:     let $\Delta \leftarrow \frac{S-V}{W-V}$
9:     if $random() < \Delta$ then $X \leftarrow K + 1$ else $X \leftarrow K$
```

If the computation of the base-q logarithm is not entirely accurate in the computation of K, it may fall just under an integer value that it should have equaled or exceeded. If so, K will be 1 smaller than it should have been. But this error is benign in this context, because then $\Delta > 1$, and so K will be incremented. There will be no further chance to increment K again, but that would have occurred only with probability commensurate with other floating-point errors.

However, in many practical situations, it is faster to avoid the computation of a logarithm entirely. If the implementation pretabulates the values of $f(x)$ in an array f' (as already described above for use by the *read* operation) the array can be searched for the correct position. For this purpose it is best to make f' have length $2^b + 2$, with $f'[X] = f(X)$ for $0 \leq X \leq 2^b$, and place a sentinel value—either $+\infty$ or a very large finite value—in the last position at index $2^b + 1$. Also, note that $W - V$ can be simplified to q^k, so is it is helpful to pretabulate q^{-k} in a length-2^b array p. Then for b-bit words with overflow checking, we have:

```
 1: procedure add(var $X$: $\mathbb{Z}_{2^b}$, $Z$: $\mathbb{Z}_{2^b}$)
 2:     let $S \leftarrow f'[X] + f'[Z]$
 3:     let $K \leftarrow \max(X, Z)$
 4:     while $S \geq f'[K+1]$ do $K \leftarrow K + 1$
 5:     if $K \leq \langle 2^b - 1 \rangle$ then
 6:         let $V \leftarrow f'[K]$
 7:         if $random() < (p[K])(S - V)$ then
 8:             if $K \neq \langle 2^b - 1 \rangle$ then $X \leftarrow K + 1$
 9:             else overflow error: $X \leftarrow \langle 2^b - 1 \rangle$
10:         else $X \leftarrow K$
11:     else overflow error: $X \leftarrow \langle 2^b - 1 \rangle$
```

If $q = 1.08$, for example, the **while** loop should iterate no more than 8 times, so this may be much faster that the calculation of a logarithm in software.

Note that the explicit overflow checking only ensures that the value of K, once computed, will fit in a b-bit word. It is assumed that the computation of S will use an arithmetic with sufficient range to avoid overflow. This is typically not difficult in practice: if $b = 8$, and S is computed using standard IEEE 754 double-precision floating-point operations, then the computation of S will never suffer overflow.

Morris asserts (without proof) that the variance in the estimated value of a counter after n *increment* operations have been performed is $n(n-1)/2a = \frac{q-1}{2}n(n-1)$. In our Appendix, Lemma 1 provides an explicit proof of a generalization of this proposition, and Theorem 1 shows that when *add* operations are also used, the variance is bounded by $\frac{q-1}{2}n(n-1) + \rho$ where $\rho = \frac{1}{-2(q^2-4q+1)}$. Note that ρ lies in the range $[\frac{1}{6}, \frac{1}{4})$ for $1 < q \leq 2$, so this is a reasonably tight bound on the variance even for small n.

5. Binary Morris Counters

For the simplest sort of probabilistic counter defined by Morris, we choose $q = 2$ (which corresponds to $a = 1$ as used by Morris):

$$\text{type } T \text{ is } \mathbb{N} \qquad S_0 = 0$$
$$\tau(x) = x + 1 \qquad Q(x) = 2^{-x}$$

It follows that

$$f(x) = 2^x - 1$$
$$\phi(v) = \lfloor \log_2(v+1) \rfloor$$

This makes the *increment* and *read* operations especially simple:

```
1: procedure increment(var $X$: $\mathbb{N}$)
2:     if $allZeroRandomBits(X)$ then $X \leftarrow X + 1$
```

```
1: procedure read(var $X$: $\mathbb{N}$)
2:     return $2^X - 1$
```

Skilled programmers know various clever ways to compute $2^X - 1$, depending on whether the result is to be represented as an integer (perhaps by using a left-shift operation) or as a floating-point value (using a scaling operation to adjust the exponent field). For an integer result, the *read* operation might be simply:

```
1: procedure read(var $X$: $\mathbb{N}$)
2:     return $(1 \ll X) - 1$
```

Note that in C, C++, the Java$^{\text{TM}}$ programming language, or other languages that have similar rules for the shift operator ≪, it may be important to write the literal 1 in the expression $1 \ll X$ as, say, `1L` or `1ull` in order to ensure that the result has a specific (sufficiently large) type, such as (respectively) `long` or `unsigned long long`.

It is especially easy to add counter Z into counter X: because $f(k+1) = 2^{k+1} - 1 > 2(2^k - 1) = 2f(k)$, it follows that V is always equal to $f(\max(X, Z))$, so $S - V = f(\min(X, Z))$, and therefore $\Delta = \frac{2^{\min(X,Z)}-1}{2^{\max(X,Z)}}$. This leads to a very simple procedure that does not need to implement the ϕ function explicitly:

```
1: procedure add(var $X$: $\mathbb{N}$, $Z$: $\mathbb{N}$)
2:     let $K \leftarrow \max(X, Z)$
3:     let $L \leftarrow \min(X, Z)$
4:     if $allZeroRandomBits(K - L)$ then
5:         if $\neg allZeroRandomBits(L)$ then $X \leftarrow K + 1$
6:         else $X \leftarrow K$
7:     else $X \leftarrow K$
```

With overflow checking, *increment* and *add* look like this:

```
1: procedure increment(var $X$: $\mathbb{Z}_{2^b}$)
2:     if $allZeroRandomBits(X)$ then
3:         if $X \neq \langle 2^b - 1 \rangle$ then $X \leftarrow X + 1$
4:         else overflow error
```

```
1: procedure add(var $X$: $\mathbb{Z}_{2^b}$, $Z$: $\mathbb{Z}_{2^b}$)
2:     let $K \leftarrow \max(X, Z)$
3:     let $L \leftarrow \min(X, Z)$
4:     if $allZeroRandomBits(K - L)$ then
5:         if $\neg allZeroRandomBits(L)$ then
6:             if $K \neq \langle 2^b - 1 \rangle$ then $X \leftarrow K + 1$
7:             else overflow error: $X \leftarrow K$
8:         else $X \leftarrow K$
9:     else $X \leftarrow K$
```

While the binary Morris approximate counter is in principle a special case of the general Morris approximate counter with $q = 2$, our addition algorithm is rather different from the one presented for the general case. Therefore in our Appendix we present Theorem 2, a separate proof of bounded variance for binary Morris approximate counters when *add* operations are used, showing that the variance is bounded by $\frac{n(n-1)}{2}$ (no additive constant ρ is needed).

6. Csűrös Floating-Point Counters

For the general (that is, scaled) floating-point counters defined by Csűrös [2], parametrized not only by a base q $(1 < q \leq 2)$ but also by $M = 2^s$ for some integer $s \geq 0$, we have

$$\text{type } T \text{ is } \mathbb{N} \qquad S_0 = 0$$
$$\tau(x) = x + 1 \qquad Q(x) = q^{-\lfloor x/M \rfloor}$$

For convenience, let $\mu = \frac{M}{q-1}$; it follows that

$$f(x) = \big(\mu + (x \bmod M)\big)q^{\lfloor x/M \rfloor} - \mu$$
$$\phi(v) = d \cdot M + \left\lfloor \frac{v+\mu}{q^d} - \mu \right\rfloor \text{ where } d = \left\lfloor \log_q \frac{v+\mu}{\mu} \right\rfloor$$

In effect, all bits of an integer x *except* the s lower order bits are treated as a binary[3] exponent $e = \lfloor x/M \rfloor$, and the s low-order bits of x (that is, $x \bmod M$) are treated as a floating-point significand with an implicit leading 1-bit (that's why you have to add μ before multiplying by 2^e); finally, μ is subtracted so that the integer representation 0 will map to the estimated value 0 (and, most pleasantly, every integer representation x less than M maps to the estimated value x). Again the *increment* operation is very simple, and *read* is reasonably simple:

```
1: procedure increment(var $X$: $\mathbb{N}$)
2:     if $random() < q^{-\lfloor X/M \rfloor}$ then $X \leftarrow X + 1$
```

```
1: procedure read(var $X$: $\mathbb{N}$)
2:     return $\big(\mu + (X \bmod M)\big)q^{\lfloor X/M \rfloor} - \mu$
```

And again, in some implementation contexts, if the range of values for X is relatively small, it may be worthwhile to tabulate the functions Q and f as arrays Q' and f' (but letting the last element of Q' be 0) so that the *increment* and *read* operations are simply:

```
1: procedure increment(var $X$: $\mathbb{Z}_{2^b}$)
2:     if $random() < Q'[X]$ then $X \leftarrow X + 1$
1: procedure read(var $X$: $\mathbb{Z}_{2^b}$)
2:     return $f'[X]$
```

[3] Csűrös [2] primarily assumes binary counters ($q = 2$), but also briefly discusses "scaled floating-point counters," relating them to earlier work by Stanojević [12]. These bear the same relationship to binary Csűrös floating-point counters that general Morris counters have to binary Morris counters. We wish to add the observation that the primary motivation for requiring M to be a power of 2 is to make it easy to compute $\lfloor x/M \rfloor$ and $x \bmod M$ using bit shifting and masking operations. For some applications, using base-2 Csűrös floating-point counters with M an integer that is not a power of 2 may well be preferable to limiting M to be some power of 2 and then choosing a value of q other than 2. For other applications, it may be convenient to choose some $q < 2$ but then choose M so that μ is an integer. Our pseudocode assumes only that M is an integer unless otherwise indicated, and the proofs in our Appendix assume only that M is an integer.

But for $q = 2$ and $M = 2^s$, shifting and bitwise operations can be useful (we show a version of *increment* with overflow checking):

```
procedure increment(var $X: \mathbb{Z}_{2^b}$)
    if $allZeroRandomBits(X \gg s)$ then
        if $X \neq \langle 2^b - 1 \rangle$ then $X \leftarrow X + 1$ else overflow error
procedure read(var $X: \mathbb{N}$)
    return $((M + (X \,\&\, \langle M - 1 \rangle)) \ll (X \gg s)) - M$
```

In general (for any q and M), addition goes like this:

```
procedure add(var $X: \mathbb{N}$, $Z: \mathbb{N}$)
    let $v \leftarrow \big(\mu + (X \bmod M)\big)q^{\lfloor X/M \rfloor} - \mu$
    let $w \leftarrow \big(\mu + (Z \bmod M)\big)q^{\lfloor Z/M \rfloor} - \mu$
    let $S \leftarrow v + w$
    let $d \leftarrow \left\lfloor \log_q \frac{S+\mu}{\mu} \right\rfloor$
    let $K \leftarrow d \cdot M + \left\lfloor \frac{S+\mu}{q^d} - \mu \right\rfloor$
    let $V \leftarrow \big(\mu + (K \bmod M)\big)q^{\lfloor K/M \rfloor} - \mu$
    let $W \leftarrow \big(\mu + ((K+1) \bmod M)\big)q^{\left\lfloor \frac{K+1}{M} \right\rfloor} - \mu$
    let $\Delta \leftarrow \frac{S-V}{W-V}$
    if $random() < \Delta$ then $X \leftarrow K + 1$ else $X \leftarrow K$
```

Note that $d \geq 0$; indeed, $d \geq \max(\lfloor X/M \rfloor, \lfloor Z/M \rfloor)$. Now $1 < q \leq 2$ and $0 \leq r < 1$ together imply $q^r < 2$; therefore $\lfloor \frac{S+\mu}{2^d} - \mu \rfloor = \left\lfloor \frac{S+\mu}{q^{\lfloor \log_q \frac{S+\mu}{\mu} \rfloor}} - \mu \right\rfloor = \left\lfloor \frac{S+\mu}{q^{\log_q \frac{S+\mu}{\mu} - r}} - \mu \right\rfloor = \lfloor q^r \mu - \mu \rfloor \leq q^r \mu - \mu = (q^r - 1)\mu < \mu \leq M$. Therefore $K \bmod M = \left\lfloor \frac{S+\mu}{q^d} - \mu \right\rfloor$, and so $\lfloor K/M \rfloor = d$ and we have

$$\begin{aligned}
S - V &= S - \big((\mu + (K \bmod M))q^{\lfloor K/M \rfloor} - \mu\big) \\
&= S - \left(\left(\mu + \left\lfloor \tfrac{S+\mu}{q^d} \right\rfloor - \mu\right) q^{\lfloor K/M \rfloor} - \mu\right) \\
&= S - \left(\left\lfloor \tfrac{S+\mu}{q^d} \right\rfloor q^d - \mu\right) \\
&= S - \big(((S+\mu) - ((S+\mu) \bmod q^d)) - \mu\big) \\
&= (S+\mu) \bmod q^d
\end{aligned}$$

Next, it's worth simplifying $W - V$ by case analysis as follows:
(a) If $(K \bmod M) < M - 1$, then $\lfloor (K+1)/M \rfloor = \lfloor K/M \rfloor$, and

$$\begin{aligned}
W - V &= \big((\mu + ((K+1) \bmod M))q^{\lfloor (K+1)/M \rfloor} - \mu\big) \\
&\quad - \big((\mu + (K \bmod M))q^{\lfloor K/M \rfloor} - \mu\big) \\
&= \big(\mu + ((K+1) \bmod M)\big)q^{\lfloor K/M \rfloor} \\
&\quad - \big(\mu + (K \bmod M)\big)q^{\lfloor K/M \rfloor} \\
&= \big(((K+1) \bmod M) - (K \bmod M)\big)q^{\lfloor K/M \rfloor} \\
&= \big(((K+1) - K) \bmod M\big)q^{\lfloor K/M \rfloor} = q^{\lfloor K/M \rfloor}
\end{aligned}$$

(b) If $(K \bmod M) = M - 1$, then $\lfloor (K+1)/M \rfloor = \lfloor K/M \rfloor + 1$:

$$\begin{aligned}
W - V &= \big((\mu + ((K+1) \bmod M))q^{\lfloor (K+1)/M \rfloor} - \mu\big) \\
&\quad - \big((\mu + (K \bmod M))q^{\lfloor K/M \rfloor} - \mu\big) \\
&= \big(\mu + ((K+1) \bmod M)\big)q^{\lfloor K/M \rfloor + 1} \\
&\quad - \big(\mu + (K \bmod M)\big)q^{\lfloor K/M \rfloor} \\
&= \big(q\mu + q((K+1) \bmod M)\big)q^{\lfloor K/M \rfloor} \\
&\quad - \big(\mu + (K \bmod M)\big)q^{\lfloor K/M \rfloor} \\
&= \big(q\mu + q \cdot 0 - \mu - (M-1)\big)q^{\lfloor K/M \rfloor} = q^{\lfloor K/M \rfloor}
\end{aligned}$$

Therefore in all cases $W - V = q^{\lfloor K/M \rfloor} = q^d$.

So, after some simplification, we have:

```
procedure add(var $X: \mathbb{N}$, $Z: \mathbb{N}$)
    let $S \leftarrow (\mu + (X \bmod M))q^{\lfloor X/M \rfloor} +$
            $(\mu + (Z \bmod M))q^{\lfloor Z/M \rfloor} - \langle 2\mu \rangle$
    let $d \leftarrow \left\lfloor \log_q \frac{S+\mu}{\mu} \right\rfloor$
    let $K \leftarrow d \cdot M + \left\lfloor \frac{S+\mu}{q^d} - \mu \right\rfloor$
    if $random() < \frac{S+\mu}{q^d} \bmod 1$ then $X \leftarrow K + 1$ else $X \leftarrow K$
```

However, in practice it may be faster to search an array f' as described in Section 4.

If we restrict our attention to $q = 2$ and $M = 2^s$, and assume the use of a machine word B bits wide for representing estimated values (in contrast to words b bits wide that may be used for counter representation values) and an instruction to count the number of leading zeros in a B-bit word, we can recast the algorithm (with overflow checking) as follows (letting $S' = S + \mu > 0$):

```
procedure add(var $X: \mathbb{Z}_{2^b}$, $Z: \mathbb{Z}_{2^b}$)
    let $S' \leftarrow \big((\mathbb{Z}_{2^B})(M + (X \,\&\, \langle M-1 \rangle)) \ll (X \gg s)\big) +$
            $\big((\mathbb{Z}_{2^B})(M + (Z \,\&\, \langle M-1 \rangle)) \ll (Z \gg s)\big) - M$
    let $d \leftarrow \langle B - (s+1) \rangle - countLeadingZeros(S')$
    let $K \leftarrow (d \ll s) + (S' \gg d) - M$
    if $K \leq \langle 2^b - 1 \rangle$ then
        if $randomBits(d) < \big(S' \,\&\, ((1 \ll d) - 1)\big)$ then
            if $K \neq \langle 2^b - 1 \rangle$ then $X \leftarrow K + 1$
            else overflow error: $X \leftarrow \langle 2^b - 1 \rangle$
        else $X \leftarrow K$
    else overflow error: $X \leftarrow \langle 2^b - 1 \rangle$
```

In our Appendix is Theorem 3, a proof of bounded variance for scaled Csűrös approximate counters when *add* operations are used.

7. DLM Probability Counters

For the probabilistic counter defined by DLM [4], where the value in a counter is not approximately the logarithm of the number of increment operations but rather the probability that the next increment operation should change the counter:

type T is the closed real interval $[0, 1]$

$$S_0 = 1.0 \qquad \tau(x) = \tfrac{x}{q} \qquad Q(x) = x$$

If, as before, we let $q = 1 + \frac{1}{a}$ (thus $a = \frac{1}{q-1}$), it follows that

$$f(x) = \frac{a}{x} - a$$

(which is equivalent to the formula $a(\frac{1}{x} - 1)$ given by DLM but has one fewer operation). Then the *increment* and *read* operations may be implemented as follows:

```
procedure increment(var $X: [0, 1]$)
    if $random() < X$ then $X \leftarrow \langle q^{-1} \rangle X$
procedure read(var $X: [0, 1]$)
    return $\frac{a}{X} - a$
```

The counter representation is not quantized to integers, only to floating-point approximations[4] of real numbers, so it may not be necessary in practice when implementing the ϕ function to choose randomly between two possible values, though for complete accuracy one would indeed perform the floating-point calculations to "infinite precision" and then do the final floating-point rounding in an appropriately probabilistic manner, as described by Forsythe [6] and later by Callahan [1] and Parker [10, §6.4][9]. Assuming no

[4] Although we cannot resist pointing out that fixed-point fractions may be a practical alternate representation for probabilities, with the advantage of allowing the use of a random number generator that generates random b-bit integers, which may be faster than generating random floating-point values.

need to be that finicky, we pretend that ϕ is an exact inverse of f:

$$\phi(v) = \frac{a}{v+a}$$

and so counter Z may be added into counter X as follows:

```
procedure $add$(var $X$: $[0,1]$, $Z$: $[0,1]$)
    let $S \leftarrow (\frac{a}{X} - a) + (\frac{a}{Z} - a)$
    let $K \leftarrow \frac{a}{S+a}$
    $X \leftarrow K$
```

After simplification, this becomes:

```
procedure $add$(var $X$: $[0,1]$, $Z$: $[0,1]$)
    $X \leftarrow \dfrac{1}{\frac{1}{X} + \frac{1}{Z} - 1}$
```

The remarkable thing about this representation is that the parameter a is not required for computing the sum of two counters, *provided*, of course, that the two counters do use the same parameter a.

While DLM probability counters may not require fewer bits for their representation than ordinary integer counters, they do share with other kinds of approximate counters the property that the *increment* operation does not always perform a write to memory.

8. DLM Floating-Point Counters

For the floating-point counters defined by DLM [4], which are parametrized by a constant $M = 2^s$ for some integer $s \geq 0$ and by a second constant Θ (called `MantissaThreshold` by DLM) that is a positive even integer not greater than M, we have

type T is $\mathbb{N}$ $\qquad S_0 = 0 \qquad Q(x) = 2^{-\lfloor x/M \rfloor}$

$$\tau(x) = \textbf{if } (k \bmod M) + 1 < \Theta \textbf{ then } k + 1$$
$$\textbf{else } k - (k \bmod M) + M + \tfrac{\Theta}{2}$$

and it follows that

$$f(x) = (k \bmod M)2^{\lfloor k/M \rfloor}$$

In effect, all bits of an integer k *except* the s lower order bits are treated as a binary exponent $e = \lfloor k/M \rfloor$, and the s low-order bits of k (that is, $k \bmod M$) are treated as a floating-point significand that does *not* have an implicit leading 1-bit and whose leading bit may or may not be 1 (that is, the representation is not necessarily in normalized form). As a result, there is some redundancy in the representation: two or more counter representation values may map to the same estimated value, and ϕ is not a function but rather a relation. One similarity to the floating-point representation used by Csűrös is that every integer representation k less than M maps to the estimated value k. Again the *increment* operation is very simple, and *read* is reasonably simple:

```
procedure $increment$(var $X$: $\mathbb{N}$)
    if $allZeroRandomBits\left(\lfloor \frac{X}{M} \rfloor\right)$ then
        if $(X \bmod M) + 1 < \Theta$ then $X \leftarrow X + 1$
        else $X \leftarrow X - (X \bmod M) + \langle M + \frac{\Theta}{2} \rangle$
```

```
procedure $read$(var $X$: $\mathbb{N}$)
    return $(X \bmod M)2^{\lfloor X/M \rfloor}$
```

Again, we cleverly use shifting and bitwise operations:

```
procedure $increment$(var $X$: $\mathbb{N}$)
    if $allZeroRandomBits\ (X \gg s)$ then
        if $(X \,\&\, \langle M-1 \rangle) < \langle \Theta - 1 \rangle$ then $X \leftarrow X + 1$
        else $X \leftarrow (X \,\&\, \langle \neg(M-1) \rangle) + \langle M + \frac{\Theta}{2} \rangle$
```

```
procedure $read$(var $X$: $\mathbb{N}$)
    return $(X \,\&\, \langle M-1 \rangle) \ll (X \gg s)$
```

DLM point out [4] that the purpose of using a redundant representation and allowing Θ to be smaller than M is to allow a choice of representations that are equivalent in value but differ in their probability of changing the counter during an *increment* operation; this flexibility is used to address situations in which there appears to be high contention among multiple threads for a shared counter. We will assume that this flexibility is not needed when performing an *add* operation, and therefore we are free to choose an implementation for ϕ that does not take Θ into account:

```
procedure $add$(var $X$: $\mathbb{N}$, $Z$: $\mathbb{N}$)
    let $S \leftarrow (X \bmod M)2^{\lfloor X/M \rfloor} + (Z \bmod M)2^{\lfloor Z/M \rfloor}$
    if $S < M$ then $X \leftarrow S$
    else
        let $d \leftarrow \lfloor \log_2(S+1) \rfloor - \langle s+1 \rangle$
        let $K \leftarrow d \cdot 2^s + \lfloor \frac{S}{2^d} \rfloor$
        let $V \leftarrow (K \bmod M)2^{\lfloor K/M \rfloor}$
        let $W \leftarrow ((K+1) \bmod M)2^{\lfloor (K+1)/M \rfloor}$
        let $\Delta \leftarrow \frac{S-V}{W-V}$
        if $random() < \Delta$ then
            if $(K \bmod M) = \langle M-1 \rangle$ then
                $X \leftarrow (d+1)2^s + \langle \frac{M}{2} \rangle$
            else $X \leftarrow K + 1$
        else $X \leftarrow K$
```

After simplification and introduction of bitwise operations this is:

```
procedure $add$(var $X$: $\mathbb{N}$, $Z$: $\mathbb{N}$)
    let $S \leftarrow \big((X \,\&\, \langle M-1 \rangle) \ll (X \gg s)\big) +$
            $\big((Z \,\&\, \langle M-1 \rangle) \ll (Z \gg s)\big)$
    if $S < M$ then $X \leftarrow S$
    else
        let $d \leftarrow \langle B - s \rangle - countLeadingZeros(S')$
        let $K \leftarrow (d \ll s) + (S \gg d)$
        if $randomBits(d) < \big(S \,\&\, ((1 \ll d) - 1)\big)$ then
            if $(K \bmod M) = \langle M-1 \rangle$ then
                $X \leftarrow K + \langle \frac{M}{2} + 1 \rangle$
            else $X \leftarrow K + 1$
        else $X \leftarrow K$
```

With overflow checking, the *increment* and *add* operations look like this (we assume that one may disobey the Θ threshold if doing so will postpone an overflow error):

```
procedure $increment$(var $X$: $\mathbb{N}$)
    if $allZeroRandomBits\ (X \gg s)$ then
        if $(X \,\&\, \langle M-1 \rangle) < \langle \Theta - 1 \rangle$ then $X \leftarrow X + 1$
        else if $(X \gg s) = \langle 2^{b-s} - 1 \rangle$ then
            if $X \neq \langle 2^b - 1 \rangle$ then $X \leftarrow X + 1$
            else overflow error
        else $X \leftarrow X \,\&\, \langle \neg(M-1) \rangle) + \langle M + \frac{\Theta}{2} \rangle$
procedure $add$(var $X$: $\mathbb{N}$, $Z$: $\mathbb{N}$)
    let $S \leftarrow \big((X \,\&\, \langle M-1 \rangle) \ll (X \gg s)\big) +$
            $\big((Z \,\&\, \langle M-1 \rangle) \ll (Z \gg s)\big)$
    if $S < M$ then $X \leftarrow S$
    else
        let $d \leftarrow \langle B - s \rangle - countLeadingZeros(S')$
        let $K \leftarrow (d \ll s) + (S \gg d)$
        if $K \leq \langle 2^b - 1 \rangle$ then
            if $randomBits(d) < \big(S \,\&\, ((1 \ll d) - 1)\big)$ then
                if $K \neq \langle 2^b - 1 \rangle$ then
                    if $(K \bmod M) = \langle M-1 \rangle$ then
                        $X \leftarrow K + \langle \frac{M}{2} + 1 \rangle$
                    else $X \leftarrow K + 1$
                else overflow error: $X \leftarrow \langle 2^b - 1 \rangle$
            else $X \leftarrow K$
        else overflow error: $X \leftarrow \langle 2^b - 1 \rangle$
```

	8-bit counter: $\log_2 f\big(2^8-1\big)$						10-bit counter: $\log_2 f\big(2^{10}-1\big)$						16-bit counter: $\log_2 f\big(2^{16}-1\big)$					
M	1	2	4	8	16	32	4	8	16	32	64	128	256	512	1024	2048	4096	8192
$q = 2$	255.0	128.5	65.8	34.9	19.9	12.9	257.8	130.9	67.9	36.9	21.9	14.9	263.9	136.9	73.9	42.9	27.9	20.9
$q = 1.9$	236.2	119.2	61.2	32.6	18.9	12.5	239.0	121.5	63.3	34.7	20.9	14.5	245.2	127.7	69.4	40.8	27.0	20.6
$q = 1.8$	216.5	109.5	56.4	30.3	17.8	12.0	219.2	111.7	58.5	32.4	19.8	14.0	225.4	117.9	64.6	38.5	25.9	20.1
$q = 1.7$	195.7	99.1	51.3	27.9	16.7	11.5	198.3	101.4	53.4	29.9	18.7	13.6	204.5	107.5	59.5	36.0	24.8	19.6
$q = 1.6$	173.6	88.2	45.9	25.3	15.5	11.1	176.1	90.4	48.0	27.4	17.5	13.1	182.3	96.5	54.1	33.4	23.6	19.1
$q = 1.5$	150.1	76.6	40.3	22.6	14.3	10.6	152.6	78.8	42.4	24.7	16.3	12.6	158.7	84.9	48.4	30.7	22.4	18.6
$q = 1.4$	125.1	64.2	34.2	19.8	13.0	10.0	127.4	66.4	36.3	21.8	15.0	12.1	133.6	72.5	42.4	27.9	21.1	18.1
$q = 1.3$	98.2	51.0	27.8	16.8	11.7	9.5	100.5	53.1	29.9	18.8	13.7	11.5	106.6	59.2	36.0	24.8	19.8	17.6
$q = 1.2$	69.3	36.8	21.0	13.7	10.4	9.0	71.5	38.9	23.1	15.7	12.4	11.0	77.7	45.0	29.2	21.7	18.5	17.0
$q = 1.1$	38.3	21.8	14.0	10.6	9.1	8.5	40.4	23.9	16.1	12.6	11.1	10.5	46.5	29.9	22.1	18.7	17.2	16.5
$q = 1.09$	35.1	20.3	13.3	10.3	9.0	8.4	37.2	22.3	15.4	12.3	11.0	10.4	43.3	28.4	21.4	18.4	17.0	16.5
$q = 1.08$	31.9	18.8	12.7	10.0	8.9	8.4	34.0	20.8	14.7	12.0	10.9	10.4	40.1	26.9	20.7	18.1	16.9	16.4
$q = 1.07$	28.7	17.2	12.0	9.7	8.7	8.3	30.8	19.3	14.0	11.7	10.7	10.3	36.8	25.3	20.1	17.8	16.8	16.4
$q = 1.06$	25.4	15.7	11.3	9.4	8.6	8.3	27.5	17.8	13.3	11.5	10.6	10.3	33.6	23.8	19.4	17.5	16.7	16.3
$q = 1.05$	22.2	14.2	10.7	9.2	8.5	8.2	24.3	16.3	12.7	11.2	10.5	10.2	30.3	22.3	18.8	17.2	16.6	16.3
$q = 1.04$	19.0	12.8	10.1	8.9	8.4	8.1	21.1	14.8	12.1	10.9	10.4	10.2	27.1	20.9	18.1	17.0	16.4	16.2
$q = 1.03$	15.9	11.4	9.5	8.7	8.3	8.1	17.9	13.4	11.5	10.7	10.3	10.1	24.0	19.5	17.6	16.7	16.3	16.2
$q = 1.02$	12.9	10.1	8.9	8.4	8.2	8.0	14.9	12.1	10.9	10.4	10.2	10.0	20.9	18.2	17.0	16.5	16.2	16.1
$q = 1.01$	10.1	8.9	8.4	8.2	8.1	8.0	12.1	11.0	10.4	10.2	10.1	10.0	18.2	17.0	16.5	16.2	16.1	16.1

Table 1. The base-2 logarithm of the largest representable value of an 8-bit, 10-bit, or 16-bit general Csűrös counter for various values of q and M, truncated to one decimal place. A counter for which this value is k is comparable in range to an ordinary k-bit integer counter. Remember that a Morris counter is the same as a Csűrös counter with $M = 1$, and a binary counter is simply a general counter with $q = 2$.

9. Adding Counters of Different Types

If counter X is of type T_1 (with transition function τ_1, estimation function f_1, and representation-finding function ϕ_1) and we wish to add into it a counter Z of type T_2 (with estimation function f_2), our general algorithm for the *add* operation accommodates this easily:

```
1: procedure add(var $X$: $T_1$, $Z$: $T_2$)
2:     let $v \leftarrow f_1(X)$
3:     let $w \leftarrow f_2(Z)$
4:     let $S \leftarrow v + w$
5:     let $K \leftarrow \phi_1(S)$
6:     let $V \leftarrow f_1(K)$
7:     let $W \leftarrow f_1(\tau_1(K))$
8:     let $\Delta \leftarrow \frac{S-V}{W-V}$
9:     if $random() < \Delta$ then $X \leftarrow \tau_1(K)$ else $X \leftarrow K$
```

Whether a specific optimized version is worthwhile will depend on the two specific choices of counter representation.

It is even possible to add two counters of different types to produce a result in a third representation T_3 (with transition function τ_3, estimation function f_3, and representation-finding function ϕ_3):

```
1: procedure add($X$: $T_1$, $Z$: $T_2$): $T_3$
2:     let $v \leftarrow f_1(X)$
3:     let $w \leftarrow f_2(Z)$
4:     let $S \leftarrow v + w$
5:     let $K \leftarrow \phi_3(S)$
6:     let $V \leftarrow f_3(K)$
7:     let $W \leftarrow f_3(\tau_3(K))$
8:     if $random() < \frac{S-V}{W-V}$ then return $\tau_3(K)$ else return $K$
```

At this time we do not know of any specific practical application for this ability.

10. Discussion

Using approximate counters (perhaps with addition) is easy: they are a direct replacement for increment-only integer counters. The API is trivial: *read*, *increment*, and perhaps *add* operations, plus a way to initialize them to zero. The question is under what circumstances (that is, in what sort of algorithm) it is appropriate to do such a replacement. We speak to that briefly in the last paragraph of Section 12.

Approximate counters deliver only a statistical estimate of the true number of increment operations. The standard deviation of this estimate is the square root of the variance (this is the very definition of "standard deviation"). Using the standard "68-95-99.7 Rule," we can expect the statistical estimate to be less than one standard deviation away from the true value about 68% of the time, to be less than two standard deviations away from the true value about 95% of the time, and to be less than three standard deviations away from the true value "nearly always" (about 99.7% of the time). By our Theorem 1, the variance of a general Morris counter is no larger than $\frac{q-1}{2}n(n-1)+\rho$ (where $\frac{1}{6} \leq \rho < \frac{1}{4}$), so the standard deviation is the square root of that, and for large n this is approximately $\big(\sqrt{(q-1)/2}\big)n$. As an example, for $q = 1.1$, the standard deviation is approximately $0.22n$, so we can expect a counter estimate to be within 22% of the true value 68% of the time, to be within about 44% of the true value 95% of the time, and to be within about 66% of the true value "nearly always."

Csűrös counters and Morris counters have similar behavior when the number of increments is large (which is not surprising, because Morris counters are the special case of Csűrös counters with $M = 1$). An advantage of Csűrös counters over Morris counters is that they are exact until the number of increments reaches M, but there is no free lunch: the tradeoff is that in exchange for achieving a variance of exactly 0 for small values of the counter, the upper bound on the variance of larger values of the counter is slightly worse. As an example, consider an 8-bit general Csűrös counter with $M = M_{\mathrm{C}} = 8$ and $q = q_{\mathrm{C}} = 1.2$; the largest representable value is $\big(\frac{8}{1.2-1} + (255 \bmod 8)\big)(1.2)^{\lfloor 255/8 \rfloor} - \frac{8}{1.2-1} = \big(40+7\big)(1.2)^{31} - 40 \approx 13348.02$. An 8-bit general Morris counter described by $q = q_{\mathrm{M}}$ whose highest representable value is the same must satisfy $\frac{q_{\mathrm{M}}^{255}-1}{q_{\mathrm{M}}-1} \approx 13348.02$; solving for q_{M} gives $q_{\mathrm{M}} \approx 1.022667$. So the coefficient of $n(n-1)$ in the formula given by Theorem 3 for the upper bound on the variance of the Csűrös counter is $\frac{1}{2\mu_C} = \frac{q_{\mathrm{C}}-1}{2M_{\mathrm{C}}} = \frac{0.2}{16} = 0.0125$, but the coefficient of $n(n-1)$ in the formula given by Theorem 1 for the upper bound on the variance of the Morris counter is only $\frac{q_{\mathrm{M}}-1}{2} = \frac{0.022667}{2} = 0.011333$, slightly smaller than the coefficient for the Csűrös counter.

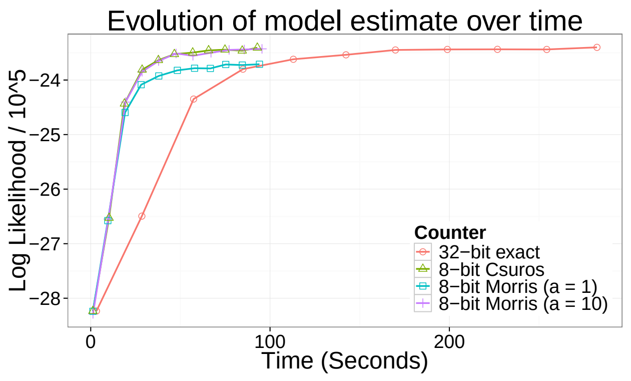

Figure 1. 32-bit integer counters vs. 8-bit approximate counters used in a distributed (multiple-GPU) application (LDA Gibbs topic modeling) coded in CUDA using MPI, compared by log likelihood

An advantage of a binary Csűrös counter over a non-binary Morris counter is that the incrementation code is cheaper. However, binary Csűrös counters support only a limited choice of maximum counter value. If your application happens to have a maximum counter value that well matches one of the possibilities for binary Csűrös counters, then that is a good choice. Otherwise, one might be better off using a general Morris counter with an appropriately calculated value for q, unless there is a specific desire to have small counter values be exact, in which case a general Csűrös counter may be a better choice, where one first chooses the necessary M and then calculates the appropriate value for q.

Table 1 shows the base-2 logarithm of the largest representable value of an 8-bit, 10-bit, or 16-bit general Csűrös counter for various values of q and M, truncated to one decimal place. (One can pack six 10-bit counters into a 64-bit word.) This table is useful for getting an idea of what parameters might be appropriate for an approximate counter intended to replace a k-bit ordinary integer counter. For example, if an application uses 64-bit integer counters but there is reason to believe that no counter value ever exceeds 2^{40}, then it is reasonable to look for values just above 40 in the table. If we wish to use an 8-bit counter, then $M = 1$ (a general Morris counter) and $q = 1.11$ may be about right; we can also see that no 8-bit counter works for $M \geq 8$, but $M = 4$ and $q = 1.5$ is a possibility. If we are willing to use a 16-bit counter, then a binary Csűrös counter ($q = 2$) with $M = 2048$ should easily suffice, so this may be a better choice than $q = 1.08$ and $M = 256$.

11. Measurements of Two Applications

We implemented a distributed version of the LDA Gibbs topic-modeling algorithm described by Tristan *et al.* [13] on a cluster of four nodes (connected by 1Gb/s Ethernet), each with an Intel Core-i7 4820k CPU and two NVIDIA Titan Black GPU cards, for a total of 8 GPUs. The code is written in C++ and CUDA. We use MPI for internode communication. The behavior is quite similar to that of the single-GPU version reported by Tristan *et al.*: replacing integer counters with approximate counters does not affect the statistical performance of the algorithm, but allows the same hardware to process larger datasets. Moreover, the speed of the algorithm is markedly increased, largely because of a third effect: less data needs to be pushed through the network connecting the nodes. In one test, either 32-bit integer counters were used, or 8-bit approximate counters; using approximate counters improved the overall speed of each iteration of the distributed application by over 65% (see Figure 1, which shows measurements of time and log likelihood). Using 8-bit general Morris counters with $a = 10$ (that is, $q = \frac{11}{10} = 1.1$) worked well, in that topics were produced using the same number of iterations as with 32-bit integer counters, without

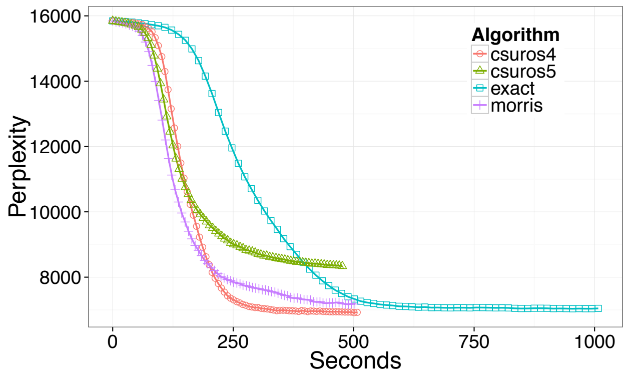

Figure 2. 32-bit integer counters vs. 8-bit approximate counters used in a distributed (multiple-CPU) application (SCA topic modeling) coded in Java using MPI, compared by perplexity

decreasing the log-likelihood measure of statistical performance. (Because the maximum counts for this particular dataset are not large, we could have used an even smaller value of q, say 1.03, to achieve smaller variance, but we found that $q = 1.1$ provided sufficient statistical quality and moreover was sufficient to handle the much larger counts, up to 2^{32} and beyond, of larger datasets.) Also effective were 8-bit binary Csűrös counters with $s = 5$. However, using 8-bit binary Morris counters ($a = 1$) caused the algorithm to converge to a smaller (less desirable) value of log likelihood; we conjecture that this occurs because an 8-bit binary Morris counter has huge dynamic range, much of which is wasted, so the part of the range that is actually used is too coarse to be fully effective (put another way, the variance is too large).

We have also tested a distributed implementation of a stochastic cellular automaton (SCA) for topic modeling as described by Zaheer *et al.* [15, 16]. This is an alternative approach to topic modeling that keeps in memory only the sufficient statistics θ (topics per document) and ϕ (words per topic); the values of the latent variables z that actually specify the assignment of topics to words are computed on the fly but not stored into memory except during a final output pass, during which they are streamed to an output file rather than kept in memory. This approach greatly reduces the memory footprint of the algorithm. In a distributed implementation, only values of the ϕ array need to be exchanged between processors; representing ϕ using approximate counters further reduces the memory footprint and moreover reduces the communication traffic.

We tested a version of the SCA algorithm implemented in the Java programming language. To achieve good performance, we use only arrays of primitive types and pre-allocate all arrays before learning starts. We implement multithreaded parallelization within a node using the work-stealing Fork/Join framework (tasks are recursively subdivided until the number of data points to be processed is less than 10^5). Internode communication uses the Java binding to OpenMPI. We use a sparse array representation for the counts of topics per documents and use Walker's alias method [14] to draw from discrete distributions. We run our experiments on a small cluster of 16 nodes (connected by 10Gb/s Ethernet), each with two 8-core Intel Xeon E5 processors (some nodes have Ivy Bridge processors while others have Sandy bridge processors) for a total of 32 hardware threads per node and 256GB of memory. We run 32 JVM instances (one per Xeon socket), assigning each one 20 gigabytes of memory. The number of topics (K) is 100, the number of training iterations is 75, and $\alpha = \beta = 0.1$. We use two datasets, both of which are cleaned by removing stop words and rare words: we use the English Wikipedia dataset for training and the Reuters RCV1 dataset for testing. Our Wikipedia dataset has

6,749,797 documents comprising 6,749,797 tokens with a vocabulary of 291,561 words. Our Reuters dataset has 806,791 documents comprising 105,989,213 tokens with a vocabulary of 43,962 words.

Time and perplexity measurements of the SCA algorithm are shown in Figure 2. Versions that use 8-bit approximate counters (Morris with $q = 1.08$, and Csűrös with either $s = 4$ or $s = 5$) are approximately twice as fast as the baseline that uses 32-bit integer counters. Csűrös counters with $s = 4$ achieved the best (lowest) perplexity in these tests. Csűrös counters with $s = 5$ do not have enough range for the application, and we observe that its asymptotic perplexity score is markedly worse.

12. Why Hasn't This Been Done Before?

When we set out to test a distributed version of the single-GPU LDA Gibbs algorithm with approximate counters, we recognized the need to replicate the counters—simple tests showed that trying to increment counters stored on a remote node would result in much slower performance—and therefore the need to add approximate counter values. We thought it would be easy to locate the necessary algorithm in the literature; approximate counters have been around for almost four decades, and it's an "obvious" operation to provide. But our best efforts uncovered no mention at all of this operation.

We speculate that the need simply has not arisen until now, and offer this "just-so" story: If counters are stored in a central memory, then it never makes sense to use a replicated representation for the counters; if one can afford to store two copies of an 8-bit approximate counter, then one is better off using one 16-bit approximate counter, affording greater range or precision or both. So adding approximate counters makes sense only in a distributed setting. But most uses of approximate counters in the literature have had to do with counting events (such as performance counters in a hardware processor); it does make sense, for example, for every processor in a cluster to have its own counters, but typically one clears the counters, runs a computation, and then gathers and aggregates the performance data just once, after the computation has completed. For database applications, past use has typically focused on counting features while making a single (possibly distributed) pass over the database. In such cases there is nothing to be gained by reducing the aggregated values back to the approximate-counter representation; rather, one communicates the approximate counter values, expands each to a full integer, and then sums the integers, and that's it.

What motivates addition of approximate counters is an *iterative* distributed application that can use replicated approximate counters within each iteration, where each iteration also includes the aggregation and redistribution of such replicated counters. We found that machine-learning algorithms (such as topic modeling) and stochastic cellular automata fit this description and benefit accordingly. We admit that one benefit of adding approximate counters in such applications, namely the reduction in network traffic, could be had in other ways, such as applying a generic data-compression algorithm to an array of ordinary integer counters. However, the addition process is fairly quick, and there may be other benefits to maintaining intermediate values in approximate-counter form.

13. Conclusions and Future Work

Statistically independent approximate counters can be added, producing a result in the same representation, so that the expected value of the result is the sum of the expected values of the operands, and the variance of the result is bounded. We present specific novel algorithms for adding five kinds of approximate counters in the literature; for three of them (general Morris, binary Morris, and Csűrös) we present proofs of bounded variance. We report that replacing integer counters with approximate counters maintains the statistical behavior of a distributed multiple-GPU implementation of a machine-learning application while improving its overall performance by almost a factor of 3, and of a distributed multiple-CPU implementation of another machine-learning application while improving its overall performance by almost a factor of 2.

It remains to produce proofs of bounded variance for the other two kinds of approximate counters (DLM probability and DLM floating-point) and to investigate what other sorts of applications might benefit from adding approximate counters.

Acknowledgments

We thank Yossi Lev for discussion of this material and for pointing us to important related work. We also thank Daniel Ford and the anonymous referees for helpful comments.

References

[1] A. C. Callahan. Random rounding: Some principles and applications. In *ICASSP '76: IEEE Intl. Conf. Acoustics, Speech, and Signal Processing*, volume 1, pages 501–504, Apr 1976.

[2] Miklós Csűrös. Approximate counting with a floating-point counter. In *COCOON '10: Proc. 16th Annual International Conf. Computing and Combinatorics*, pages 358–367, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] Andrej Cvetkovski. An algorithm for approximate counting using limited memory resources. In *SIGMETRICS '07: Proc. 2007 ACM SIGMETRICS International Conf. Measurement and Modeling of Computer Systems*, pages 181–190, New York, 2007. ACM.

[4] Dave Dice, Yossi Lev, and Mark Moir. Scalable statistics counters. In *PPoPP '13: Proc. 18th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, pages 307–308, New York, 2013. ACM.

[5] Philippe Flajolet. Approximate counting: A detailed analysis. *BIT Numerical Mathematics*, 25(1):113–134, 1985.

[6] George E. Forsythe. Reprint of a note on rounding-off errors. *SIAM Review*, 1(1):66–67, 1959. dx.doi.org/10.1137/1001011.

[7] Scott A. Mitchell and David M. Day. Flexible approximate counting. In *IDEAS '11: Proc. 15th Symp. International Database Engineering & Applications*, pages 233–239, New York, 2011. ACM.

[8] Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, October 1978.

[9] D. S. Parker, B. Pierce, and P. R. Eggert. Monte Carlo arithmetic: How to gamble with floating point and win. *Computing in Science Engineering*, 2(4):58–68, July 2000.

[10] D. Stott Parker. Monte Carlo arithmetic: Exploiting randomness in floating-point arithmetic. Technical Report 970002, Computer Science Department, UCLA, Los Angeles, CA, March 1997. http://fmdb.cs.ucla.edu/Treports/970002.pdf.

[11] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI—The Complete Reference: Volume 1, The MPI Core*. MIT Press, Cambridge, Massachusetts, second edition, 1998.

[12] Rade Stanojevic. Small active counters. In *26th IEEE International Conf. Computer Communications (INFOCOM 2007)*, pages 2153–2161, May 2007.

[13] Jean-Baptiste Tristan, Joseph Tassarotti, and Guy L. Steele Jr. Efficient training of LDA on a GPU by Mean-For-Mode Gibbs sampling. In *ICML 2015: 32nd International Conf. Machine Learning*, volume 37, July 2015. Volume 37 of the Journal in Machine Learning Research: Workshop and Conference Proceedings.

[14] A. J. Walker. Fast generation of uniformly distributed pseudorandom numbers with floating-point representation. *Electronics Letters*, 10(25):533–534, December 1974.

[15] Manzil Zaheer, Michael Wick, Jean-Baptiste Tristan, Alex Smola, and Guy L. Steele Jr. Exponential stochastic cellular automata for massively parallel inference. In *LearningSys: Workshop on Machine Learning Systems at Neural Information Processing Systems (NIPS)*, December 2015. http://learningsys.org/papers/LearningSys_2015_paper_11.pdf.

[16] Manzil Zaheer, Michael Wick, Jean-Baptiste Tristan, Alex Smola, and Guy L. Steele Jr. Exponential stochastic cellular automata for massively parallel inference. In *AISTATS: 19th Intl. Conf. Artificial Intelligence and Statistics*, May 2016. To appear.

Appendix: Proofs of Bounded Variance

Lemma 1. [generalized from Morris [8]] *Let X be a general Morris approximate counter whose expected value is n, and let X' be the result of applying an increment operation to that counter. Let κ be an arbitrary constant. Assume* $\mathrm{Var}\left(f(X)\right) \le \frac{q-1}{2}n(n-1)+\kappa$*; then* $\mathrm{Var}\left(f(X')\right) \le \frac{q-1}{2}(n+1)n+\kappa$*. Alternatively, assume* $\mathrm{Var}\left(f(X)\right) = \frac{q-1}{2}n(n-1)+\kappa$*; in that case,* $\mathrm{Var}\left(f(X')\right) = \frac{q-1}{2}(n+1)n+\kappa$*.*

Proof. Let $\Delta = q^{-X} = \frac{q-1}{(q^{X+1}-1)-(q^X-1)} = \frac{1}{f(X+1)-f(X)}$.

$$\begin{aligned}
&\mathrm{Var}\left(f(X')\right)\\
&= \mathbb{E}\left[\left(f(X')\right)^2\right] - \left(\mathbb{E}\left[f(X')\right]\right)^2\\
&= \mathbb{E}\left[(1-\Delta)\left(f(X)\right)^2 + \Delta\left(f(X+1)\right)^2\right] - (n+1)^2\\
&= \mathbb{E}\left[\left(f(X)\right)^2 + \Delta\left(\left(f(X+1)\right)^2 - \left(f(X)\right)^2\right)\right] - (n+1)^2\\
&= \mathbb{E}\left[\left(f(X)\right)^2\right] + \mathbb{E}\left[f(X+1)+f(X)\right] - (n+1)^2\\
&= \mathbb{E}\left[\left(f(X)\right)^2\right] + \mathbb{E}\left[\tfrac{q((q-1)f(X)+1)-1}{q-1} + f(X)\right] - (n+1)^2\\
&= \left(\mathrm{Var}\left(f(X)\right) + \left(\mathbb{E}\left[f(X)\right]\right)^2\right) + \mathbb{E}\left[(q+1)f(X)+1\right]\\
&\quad - (n+1)^2\\
&= \left(\mathrm{Var}\left(f(X)\right) + n^2\right) + (q+1)n + 1 - (n+1)^2\\
&\le \tfrac{q-1}{2}n(n-1) + \kappa + n^2 + (q+1)n + 1 - (n+1)^2\\
&= \tfrac{q-1}{2}n(n+1) + \kappa
\end{aligned}$$

with equality holding when $\mathrm{Var}\left(f(X)\right) = \frac{n(n-1)}{2} + \kappa$ holds. □

Lemma 2. *Given values q and S such that $1 < q \le 2$ and $S \ge 0$, let $K = \lfloor \log_q((q-1)S+1)\rfloor$, $V = \frac{q^K-1}{q-1}$, $W = \frac{q^{K+1}-1}{q-1}$, and $A = (S-V)(W-S)$. Then $A \le \frac{((q-1)S+1)^2}{4q}$.*

Proof. From the definition of the floor operation $\lfloor\cdot\rfloor$, we have $K = \log_q((q-1)S+1) - r$ for some $0 \le r < 1$. If we fix q and S and view V as a function of r, it is easy to see that it is monotonic; if we then compute $L = V_{r=1} = \frac{q^{-1}((q-1)S+1)-1}{q-1}$ it follows that $L < V \le S$.

We will now recharacterize V as an element of the range $(L, S]$ using a more convenient parameter $\delta = \frac{V-L}{S-L}$ such that $0 < \delta \le 1$. We write V in terms of S, q, and δ as $V = L + \delta(S-L) = q^{-1}(\delta((q-1)S+1) + S - 1)$ and $W = \frac{q((q-1)V+1)-1}{q-1} = \delta((q-1)S+1)+S$. Then we have $A = (S-V)(W-S) = \delta(1-\delta)q^{-1}((q-1)S+1)^2$.

Regard A as a function of δ and ask what value of δ maximizes it; the answer, easily derived by solving $\frac{d}{d\delta}\left(\delta(1-\delta)\right) = 0$, is $\delta = \frac{1}{2}$, and the second derivative there is negative, so the maximum possible value for A is $\frac{1}{2}(1-\frac{1}{2})q^{-1}((q-1)S+1)^2$, and therefore for any value of δ in $(0, 1]$ we have $A \le \frac{((q-1)S+1)^2}{4q}$. □

Lemma 3. *Let X be a general Morris approximate counter with parameter q ($1 < q \le 2$) whose expected value is n, let Z be a general Morris approximate counter with parameter q whose expected value is m, and assume that X and Z are statistically independent. Let $\rho = \frac{1}{-2(q^2-4q+1)}$, and also assume that* $\mathrm{Var}\left(f(X)\right) \le \frac{q-1}{2}n(n-1)+\rho$ *and that* $\mathrm{Var}\left(f(Z)\right) \le \frac{q-1}{2}m(m-1)+\rho$*; then* $\mathrm{Var}\left(f(X\oplus Z)\right) \le \frac{q-1}{2}(n+m)((n+m)-1)+\rho$*.*

Proof. If $Z = 0$, then $m = \mathbb{E}\left[f(Z)\right] = 0$, $w = 0$, $S = v$, $V = v$, and $\Delta = 0$; therefore $f(X\oplus Z) = f(X)$ exactly, so $\mathrm{Var}\left(f(X\oplus Z)\right) = \mathrm{Var}\left(f(X)\right) \le \frac{q-1}{2}n(n-1) \le \frac{q-1}{2}(n+m)(n+m-1)+\rho$, as desired. Similarly, if $X = 0$, then $n = \mathbb{E}\left[f(Z)\right] = 0$, $v = 0$, $S = w$, $V = w$, and $(1-\Delta) = 0$; therefore $f(X\oplus Z) = f(Z)$ exactly, so $\mathrm{Var}\left(f(X\oplus Z)\right) = \mathrm{Var}\left(f(Z)\right) \le \frac{q-1}{2}m(m-1) \le \frac{q-1}{2}(n+m)(n+m-1)+\rho$.

Now suppose $X > 0$ and $Z > 0$ (therefore $n = \mathbb{E}\left[f(X)\right] \ge 1$ and $m = \mathbb{E}\left[f(Z)\right] \ge 1$):

$$\begin{aligned}
&\mathrm{Var}\left(f(X\oplus Z)\right)\\
&= \mathbb{E}\left[\left(f(X\oplus Z)\right)^2\right] - \left(\mathbb{E}\left[f(X\oplus Z)\right]\right)^2\\
&= \mathbb{E}\left[(1-\Delta)V^2 + \Delta W^2\right] - \mathbb{E}\left[S^2\right]\\
&= \mathbb{E}\left[\tfrac{W-S}{W-V}V^2 + \tfrac{S-V}{W-V}W^2 - S^2\right]\\
&= \mathbb{E}\left[(S-V)(W-S)\right]\\
&\le \mathbb{E}\left[\tfrac{((q-1)S+1)^2}{4q}\right] \qquad\text{[by Lemma 2]}\\
&= \mathbb{E}\left[\tfrac{(q-1)^2S^2+2(q-1)S+1}{4q}\right]\\
&= \mathbb{E}\left[\tfrac{(q-1)^2}{4q}\left(f(X)+f(Z)\right)^2 + \tfrac{q-1}{2q}\left(f(X)+f(Z)\right) + \tfrac{1}{4q}\right]\\
&= \tfrac{(q-1)^2}{4q}\left(\mathbb{E}\left[\left(f(X)\right)^2\right] + \mathbb{E}\left[2f(X)f(Z)\right] + \mathbb{E}\left[\left(f(Z)\right)^2\right]\right)\\
&\quad + \tfrac{q-1}{2q}\mathbb{E}\left[f(X)+f(Z)\right] + \tfrac{1}{4q}\\
&= \tfrac{(q-1)^2}{4q}\left(\mathrm{Var}\left(f(X)\right) + \left(\mathbb{E}\left[f(X)\right]\right)^2\right.\\
&\qquad + \mathbb{E}\left[2f(X)f(Z)\right]\\
&\qquad \left.+ \mathrm{Var}\left(f(Z)\right) + \left(\mathbb{E}\left[f(Z)\right]\right)^2\right)\\
&\quad + \tfrac{q-1}{2q}\left(\mathbb{E}\left[f(X)+f(Z)\right]\right) + \tfrac{1}{4q}\\
&= \tfrac{(q-1)^2}{4q}\left(\mathrm{Var}\left(f(X)\right) + n^2 + 2nm + \mathrm{Var}\left(f(Z)\right) + m^2\right)\\
&\quad + \tfrac{q-1}{2q}(n+m) + \tfrac{1}{4q}\\
&= \tfrac{(q-1)^2}{4}\left(\mathrm{Var}\left(f(X)\right) + \mathrm{Var}\left(f(Z)\right) + (n+m)^2\right)\\
&\quad + \tfrac{q-1}{2q}(n+m) + \tfrac{1}{4q}\\
&\le \tfrac{(q-1)^2}{4q}\left(\tfrac{q-1}{2}n(n-1) + \rho + \tfrac{q-1}{2}m(m-1) + \rho + (n+m)^2\right)\\
&\quad + \tfrac{q-1}{2q}(n+m) + \tfrac{1}{4q}\\
&= \tfrac{(q-1)(q^2-1)}{8q}(n^2+m^2) + \tfrac{(q-1)^2}{4q}(2nm)\\
&\quad - \tfrac{(q-1)(q+1)(q-3)}{8q}(n+m) + \tfrac{(q-1)^2}{2q}\rho + \tfrac{1}{4q}\\
&= \tfrac{(q-1)(q^2-1)}{8q}(n^2+m^2) + \tfrac{(q-1)^2}{4q}(2nm)\\
&\quad - \tfrac{(q-1)(q+1)(q-3)}{8q}(n+m) + \rho
\end{aligned}$$

We wish to show that this last quantity is less than or equal to $\frac{q-1}{2}(n+m)(n+m-1)+\rho$; we do so by showing that $\frac{8q}{q-1}$ times their difference is nonpositive (note that $\frac{8q}{q-1} > 0$).

$$\begin{aligned}
&\tfrac{8q}{q-1}\Big(\tfrac{(q-1)(q^2-1)}{8q}(n^2+m^2) + \tfrac{(q-1)^2}{4q}(2nm)\\
&\qquad - \tfrac{(q-1)(q+1)(q-3)}{8q}(n+m) + \rho\\
&\qquad - \left(\tfrac{q-1}{2}(n+m)(n+m-1)+\rho\right)\Big)\\
&= (q^2-4q-1)n^2 - (q^2-6q-3)n - 2(q+1)nm +\\
&\quad (q^2-4q-1)m^2 - (q^2-6q-3)m - 2(q+1)nm
\end{aligned}$$

We now need only show that

$$(q^2-4q-1)n^2 - (q^2-6q-3)n - 2(q+1)nm \le 0$$

and

$$(q^2 - 4q - 1)m^2 - (q^2 - 6q - 3)m - 2(q + 1)nm \leq 0$$

The two proofs are identical in form; here we present just one of the proofs. Because $2(q + 1)n > 0$ and $m \geq 1$,

$$\begin{aligned}
&(q^2 - 4q - 1)n^2 - (q^2 - 6q - 3)n - 2(q + 1)nm \\
&\leq (q^2 - 4q - 1)n^2 - (q^2 - 6q - 3)n - (2q + 2)n \\
&= (q^2 - 4q - 1)n^2 - (q^2 - 4q - 1)n \\
&= (q^2 - 4q - 1)n(n - 1) \leq 0
\end{aligned}$$

because $n(n - 1) \geq 0$ for all $n \geq 1$ and $q^2 - 4q - 1 < 0$ for all $1 < q \leq 2$ (the roots of $q^2 - 4q - 1 = 1$ are $2 - \sqrt{5} \approx -0.236\ldots$ and $2 + \sqrt{5} \approx 4.236\ldots$). □

Theorem 1. *Let $1 < q \leq 2$, let $\rho = \frac{1}{-2(q^2-4q+1)}$, and let X be a general Morris approximate counter with parameter q whose expected value is n. Then* $\mathrm{Var}\,(f(X)) \leq \frac{q-1}{2}n(n - 1) + \rho$.

Proof. We proceed by structural induction on n.

Basis case: Suppose that X a newly created zero counter. Then its value is definitely 0, and therefore its expected value is 0 and its variance is 0, and so $\mathrm{Var}\,(f(X)) = 0 \leq \frac{0(0-1)}{2} \leq \frac{n(n-1)}{2} + \rho$.

Inductive case 1 (incrementation): Let us suppose that X was produced by incrementing some counter Y having expected value $n - 1$. By our inductive hypothesis we have $\mathrm{Var}\,(f(Y)) \leq \frac{(n-1)(n-2)}{2} + \rho$, and therefore by Lemma 1 with $\kappa = \rho$ we have $\mathrm{Var}\,(f(X)) \leq \frac{n(n-1)}{2} + \rho$.

Inductive case 2 (addition): Suppose that X was produced by adding a counter Y with expected value y and a statistically independent counter Z with expected value z, such that $y + z = n$. By inductive hypothesis we have $\mathrm{Var}\,(f(Y)) \leq \frac{q-1}{2}y(y - 1) + \rho$ and $\mathrm{Var}\,(f(Z)) \leq \frac{q-1}{2}z(z - 1) + \rho$, and therefore by Lemma 3 we have $\mathrm{Var}\,(f(X)) \leq \frac{q-1}{2}n(n - 1) + \rho$.

Therefore in all cases $\mathrm{Var}\,(f(X)) \leq \frac{q-1}{2}n(n - 1) + \rho$. □

Lemma 4. *Let X be a binary Morris approximate counter whose expected value is n, and Z be a statistically independent binary Morris approximate counter whose expected value is m. Without loss of generality assume $m \leq n$. Assume also that* $\mathrm{Var}\,(f(X)) \leq \frac{n(n-1)}{2}$*; then we have* $\mathrm{Var}\,(f(X \oplus Z)) \leq \frac{(n+m)((n+m)-1)}{2}$. *As an alternative, assume also that $m \leq 1$ and* $\mathrm{Var}\,(f(X)) = \frac{n(n-1)}{2}$*; then* $\mathrm{Var}\,(f(X \oplus Z)) = \frac{(n+m)((n+m)-1)}{2}$.

Proof. Let $\Delta = \frac{f(Z)}{f(X+1)-f(X)}$. Then:

$$\begin{aligned}
&\mathrm{Var}\,(f(X \oplus Z)) \\
&= \mathbb{E}\left[\big(f(X \oplus Z)\big)^2\right] - \big(\mathbb{E}\left[f(X \oplus Z)\right]\big)^2 \\
&= \mathbb{E}\left[(1 - \Delta)\big(f(X)\big)^2 + \Delta\big(f(X + 1)\big)^2\right] - (n + m)^2 \\
&= \mathbb{E}\left[\big(f(X)\big)^2 + \Delta\left(\big(f(X + 1)\big)^2 - \big(f(X)\big)^2\right)\right] - (n + m)^2 \\
&= \mathbb{E}\left[\big(f(X)\big)^2\right] + \mathbb{E}\left[f(Z)\big(f(X + 1) + f(X)\big)\right] - (n + m)^2 \\
&= \big(\mathrm{Var}\,(f(X)) + (\mathbb{E}\left[f(X)\right])^2\big) \\
&\quad + \mathbb{E}\left[f(Z)\big(3f(X) + 1\big)\right] - (n + m)^2 \\
&\leq \frac{n(n - 1)}{2} + n^2 + \mathbb{E}\left[f(Z)\big(3f(X) + 1\big)\right] - (n + m)^2 \\
&= \frac{n(n - 1)}{2} + n^2 + m(3n + 1) - (n + m)^2 \\
&= \frac{n^2}{2} + nm - m^2 - \frac{n}{2} + m \\
&\leq \frac{n^2}{2} + nm - m^2 - \frac{n}{2} + m + \frac{3}{2}(m^2 - m) \\
&= \frac{(n + m)((n + m) - 1)}{2}
\end{aligned}$$

(because $\frac{3}{2}(m^2 - m) \geq 0$ for all integer $m \geq 0$), with equality holding when $m \leq 1$ and $\mathrm{Var}\,(f(X)) = \frac{n(n-1)}{2}$ (because we have $\frac{3}{2}(m^2 - m) = 0$ when $m = 0$ or $m = 1$). □

Theorem 2. *Let X be a binary Morris approximate counter whose expected value is n. Then* $\mathrm{Var}\,(f(X)) \leq \frac{n(n-1)}{2}$.

Proof. We proceed by structural induction on n.

Basis case: Suppose that X a newly created zero counter. Then its value is definitely 0, and therefore its expected value is 0 and its variance is 0, and so $\mathrm{Var}\,(f(X)) = 0 \leq \frac{0(0-1)}{2} = \frac{n(n-1)}{2}$.

Inductive case 1 (incrementation): Let us suppose that X was produced by incrementing counter Y having expected value $n - 1$. By our inductive hypothesis we have $\mathrm{Var}\,(f(Y)) \leq \frac{(n-1)(n-2)}{2}$, and so by Lemma 1 with $\kappa = 0$ we have $\mathrm{Var}\,(f(X)) \leq \frac{n(n-1)}{2}$.

Inductive case 2 (addition): Suppose that X was produced by adding a counter Y with expected value y and a statistically independent counter Z with expected value z, such that $z \leq y$ and $y+z = n$. By inductive hypothesis we have $\mathrm{Var}\,(f(Y)) \leq \frac{y(y-1)}{2}$, and therefore by Lemma 4 we have $\mathrm{Var}\,(f(X)) \leq \frac{n(n-1)}{2}$.

Therefore in all cases $\mathrm{Var}\,(f(X)) \leq \frac{n(n-1)}{2}$. □

Lemma 5. *Given real $S \geq 0$, real $1 < q \leq 2$, and integer $M \geq 1$, let $\mu = \frac{M}{q-1}$, $d = \left\lfloor \log_q \frac{S+\mu}{\mu} \right\rfloor$, $\mathcal{V} = M\frac{q^d-1}{q-1}$, $\mathcal{W} = M\frac{q^{d+1}-1}{q-1}$, $\omega = \frac{\mathcal{W}-\mathcal{V}}{M}$, $j = \left\lfloor \frac{S-\mathcal{V}}{\omega} \right\rfloor$, $V = \mathcal{V} + j\omega$, $W = V + \omega$, and $A = (S - V)(W - S)$. Then $A \leq \frac{((q-1)S+M)^2}{4M(M+(q-1))} = \frac{(S+\mu)^2}{4\mu(\mu+1)}$.*

Proof. The interval $[\mathcal{V}, \mathcal{W}]$ is divided into M subintervals of width ω; subinterval j is the one that contains S. (If S falls right on the boundary between two subintervals, we can regard it as being in either; then either $S = V$ or $S = W$, and so $(S-V)(W-S) = 0$.)

If S is in subinterval j ($0 \leq j < M$), then the smallest value $\mathcal{V}_{\min}$ that $\mathcal{V}$ could have (holding q, M, and j fixed while letting S vary) occurs when S is at the upper end of subinterval j, and similarly $\mathcal{W}_{\min} = q\mathcal{V}_{\min} + M$ and $\omega_{\min} = \frac{(q-1)\mathcal{V}_{\min}+M}{M}$. But where is $\mathcal{V}_{\min}$ with respect to S? If S is at the upper end of subinterval j, then $S - \mathcal{V}_{\min} = (j + 1)\omega_{\min}$ and $\mathcal{W}_{\min} - S = (q\mathcal{V}_{\min} + M) - S = (M - (j + 1))\omega_{\min}$ similarly. (Note that $\mathcal{W}$ and ω are minimized exactly when $\mathcal{V}$ is minimized, which is what justifies using the "min" subscripts on $\mathcal{W}_{\min}$ and $\omega_{\min}$.) Eliminating $\omega_{\min}$ gives us $(M - (j + 1))(S - \mathcal{V}_{\min}) = (j + 1)((q\mathcal{V}_{\min} + M) - S)$. Solving this for $\mathcal{V}_{\min}$ gives $\mathcal{V}_{\min} = \frac{M(S-(j+1))}{M+(q-1)(j+1)} = \frac{\mu(S-(j+1))}{\mu+(j+1)}$.

Now $V_{\min}$, the lowest possible value for V (which is the lower end of the subinterval containing S), is j subinterval-widths above $\mathcal{V}_{\min}$ and so $V_{\min} = (\mathcal{V}_{\min} + j\omega_{\min}) = \frac{((q-1)jS+M(S-1)}{M+(q-1)(j+1)} = \frac{(jS+\mu(S-1)}{\mu+(j+1)}$. (Note that $V_{\min}$ corresponds to L as used in the proof of Lemma 2.) We characterize V using a parameter δ as $V = ((1 - \delta)V_{\min} + \delta S) = \frac{\delta(M+(q-1)S)+(q-1)jS+M(S-1)}{M+(q-1)(j+1)} = \frac{\delta(\mu+S)+jS+\mu(S-1)}{\mu+(j+1)}$. Now we have a formula for V in terms of S and δ (and fixed q, M, and j); in exchange for introducing the parameter δ, we have gotten rid of those pesky floor functions.

Once V has been defined in this way and regarded as the actual lower end of subinterval j, we can calculate a corresponding $\mathcal{V}_V$ that is exactly j subinterval-widths below V by solving $(V - \mathcal{V}_V)(M - j) = ((q\mathcal{V}_V + M) - V)j$ to get the result $\mathcal{V}_V = \frac{M(\delta(M+(q-1)S)-(q-1)j^2+j(q-1)(S-1)+MS-(j+1)M)}{(M+(q-1)j)(M+(q-1)(j+1))} =$

$\frac{\mu(\delta(\mu+S)-j^2+j(S-1)+\mu S-(j+1)\mu)}{(\mu+j)(\mu+(j+1))}$. The width of each subinterval is $\omega_V = \frac{(q-1)\mathcal{V}_V+M}{M}$ and therefore we have $W = V + \omega_V = S + \delta\frac{M+(q-1)S}{M+(q-1)j} = S + \delta\frac{\mu+S}{\mu+j}$.

Then $(S-V)(W-S) = \delta(1-\delta)\frac{((q-1)S+M)^2}{(M+(q-1)j)(M+(q-1)(j+1))}$. We now have $(S-V)(W-S)$ in terms of S and δ and q and M and j. Now switch gears: hold S and q and M fixed, and allow δ and j to vary. Then $(S-V)(W-S)$ is maximal when $\delta = \frac{1}{2}$ and $(\mu+j)(\mu+(j+1))$ is minimal. Now $(2\mu+1)$ is positive for $1 < q \leq 2$ and $M \geq 1$; therefore for $j \geq 0$, $(\mu+j)(\mu+(j+1)) = j^2 + (2\mu+1)j + \mu(\mu+1)$ is minimal when $j = 0$, and so $A = (S-V)(W-S) \leq \frac{((q-1)S+M)^2}{4M(M+(q-1))} = \frac{(S+\mu)^2}{4\mu(\mu+1)}$. □

Lemma 6. *Let X be a Csűrös approximate counter with integer parameter M ($M \geq 1$) and real parameter q ($1 < q \leq 2$) whose expected value is n, let Z be a Csűrös approximate counter with the same parameters M and q whose expected value is m, and assume that X and Z are statistically independent. Let $\mu = \frac{M}{q-1}$ and $\rho = \frac{\mu^2}{4\mu^2+4\mu-2}$, and assume that* $\mathrm{Var}\left(f(X)\right) \leq \frac{1}{2\mu}n(n-1)+\rho$ *and that* $\mathrm{Var}\left(f(Z)\right) \leq \frac{1}{2\mu}m(m-1)+\rho$*; then* $\mathrm{Var}\left(f(X \oplus Z)\right) \leq \frac{1}{2\mu}(n+m)((n+m)-1)+\rho$.

Proof. As in the proof of Lemma 3, if $Z = 0$, then $m = \mathbb{E}\left[f(Z)\right] = 0$, $w = 0$, $S = v$, $V = v$, and $\Delta = 0$; therefore $f(X \oplus Z) = f(X)$, so $\mathrm{Var}\left(f(X \oplus Z)\right) = \mathrm{Var}\left(f(X)\right) \leq \frac{1}{2\mu}n(n-1) \leq \frac{1}{2\mu}(n+m)(n+m-1)+\rho$, as desired. Similarly, if $X = 0$, then $n = \mathbb{E}\left[f(Z)\right] = 0$, $v = 0$, $S = w$, $V = w$, and $(1-\Delta) = 0$; therefore $f(X \oplus Z) = f(Z)$, so $\mathrm{Var}\left(f(X \oplus Z)\right) = \mathrm{Var}\left(f(Z)\right) \leq \frac{1}{2\mu}m(m-1) \leq \frac{1}{2\mu}(n+m)(n+m-1)+\rho$.

Now suppose $X > 0$ and $Z > 0$ (therefore $n = \mathbb{E}\left[f(X)\right] \geq 1$ and $m = \mathbb{E}\left[f(Z)\right] \geq 1$):

$$
\begin{aligned}
&\mathrm{Var}\left(f(X \oplus Z)\right) \\
&= \mathbb{E}\left[(S-V)(W-S)\right] && \text{[as in proof of Lemma 3]} \\
&\leq \mathbb{E}\left[\tfrac{(S+\mu)^2}{4\mu(\mu+1)}\right] && \text{[by Lemma 5]} \\
&= \mathbb{E}\left[\tfrac{S^2+2\mu S+\mu^2}{4\mu(\mu+1)}\right] \\
&= \tfrac{1}{4\mu(\mu+1)}\left(\mathbb{E}\left[\left(f(X)\right)^2\right] + \mathbb{E}\left[2f(X)f(Z)\right] + \mathbb{E}\left[\left(f(Z)\right)^2\right]\right) \\
&\quad + \tfrac{1}{2(\mu+1)}\mathbb{E}\left[f(X)+f(Z)\right] + \tfrac{\mu}{4(\mu+1)} \\
&= \tfrac{1}{4\mu(\mu+1)}\big(\mathrm{Var}\left(f(X)\right) + (\mathbb{E}\left[f(X)\right])^2 \\
&\qquad + \mathbb{E}\left[2f(X)f(Z)\right] \\
&\qquad + \mathrm{Var}\left(f(Z)\right) + (\mathbb{E}\left[f(Z)\right])^2\big) \\
&\quad + \tfrac{1}{2(\mu+1)}\big(\mathbb{E}\left[f(X)+f(Z)\right]\big) + \tfrac{\mu}{4(\mu+1)} \\
&= \tfrac{1}{4\mu(\mu+1)}\big(\mathrm{Var}\left(f(X)\right) + n^2 + 2nm + \mathrm{Var}\left(f(Z)\right) + m^2\big) \\
&\quad + \tfrac{1}{2(\mu+1)}(n+m) + \tfrac{\mu}{4(\mu+1)} \\
&= \tfrac{1}{4\mu(\mu+1)}\big(\mathrm{Var}\left(f(X)\right) + \mathrm{Var}\left(f(Z)\right) + (n+m)^2\big) \\
&\quad + \tfrac{1}{2(\mu+1)}(n+m) + \tfrac{\mu}{4(\mu+1)} \\
&\leq \tfrac{1}{4\mu(\mu+1)}\big(\tfrac{n(n-1)}{2\mu} + \rho + \tfrac{m(m-1)}{2\mu} + \rho + (n+m)^2\big) \\
&\quad + \tfrac{1}{2(\mu+1)}(n+m) + \tfrac{\mu}{4(\mu+1)} \\
&= \tfrac{1}{4\mu(\mu+1)}\left(\tfrac{1}{2\mu}+1\right)(n^2+m^2) + \tfrac{1}{4\mu(\mu+1)}(2nm) + \\
&\left(\tfrac{1}{2(\mu+1)} - \tfrac{1}{4\mu^2(\mu+1)^2}\right)(n+m) + \tfrac{1}{2\mu(\mu+1)}\rho + \tfrac{\mu}{4(\mu+1)} \\
&= \tfrac{2\mu+1}{8\mu^2(\mu+1)^2}(n^2+m^2) + \tfrac{1}{4\mu(\mu+1)}(2nm) \\
&\quad + \tfrac{4\mu^2-1}{8\mu^2(\mu+1)^2}(n+m) + \rho
\end{aligned}
$$

We wish to show that this last quantity is less than or equal to $\frac{1}{2\mu}(n+m)(n+m-1)+\rho$, by showing that $8\mu^2(\mu+1)^2$ times their difference is nonpositive (note that for $1 < q \leq 2$, $\mu = \frac{M}{q-1} \geq M \geq 1$ and that $8\mu^2(\mu+1)^2 > 0$).

$$
\begin{aligned}
&8\mu^2(\mu+1)^2\Big(\tfrac{2\mu+1}{8\mu^2(\mu+1)^2}(n^2+m^2) + \tfrac{1}{4\mu(\mu+1)}(2nm) \\
&\qquad + \tfrac{4\mu^2-1}{8\mu^2(\mu+1)^2}(n+m) + \rho \\
&\qquad - \big(\tfrac{1}{2\mu}(n+m)(n+m-1)+\rho\big)\Big) \\
&= (-4\mu^2-2\mu+1)(n^2+m^2) + (8\mu^2+4\mu-1)(n+m) \\
&\quad + (-4\mu^2-2\mu)(2nm)
\end{aligned}
$$

As in the proof of Lemma 3, we will exhibit only a proof that

$$(-4\mu^2-2\mu+1)n^2 + (8\mu^2+4\mu-1)n + (-4\mu^2-2\mu)nm \leq 0$$

Because $(-4\mu^2-2\mu) < 0$ and $m \geq 1$,

$$
\begin{aligned}
&(-4\mu^2-2\mu+1)n^2 + (8\mu^2+4\mu-1)n + (-4\mu^2-2\mu)nm \\
&\leq (-4\mu^2-2\mu+1)n^2 + (8\mu^2+4\mu-1)n + (-4\mu^2-2\mu)n \\
&= (-4\mu^2-2\mu+1)n^2 + (4\mu^2+2\mu-1)n \\
&= (-4\mu^2-2\mu+1)n(n-1) \leq 0
\end{aligned}
$$

because $n(n-1) \geq 0$ for all $n \geq 1$ and $(-4\mu^2-2\mu+1) < 0$ for all $\mu \geq 1$. □

Lemma 7. *Let X be a Csűrös approximate counter with integer parameter M ($M \geq 1$) and real parameter q ($1 < q \leq 2$) whose expected value is n, let X' be the result of applying an increment operation to that counter, and let $\mu = \frac{M}{q-1}$ and $\rho = \frac{\mu^2}{4\mu^2+4\mu-2}$. Assume* $\mathrm{Var}\left(f(X)\right) \leq \frac{1}{M(M+1)}n(n-1)+\rho$*; then* $\mathrm{Var}\left(f(X')\right) \leq \frac{1}{M(M+1)}(n+1)n+\rho$.

Proof. Analysis of the *increment* and *add* algorithms presented in Section 6 shows that $increment(X)$ is equivalent in its behavior to $add(X,1)$. Moreover, the variance of a Csűrös counter with expected value 1 is 0. Therefore we can apply Lemma 6. □

Theorem 3. *Let X be a Csűrös approximate counter with integer parameter M ($M \geq 1$) and real parameter q ($1 < q \leq 2$) whose expected value is n, and let $\mu = \frac{M}{q-1}$ and $\rho = \frac{\mu^2}{4\mu^2+4\mu-2}$. Then* $\mathrm{Var}\left(f(X)\right) \leq \frac{1}{2\mu}n(n-1)+\rho$.

Proof. Basis case: Suppose that X a newly created zero counter. Then its value is definitely 0, and therefore its expected value is 0 and its variance is 0, and so $\mathrm{Var}\left(f(X)\right) = 0 \leq \frac{1}{2\mu}n(n-1)+\rho$.

Inductive case 1 (incrementation): Suppose that X was produced by incrementing a counter Y having expected value $n-1$. By inductive hypothesis we have $\mathrm{Var}\left(f(Y)\right) \leq \frac{1}{2\mu}y(y-1)+\rho$, and therefore by Lemma 7 we have $\mathrm{Var}\left(f(X)\right) \leq \frac{1}{2\mu}n(n-1)+\rho$.

Inductive case 2 (addition): Suppose that X was produced by adding a counter Y with expected value y and a counter Z with expected value z, such that $z \leq y$ and $y+z = n$. By inductive hypothesis we have $\mathrm{Var}\left(f(Y)\right) \leq \frac{1}{2\mu}y(y-1)+\rho$ and $\mathrm{Var}\left(f(Z)\right) \leq \frac{1}{2\mu}z(z-1)+\rho$, and therefore by Lemma 6 we have $\mathrm{Var}\left(f(X)\right) \leq \frac{1}{2\mu}n(n-1)+\rho$.

Therefore in all cases $\mathrm{Var}\left(f(X)\right) \leq \frac{1}{2\mu}n(n-1)+\rho$. □

Note that $\frac{1}{6} \leq \rho < \frac{1}{4}$, so the bound on the variance of Csűrös counters with addition, no matter what the values of M and q, is just as reasonably tight as the bound on the variance of Morris counters.