Approximate Shared-Memory Counting Despite a Strong Adversary

JAMES ASPNES
Department of Computer Science, Yale University
and
KEREN CENSOR
Department of Computer Science, Technion

A new randomized asynchronous shared-memory data structure is given for implementing an approximate counter that can be incremented once by each of n processes in a model that allows up to $n-1$ crash failures. For any fixed ϵ, the counter achieves a relative error of δ with high probability, at the cost of $O(((1/\delta)\log n)^{O(1/\epsilon)})$ register operations per increment and $O(n^{4/5+\epsilon}((1/\delta)\log n)^{O(1/\epsilon)})$ register operations per read. The counter combines randomized sampling for estimating large values with an expander for estimating small values. This is the first counter implementation that is sublinear the number of processes and works despite a strong adversary scheduler that can observe internal states of processes.

An application of the improved counter is an improved protocol for solving randomized shared-memory consensus, which reduces the best previously known individual work complexity from $O(n \log n)$ to an optimal $O(n)$, resolving one of the last remaining open problems concerning consensus in this model.

Categories and Subject Descriptors: D.1.3 [**Software**]: Programming Techniques—*Concurrent programming*; F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*Probabilistic algorithms*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Distributed Computing, Approximate Counting, Martingales, Expanders, Consensus

1. INTRODUCTION

Counting is a fundamental algorithmic task. Unfortunately, in an asynchronous shared-memory setting using only read/write registers, exact counting appears to be quite expensive. The main limitation is the need to avoid lost updates, where

A preliminary version of this paper appeared in Proceedings of the 20th Annual Symposium on Discrete Algorithms (SODA), 2009.
Author's address: James Aspnes, Department of Computer Science, Yale University, New Haven, CT 06520-8285, USA. Email: aspnes@cs.yale.edu. Supported in part by NSF grant CNS-0435201. Keren Censor, Department of Computer Science, Technion, Haifa, Israel. Email: ckeren@cs.technion.ac.il. Supported in part by the *Israel Science Foundation* (grant number 953/06), and by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

one incrementer overwrites values left by another. In a model with n incrementers, the simplest implementation, where each incrementer writes its increment to a separate location in an array of registers, requires n register operations to perform a read, since the reader much read every location in the array. Even with more sophisticated primitives, there are still strong lower bounds on the time- and space-complexity for exact counting in a distributed system [Fich et al. 2005; Attiya et al. 2006].

This suggests relaxing the requirements and allowing a reader of the counter to return only an approximate value. There is a substantial literature on probabilistic approximate counting in a non-adversarial, sequential context. Some early examples are given in [Morris 1978; Flajolet 1985; Flajolet and Martin 1985], where the space taken by a counter is reduced by storing only an approximation to the log of the number of increments. This involves incrementing the stored counter value only every 2^C steps on average, which is accomplished by applying increments only with probability 2^{-C}. However, in a distributed setting, a **strong adversary** that can halt processes after observing their internal states can discard these rare updates, causing the counter value to appear much smaller than it should. Solving the approximate counting problem in this model thus requires new techniques.

Formally, we consider the standard model of an asynchronous shared-memory system, where n processes communicate by reading and writing to shared **multi-writer multi-reader** registers. Timing is under the control of a strong adversary, which can cause up to $n-1$ undetected crash failures. We assume that each process can both read and increment the counter, but that increments are **one-shot**: each process increments the counter at most once, giving a bound n on the total number of increments.

Each **step** consists of some local computation, including an arbitrary number of local coin-flips (possibly biased) and one shared memory **event**, which is either a read or a write to some register. The interleaving of processes' steps is governed by a strong adversary that observes the results of the local coin-flips before scheduling the next process.

A minimal requirement on an exact distributed counter would be that a read operation always returns a value between k and K, where k is the number of increment operations that finished before the read operation started, and K is the number of increment operations that started before the read operation finished.[1] For an inexact counter, we can characterize its accuracy by how close it gets to this ideal:

Definition 1. *Let C be a counter supporting the operations* `CounterRead` *and* `CounterIncrement`. *A call to* `CounterRead` *is **δ-accurate** if its return value R satisfies $(1-\delta)k \le R \le (1+\delta)K$, where k is the number of* `CounterIncrement` *operations that finish before this call to* `CounterRead` *starts, and K is the number of* `CounterIncrement` *operations that start before this call to* `CounterRead` *finishes. Otherwise, this call to* `CounterRead` *is **δ-inaccurate**.*

Our paper makes two main contributions: an implementation of an approximate

[1]This is a weaker condition than, say, linearizability [Herlihy and Wing 1990], because it makes no consistency guarantees on the values returned by different read operations.

counter for a strong-adversary model with one-shot increments that is δ-accurate with high probability and allows reads with a cost sublinear in the number of incrementers, and an application of this counter to obtain a protocol for **randomized consensus** with n processes that requires an optimal $O(n)$ operations per process. This latter protocol completely resolves the question of the complexity of randomized consensus in a strong-adversary model, a twenty-year-old open problem.

1.1 Counting

The running time of our counter is $O(((1/\delta)\log n)^{O(1/\epsilon)})$ register operations per increment and $O(n^{4/5+\epsilon}((1/\delta)\log n)^{O(1/\epsilon)})$ register operations per read, where $\epsilon > 0$ is a parameter that can be chosen to trade off the costs of increment and read operations. The counter is specialized for the case where each process increments at most once.

A full description of the counter is given in Section 2. The essential idea is that when many increments have occurred, sampling an array of bits representing increments works despite a strong adversary as long as both incrementers and readers choose randomly which array locations they will use. The analysis of this component of the counter is complicated both by the possibility that some increments will collide (write to the same bit) and by the fact that the upper bound on the estimated counter value depends on the number of increments that start before a read finishes, a quantity that can be altered by the adversary while the read is still in progress. Nonetheless we show that this part of the counter works using a combination of Chernoff bounds on the lower-bound side and the method of bounded differences on the upper-bound side.

This still leaves the problem of what to do when there are few increments. Here the sampling component breaks down: a small sample from the array is likely to miss all the increments—thus return 0, much less than $(1-\delta)k$. Instead, we use an **expander** to determine which bits in a second array are set by each increment; since the sets of bits set by each increment don't overlap much (this is the expansion property), we can obtain a good estimate of the number of increments by simply counting all ones in the array and dividing by the degree of the expander. This estimate also lets us detect when we have exceeded the useful range of the expander-based counter and must switch to sampling. The sampling bounds apply if this occurs because the sampling component is always incremented first, and therefore a large number of increments to the expander component must be preceded by a large number of increments to the sampling component.

Although our implementation is only mildly more efficient than a linear counter, having any sublinear counter can be crucial for some applications. This is demonstrated in an application of the approximate counter in a **weak shared coin** protocol used for **randomized consensus**, which is the second contribution of this paper.

1.2 Consensus

In the consensus problem, which is a fundamental task in asynchronous systems, n processes starting with individual inputs are required to arrive at the same decision value. To prevent trivial solutions, this **agreement** condition is accompanied by a **validity** condition which requires the decision value to be the input of some process. A **termination** condition requires that eventually all processes decide.

It is well known that there is no deterministic consensus protocol in an asynchronous system, if one process may fail [Fischer et al. 1985; Herlihy 1991; Loui and Abu-Amara 1987]. However, reaching consensus becomes possible using randomization with the termination condition relaxed to hold with probability 1, while the agreement and validity properties remain the same. The complexity of solving consensus is measured by the expected number of register operations carried out by all processes (**total work**) or by any single process (**per-process** or **individual work**).

Many randomized consensus protocols were designed for the asynchronous model under a strong adversary, These protocols are all **wait-free**, which means that termination occurs even if up to $n-1$ of the processes suffer crash failures. The first of these, due to Abrahamson [Abrahamson 1988], required exponential work. Subsequent work (e.g., [Aspnes and Herlihy 1990; Aspnes 1993; Saks et al. 1991]) reduced the complexity first to a polynomial number of total operations and finally to $O(n^2 \log n)$ operations in the paper of Bracha and Rachman [Bracha and Rachman 1991]. Further progress in reducing total operations stalled at this point, although Aspnes and Waarts [Aspnes and Waarts 1996] showed that $O(n \log^2 n)$ individual work could be achieved, at the cost of a slight increase in total work. The subsequent $\Omega(n^2/\log^2 n)$ lower bound on total work of Aspnes [Aspnes 1998] (which implies an $\Omega(n/\log^2 n)$ lower bound on individual work) showed that large further improvements were unlikely, although a polylogarithmic gap between the known upper and lower bounds remained.

This gap was closed for total work by Attiya and Censor [Attiya and Censor 2007], who presented a protocol with $O(n^2)$ total work and a matching $\Omega(n^2)$ lower bound. In their protocol, however, a process running alone may have to perform all this work by itself, meaning that the individual work is also $\Theta(n^2)$. By combining techniques from the Attiya-Censor protocol with the older Aspnes-Waarts protocol, Aspnes *et al.* [Aspnes et al. 2008] obtained an $O(n \log n)$ upper bound on individual work, but a logarithmic gap still remained between this upper bound and the $\Omega(n)$ lower bound that follows from the $\Omega(n^2)$ lower bound on total work.

We eliminate this gap by showing that randomized consensus can be solved with a wait-free protocol using only atomic multi-writer multi-reader registers with $O(n)$ expected individual work and $O(n^2)$ expected total work; both measures are optimal since they match the lower bound.

The cornerstone of our protocol is a new **weak shared coin** protocol that requires from each process at most $O(n)$ operations. A weak shared coin with **agreement parameter** p [Aspnes and Herlihy 1990] is a distributed protocol with the property that (a) against any adversary strategy, with probability at least p, every process returns -1, (b) against any adversary strategy, with probability at least p, every process returns $+1$. The rest of the time, the the adversary may determine the outputs of the processes or even arrange for them to disagree.

The reduction in [Aspnes and Herlihy 1990] shows that a weak shared coin protocol with agreement parameter p, expected individual work complexity I, and expected total work complexity T, yields a consensus protocol with expected individual work complexity $O((n+I)/p)$ and expected total work complexity $O((n^2+T)/p)$.

The essential idea is to solve consensus by executing a sequence of asynchronous rounds, where in each round early-arriving processes that see only processes with the same preference keep their common preference, late-arriving processes that see agreement among these leaders adopt this common preference, and confused processes that see more than one preferred value execute the shared coin protocol (which produces agreement with the other processes and any deterministic leaders with probability at least p). This mechanism reaches agreement after $O(1/p)$ rounds on average, and the worst-case cost of each round is the sum of the cost of the shared coin protocol and an $O(n)$ overhead per process to read other processes' preferences. It follows that to achieve consensus cheaply, we want to construct a weak shared coin with constant agreement parameter and per-process work as close as possible to $O(n)$.

Essentially all known shared coins are based on random voting, with some variation in how votes are collected and how termination is detected. The method we use appeared first in the $O(n^2 \log n)$ total work protocol of Bracha and Rachman [Bracha and Rachman 1991], where each process generates votes until it observes that a predetermined threshold is reached, and afterwards collects all the votes and decides upon the sign of their sum. Typically, n^2 votes are needed, so that the majority value is not overwhelmed by selective withholding of up to $n-1$ votes by the adversary.

Detecting that the threshold has been reached is a counting problem: using a standard counter with $O(n)$-operation reads means that the threshold can be checked only occasionally without blowing up the cost. Amazingly, Bracha and Rachman showed that having each process check only after each $\Theta(n/\log n)$ votes—an amortized cost of $O(\log n)$ register operations per vote—was enough to guarantee a constant agreement parameter, even with no coordination between processes. This cost was further reduced to $O(1)$ amortized operations per vote by Attiya and Censor, who added a **termination bit** that could be used to cut off voting immediately by any process that detected termination.

A second technique is needed to reduce individual work, as in the standard voting-style protocol a single process might generate most of the $\Theta(n^2)$ votes. Our shared coin is based on the **weighted voting** approach pioneered in the $O(n \log^2 n)$ individual-work protocol of Aspnes and Waarts [Aspnes and Waarts 1996], where a process that has already cast many votes becomes impatient and starts casting votes with higher weight. Combined with the termination bit, this gives an $O(n \log n)$ protocol [Aspnes et al. 2008], but it still requires that some processes execute $\Theta(\log n)$ counter reads in some executions.

By applying our sublinear counter, we can carry out these $\Theta(\log n)$ counter reads within the $O(n)$ time bound. This gives an $O(n)$ weak shared coin protocol, and thus $O(n)$ consensus. Our protocol is given in Section 3; it is adapted to use a collection of approximate counters, and also contains some technical improvements on [Aspnes and Waarts 1996; Aspnes et al. 2008] that simplify the analysis.

We conclude, in Section 4, with a discussion of our results and the problems that remain open, the main ones being reducing the complexity of the counter and applying the counter to more tasks.

2. A SUBLINEAR APPROXIMATE COUNTER

In this section we describe the full approximate counter and prove its complexity and accuracy. The sampling component appears in Section 2.1 and the expander component in Section 2.2. In Section 2.3 we show how to combine the two counters in order to obtain the approximate counter with sublinear work and a high probability for δ-accurate read operations.

2.1 The sampling component

The sampling component of the counter works by having each increment operation set a bit at a random location in an array a of size N, and having each approximate read operation sample s locations with replacement, computing an estimate of the counter value that is $(N/s)S$ where S is the number of one bits seen among these samples. The accuracy of this estimate will depend on the choice of the parameters N and s; specific values useful for our application will be given later.

When there are few increments, this estimate will have high relative error; however, we will show that after enough increments, the resulting value is neither too high nor too low, despite any strategy the adversary might attempt to bias it. What makes this difficult is that an adaptive adversary can see the locations of the writes and reads done by each operation before the actual read or write is performed, and selectively delay processes in response.

```
  shared data: array $a\,[1..N]$ of multi-writer bits, initially all 0
1 Let $r$ be a uniform random index in the range $1..N$;
2 $a[r] \leftarrow 1$;
```

Procedure `SamplingIncrement`

```
1 $S \leftarrow 0$;
2 for $i \leftarrow 1$ to $s$ do
3     Let $r$ be a uniform random index in the range $1..N$;
4     $S \leftarrow S + a[r]$;
5 end
6 return $(N/s) \cdot S$
```

Procedure `SamplingRead`

There are two main issues we have to consider in showing that `SamplingRead` works:

(1) The sampling procedure may under- or over-represent the number of bits set in a. Here we need concentration bounds on the sampled value. For the lower bound, we consider only the contribution to the sample obtained by reading bits that are set before the start of the call to `SamplingRead`. This undercounts the actual number of bits seen, but allows the use of standard Chernoff bounds since the probability of observing one of these pre-written bits at each sample is fixed and independent.. For the upper bound, the situation is more complicated,

because the adversary may choose to allow fewer or more increments depending on how the sampling so far has gone. Here we apply a more sophisticated analysis, showing that the observed sample gives an estimate not much higher than the number of increments that start before the call to `SamplingRead` finishes using the method of bounded differences.

(2) The bits set in a may under-represent the actual number of increments, for example if many calls to `SamplingIncrement` write to the same location. This problem is exacerbated by the adversary's ability to delay processes between choosing their location to write (in Line 1) and actually writing the bit (in Line 2), because the adversary can selectively hold back writes to new locations while allowing through writes to locations already written. However, it is not hard to show that the total number of *lost* writes during an execution with n increments is $\binom{n}{2}/N$ on average, and a further application of Chernoff bounds puts the number of lost writes close to this value with high probability. For a suitable choice of N and sufficiently many completed increments, the relative error contributed by these lost writes is negligible.

We begin by showing the bound on the number of lost writes. Note that this is a global bound that applies to an entire execution, so unlike the later bounds on sampling error, it does not depend on the number of calls to `SamplingRead`.

LEMMA 2. *Fix an adversary strategy, and let each of n processes execute* `SamplingIncrement` *at most once. For each t, let a_t be the state of array a after t steps and let k_t be the number of processes that have executed the write operation in Line 2 of* `SamplingIncrement` *after t steps. Then*

$$\Pr\left[\exists t : \sum_{r=1}^{N} a_t[r] \le k_t - n^2/N\right] < \exp\left(-\binom{n}{2}/3N\right).$$

PROOF. Order the n increment operations by the step at which each chooses its location to write in Line 1 of `SamplingIncrement`. For each i, let X_i be the indicator variable for the event that the i-th increment chooses a location previously chosen by some other increment. It is easy to see that $\Pr[X_i = 1 | X_1 \dots X_{i-1}] = \frac{(i-1)-\sum_{j=1}^{i-1} X_j}{N} \le \frac{i-1}{N}$, as the numerator in the middle expression simply counts the number of previously-chosen locations.

Because the conditional probability of each X_i is bounded, we can construct a second series of random variables Y_i where for each Y_i, $X_i \le Y_i$, and $\Pr[Y_i = 1 | Y_1 \dots Y_{i-1}]$ equals $\frac{i-1}{N}$ exactly. We then have $\mathrm{E}\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} \frac{i-1}{N} = \binom{n}{2}/N$.

Since the probability of each Y_i doesn't depend on the outcome of its predecessors, we also have that the Y_i are independent. So standard Chernoff bounds apply (see, e.g., [Mitzenmacher and Upfal 2005, Theorem 4.4, case 2]), and for any $0 \le \delta \le 1$,

$$\Pr\left[\sum_{i=1}^{n} Y_i \ge (1+\delta)\binom{n}{2}/N\right] \le \exp\left(-\delta^2\binom{n}{2}/3N\right).$$

For convenience, we set $\delta = 1$ and use $2\binom{n}{2} < n^2$ to simplify this to

$$\Pr\left[\sum_{i=1}^{n} Y_i \geq n^2/N\right] \leq \exp\left(-\binom{n}{2}/3N\right),$$

and the same bound immediately applies to $\sum_{i=1}^{n} X_i \leq \sum_{i=1}^{n} Y_i$.

To complete the proof, we must show that the bound also applies to the difference between k_t and $\sum a_t[r]$ for each step t. Here we observe that no two increments i and j with $i < j$ and $X_i = X_j = 0$ both write to the same location (otherwise X_j would be 1). After k_t completed increments, at least $k_t - \sum_{i=1}^{n} X_i$ of these increments thus write to different locations, giving $\sum_{r=1}^{N} a_t[r] \geq k_t - \sum_{i=1}^{n} X_i \geq k_t - \sum_{i=1}^{n} Y_i$. □

The next lemma bounds the probability that a value R returned from `SamplingRead` is too small compared to the number of bits that are set in the array. Combining it with the previous lemma will later allow us to bound the probability that a value R returned from `SamplingRead` is too small compared to the number of `SamplingIncrement` operations.

LEMMA 3. *Fix an adversary strategy, and consider an execution of* `SamplingRead`*. Let A be the set of indices r in a such that $a[r] = 1$ at the start of this execution. Let R be the random variable equal to the value returned by* `SamplingRead`*. Then for any $0 \leq \delta \leq 1$,*

$$\Pr\left[R \leq (1-\delta)\cdot|A|\,\right] \leq \exp\left(-\frac{\delta^2(s/N)\cdot|A|}{2}\right).$$

PROOF. Observe that the process increments its count S each time it reads a location i in A, which occurs with independent probability $(1/N)\cdot|A|$ per sample. Thus the final value of S is bounded below by a sum of independent random variables with total expectation $(s/N)\cdot|A|$; the probability that this sum is less than or equal to $(1-\delta)(s/N)\cdot|A|$ is less than $\exp\left(-\delta^2(s/N)|A|/2\right)$ from Chernoff bounds. But then the same bound holds for the probability that $R = (N/s)S$ is less than or equal to $(N/s)(1-\delta)(s/N)|A| = (1-\delta)|A|$. □

We now bound the probability that a value R returned by a call to `SamplingRead` is too high.

LEMMA 4. *Fix an adversary strategy, and consider an execution of* `SamplingRead` *as part of a global execution that includes at most $n \leq N$* `SamplingIncrement` *operations. Let K be the number of* `SamplingIncrement` *operations that start before this execution of* `SamplingRead` *finishes. Let R be the random variable equal to the value returned by* `SamplingRead`*. Then for every δ,*

$$\Pr\left[R \geq (1+\delta)K\right] \leq \exp\left(-\frac{(\delta K)^2 s}{2N^2}\right).$$

PROOF. Let S_i be the value of S after i iterations of the loop of the `SamplingRead` operation, and let K_i be the number of increment operations that start before the read operation in Line 4 in the i-th iteration of the loop. Let $X_i = S_i - \sum_{j=1}^{i} K_j/N$.

We will apply the method of bounded increments to show that X_s is small with high probability.

Observe that $X_{i+1} - X_i = (S_{i+1} - S_i) - K_{i+1}/N$. From the fact that $S_i \leq S_{i+1} \leq S_i + 1$ and $0 \leq K_{i+1} \leq n \leq N$, it follows that $|X_{i+1} - X_i| \leq 1$. We show in addition the supermartingale property $\mathrm{E}[X_{i+1}|X_1 \dots X_i] \leq X_i$, which allows us to apply the supermartingale version of Azuma's inequality [Wormald 1999, Lemma 4.2] to get the desired bound.

Condition on all events prior to the i-th read, and consider what happens with the $(i+1)$-th read. The S_{i+1} component of X_{i+1} rises if and only if the reader observes a 1 in its chosen location $a[r]$. Assume that r is chosen immediately after the previous read (there is no payoff to the adversary for waiting); then the probability that $a[r]$ either already contains a 1 or is covered by a pending write is at most K_i/N. The adversary can schedule additional increments after r is chosen but before $a[r]$ is read; each of these adds at most a $1/N$ probability of placing a 1 in $a[r]$, but also increases the number of started increments by 1. Summing all probabilities thus gives a bound of $K_i/N + (K_{i+1} - K_i)/N = K_{i+1}/N$ on the probability that $S_{i+1} - S_i = 1$. This is precisely the net change in $\sum_{j=1}^{i+1} K_j/N$. It follows that $\mathrm{E}[X_{i+1} - X_i|X_1 \dots X_i] \leq 0$ and thus $\mathrm{E}[X_{i+1}|X_1 \dots X_i] \leq X_i$.

We now apply Azuma's inequality. For any $\alpha > 0$, we have

$$\Pr[X_s \geq \alpha] \leq \exp\left(-\alpha^2/2s\right).$$

The definition of X_s implies $X_s = S_s - \sum_{j=1}^{s} K_j/N$, and since K_j is an increasing function of j we have $S_s \leq X_s + (s/N)K_s$. The value returned by `SamplingIncrement` is $R = (N/s)S_s$ and therefore $\Pr[R \geq \alpha(N/s) + K_s] \leq \Pr[S_s \geq \alpha + (s/N)K_s] \leq \Pr[X_s \geq \alpha] \leq \exp\left(-\alpha^2/2s\right)$.

Finally, choose $\alpha = (s/N)\delta K$ and recall that $K = K_s$; then $\alpha(N/s) = \delta K$ and hence

$$\Pr[R \geq (1+\delta)K] \leq \exp\left(-\frac{(\delta K)^2 s}{2N^2}\right).$$

□

We are now ready to prove that there is high probability for a call to `SamplingIncrement` to be δ-accurate. The choice of the parameters N and s combines two conflicting considerations. On one hand, we would like s to be as small as possible, as it determines the cost of `SamplingRead`. At the same time we would like the probability of error to be exponentially small, which requires s to be large (otherwise we get a small error probability only for large numbers of increment operations, which will require the expander component to work for larger numbers of increment operations).

THEOREM 5. *Let $0 < \epsilon < 2/5$ and $2n^{-\epsilon/4} \leq \delta \leq 1/2$. Fix an adversary strategy, and consider an execution of a single sampling counter with $s = n^{4/5+\epsilon}$ and $N = n^{6/5+\epsilon/4}$ that includes at most n calls to* `SamplingIncrement`. *Consider a call to* `SamplingRead` *that starts after at least $n^{4/5}/\delta$ calls to* `SamplingIncrement` *finish. Then the probability that this call is δ-inaccurate is at most* $\exp\left(-\delta^2 n^{\epsilon/2}/2\right)(1 + o(1))$.

PROOF. We will consider three sources of error, then show that the last one dominates the total error probability.

From Lemma 2, the probability that more than $n^2/N = n^{4/5-\epsilon/4}$ writes are lost is at most $\exp\left(-\binom{n}{2}/3N\right) \leq \exp\left(-\binom{n}{2}/3n^{6/5+\epsilon/4}\right) \leq \exp\left(-n^{4/5-\epsilon}\right)$, for sufficiently large n.

If not more than $n^2/N = n^{4/5-\epsilon/4}$ writes are lost, then for any call to `SamplingRead` starting after $k \geq n^{4/5}/\delta > n^{4/5}$ completed increments, we have $|A| \geq k - n^{4/5-\epsilon/4} \geq k(1-n^{-\epsilon/4}) \geq k(1-\delta/2)$, where A is the set of indices r in a such that $a[r] = 1$. In this case we have $(1-\delta)k \leq (1-\delta/2)^2 k \leq (1-\delta/2)|A|$, hence the probability that $R \leq (1-\delta)k$ is at most the probability that $R \leq (1-\delta/2)|A|$, which from Lemma 3 is at most

$$\begin{aligned}
&\exp\left(-\frac{(\delta/2)^2(s/N)\cdot|A|}{2}\right) \leq \exp\left(-\frac{1}{2}(\delta/2)^2 n^{-2/5+(3/4)\epsilon}(1-\delta/2)n^{4/5}\right) \\
&\leq \exp\left(-\frac{1}{2}(2n^{-\epsilon/4})^2(1-\frac{1}{4})n^{2/5+(3/4)\epsilon}\right) \leq \exp\left(-\frac{3}{2}n^{2/5+(1/4)\epsilon}\right) \\
&\leq \exp(-n^{2/5}),
\end{aligned}$$

again for sufficiently large n.

Finally, we consider the possibility that some call to `SamplingRead` returns a value that is too high. The number K of calls to `SamplingIncrement` that start before the call to `SamplingRead` finishes is at least k, which is at least $n^{4/5}$. Apply Lemma 4 to get, for a single call to `SamplingRead`,

$$\Pr[R \geq (1+\delta)K] \leq \exp\left(-\frac{(\delta K)^2 s}{2N^2}\right) \leq \exp\left(-\frac{(\delta n^{4/5})^2 n^{4/5+\epsilon}}{2(n^{6/5+\epsilon/4})^2}\right) = \exp\left(-\delta^2 n^{\epsilon/2}/2\right).$$

Summing the three probabilities gives a total error probability of

$$\exp\left(-n^{4/5-\epsilon}\right) + \exp(-n^{2/5}) + \exp\left(-\delta^2 n^{\epsilon/2}/2\right).$$

For $\epsilon < 2/5$ and $\delta \leq 1/2$ the last term is at least $\exp\left(-n^{1/5}/8\right)$, which easily dominates the others. □

Note that while the theorem bounds the probability of failure of each call to `SamplingRead`, these probabilities are not independent: there is a small but nonzero chance that the bound in Lemma 2 will fail, causing all reads to return values that are too low.

2.2 The expander component

We augment the sampling counter with a second data structure that returns accurate values for small values of k; in particular, for $k \leq K_{\max} = n^{4/5+\epsilon/4}$. Like the sampling counter, this data structure also uses an array of multi-writer bits. Each increment operation sets a subset of D bits in the array. These subsets will be chosen so that for any $k \leq K_{\max}$ increments, the number of bits set in the array will be at least $(1-\delta)Dk$.

The value of the this component is computed by collecting every bit in the second array and returning the number of ones observed divided by D. For a small number of increments this will give a value between $(1-\delta)k$ and K.

The property that each set of $k \leq K_{\max}$ increments sets of $(1-\delta)Dk$ bits is precisely the defining property of an **expander**. We use a recent explicit expander construction of [Guruswami et al. 2007] to get good performance out of the expander component of the counter. Concurrency between increment and read operations raises some additional complications; these are handled by providing separate lower and upper bounds on the return value of the counter that may diverge in the presence of concurrency.

Expanders have a long history in combinatorics, with many variants and applications; see [Hoory et al. 2006] for a recent survey. We use an explicit construction of an **unbalanced bipartite expander** of Guruswami *et al.* [Guruswami et al. 2007]. In their notation, a bipartite multigraph has a set of left-vertices $[N] = \{1 \ldots N\}$, a set of right-vertices $[M] = \{1 \ldots M\}$, and a function $\Gamma : [N] \times [D] \rightarrow [M]$ that specifies for each left-vertex and each index i in $[D] = \{1 \ldots D\}$ a corresponding right-vertex. Such a multigraph is a $(\leq K_{\max}, A)$-expander if every set S of $K \leq K_{\max}$ left-vertices has $|\Gamma(S)| \geq A \cdot |S|$. Intuitively, the right-hand neighbors of any small set of left-hand nodes don't overlap much. The main result of [Guruswami et al. 2007] is the following theorem:

Theorem 6 [Guruswami et al. 2007]. *For every constant $\alpha > 0$, every $N \in \mathbb{N}$, $K_{\max} \leq N$, and $\delta > 0$, there is an explicit $(\leq K_{\max}, (1-\delta)D)$-expander $\Gamma : [N] \times [D] \rightarrow [M]$ with degree $D = O((\log N)(\log K_{\max})/\delta)^{1+1/\alpha}$ and $M \leq D^2 \cdot K_{\max}^{1+\alpha}$. Moreover, D is a power of 2.*

Given an expander of this form, we represent the counter as an array corresponding to the right-hand side. An increment for process pid consists of setting all bits in $\Gamma(\mathsf{pid})$, and requires D register write operations. A read operation collects the entire array of M bits, taking M register read operations, and returns the number of one bits seen divided by D. The expansion property guarantees that for small enough numbers of increments, this quantity will not be too much less than the correct value.

```
  shared data: array $b[1..M]$ of multi-writer bits
1 for $i \leftarrow 1$ to $D$ do
2     $b[\Gamma(\mathsf{pid}, i)] \leftarrow 1$
3 end
```

Procedure `ExpanderIncrement`.

```
1 return $\frac{1}{D}\sum_{i=1}^{M} b[i]$
```

Procedure `ExpanderRead`

It remains to choose the parameters of the expander. The value δ we will use in the expander is the same δ used for the sampling counter. We assume as in Theorem 5 that $\delta \leq 1/2$, while allowing for the possibility that δ depends on n. We also use the parameter ϵ from the exponent of the run-time of the sampling counter, with the assumption that ϵ does not exceed $2/5$. Our goal is to arrange for the cost of reading the entire right-hand side of the expander to be asymptotically no higher than the $n^{4/5+\epsilon}$ cost of `SamplingRead`, ignoring log terms and terms depending on δ. At the same time we want to set $K_{\max}$ higher than the minimum accurate count $n^{4/5+\epsilon/4}$ for the sampling counter. We accomplish these goals by choosing α so that $(4/5+\epsilon/4)(1+\alpha) = 4/5+\epsilon$.

$$\begin{aligned}
N &= n \\
K_{\max} &= n^{4/5+\epsilon/4} \\
\alpha &= \frac{15\epsilon}{16+5\epsilon} \text{ which implies that } 1+\alpha = \frac{16+20\epsilon}{16+5\epsilon} \text{ and } 1+1/\alpha = \frac{16+20\epsilon}{15\epsilon} \\
D &= O\left((\log N)\left(\log K_{\max}\right)/\delta\right)^{1+1/\alpha} = O\left(\left(\log^2 n\right)/\delta\right)^{1+1/\alpha} \\
&= O\left(\left(\log^2 n\right)/\delta\right)^{\frac{16+20\epsilon}{15\epsilon}} \\
&\le O\left(\left(\log^2 n\right)/\delta\right)^{\frac{16+20\cdot(2/5)}{15\epsilon}} \\
&= O\left((\log n)^{\frac{16}{5\epsilon}}/\delta^{\frac{8}{5\epsilon}}\right) \\
M &\le D^2 \cdot K_{\max}^{1+\alpha} = O\left(n^{(4/5+\epsilon/4)((16+20\epsilon)/(16+5\epsilon))}\left(\log n\right)^{32/(5\epsilon)}\left(1/\delta\right)^{16/(5\epsilon)}\right) \\
&= O\left(n^{4/5+\epsilon}\left(\log n\right)^{32/(5\epsilon)}\left(1/\delta\right)^{16/(5\epsilon)}\right)
\end{aligned}$$

The cost of an increment operation is $D = O\left(((1/\delta)\log n)^{O(1/\epsilon)}\right)$. The cost of a read is $M = O\left(n^{4/5+\epsilon}\left((1/\delta)\log n\right)^{O(1/\epsilon)}\right)$. We now show that the expander-based counter works as advertised:

LEMMA 7. *Let R be the value returned by an execution of* `ExpanderRead` *that starts after k calls to* `ExpanderIncrement` *have finished and ends after K calls to* `ExpanderIncrement` *have started. Then R satisfies*

$$(1-\delta)\min(k, n^{4/5+\epsilon/4}) \le R \le K.$$

PROOF. Recall that $K_{\max} = n^{4/5+\epsilon/4}$. From the expansion property, we have that a set S_0 of at least $(1-\delta)\min(k, K_{\max})D$ bits in b are set at the start of the `ExpanderRead` operation. It is also the case that a set S_1 of at most KD bits are set when it completes, since each `ExpanderIncrement` operation sets at most D bits. Because bits are never unset once their value is 1, the set S of bits i for which $b[i] = 1$ which is included in the sum in `ExpanderRead` satisfies $S_0 \subseteq S \subseteq S_1$ and thus $(1-\delta)\min(k, K_{\max})D \le |S_0| \le |S| \le |S_1| \le KD$. But then $(1-\delta)\min(k, K_{\max}) \le \frac{1}{D}|S| \le K$ □

2.3 The complete counter

The complete counter is obtained by combining the expander and sampling components into one approximate counter as described in the introduction, in a technique that ensures high probability for having a δ-accurate read.

The following theorem combines the bounds of Theorem 5 and Lemma 7.

```
SamplingIncrement();
ExpanderIncrement();
```

Procedure `ApproxIncrement`.

```
$S \leftarrow$ ExpanderRead();
if $S < (1-\delta)n^{4/5+\epsilon/4}$ then
    return $S$;
else
    return SamplingRead();
end
```

Procedure `ApproxRead`

Theorem 8. *Let n be the maximum number of calls to* `ApproxIncrement`. *Let $0 < \epsilon < 2/5$ be a constant and $2n^{-\epsilon/4} \leq \delta \leq 1/2$, as in Theorem 5. Then* `ApproxIncrement` *and* `ApproxRead` *together implement an approximate counter where the cost of* `ApproxIncrement` *is $O\left(((1/\delta)\log n)^{O(1/\epsilon)}\right)$, the cost of* `ApproxRead` *is $O\left(n^{4/5+\epsilon}((1/\delta)\log n)^{O(1/\epsilon)}\right)$, and for each call to to* `ApproxRead`*, the probability that it is δ-inaccurate is at most* $\exp\left(-\delta^2 n^{\epsilon/2}/2\right)(1+o(1))$.

Proof. The time bounds follow from summing the time bounds of the appropriate operations of the sampling and expander counters.

For the error bound, for each call to `ApproxRead`, there are two cases, depending on whether its return value is supplied by `ExpanderRead` or `SamplingRead`.

The first case occurs if `ExpanderRead` returns a value $S < (1-\delta)n^{4/5+\epsilon/4}$. Then from Lemma 7 we have that $(1-\delta)\min(k, n^{4/5+\epsilon/4}) \leq S \leq K$ (since k acts as a lower bound on the number of calls to `ExpanderIncrement` that finish before the call to `ExpanderRead` starts and K similarly acts as an upper bound in the other direction). We immediately get $R = S \leq K \leq (1+\delta)K$. On the other side, expanding the min gives that either $(1-\delta)k \leq S$ or $(1-\delta)n^{4/5+\epsilon/4} \leq S$; but the latter case contradicts the assumption on S, so the former case holds. It follows that this call to `ApproxRead` is δ-accurate with probability 1.

The second case occurs if `ExpanderRead` returns a value $S \geq (1-\delta)n^{4/5+\epsilon/4}$. Now `ApproxRead` returns the value obtained by `SamplingRead`. Because `ApproxIncrement` calls `SamplingIncrement` before `ExpanderIncrement`, the number of started calls to `ExpanderIncrement` is a lower bound on the number of finished calls to `SamplingIncrement`. So from the fact that the number of started calls to `ExpanderIncrement` before the end of the call to `ExpanderRead` is at least $S \geq (1-\delta)n^{4/5+\epsilon/4}$, we have that the number of completed calls to `SamplingIncrement` before the start of the call to `SamplingRead` is at least $(1-\delta)n^{4/5+\epsilon/4}$. Under the assumption that $2n^{-\epsilon/4} \leq \delta \leq 1/2$, we have $(1-\delta) \geq 1/2$ and $n^{\epsilon/4} \geq 2/\delta$; it follows that $(1-\delta)n^{4/5+\epsilon/4} \geq n^{4/5}/\delta$, the condition under which Theorem 5 applies. This gives the probability bound claimed in the theorem.

□

3. APPLICATION: CONSENSUS WITH OPTIMAL INDIVIDUAL WORK

In this section we describe an application of our counting protocol: a protocol for solving randomized consensus with optimal $O(n)$ work per process. This improves the best previously known bound of $O(n \log n)$ of Aspnes *et al.* [Aspnes et al. 2008] to match the $\Omega(n)$ lower bound of Attiya and Censor [Attiya and Censor 2007]. While the Attiya-Censor results showed a tight bound of $\Theta(n^2)$ on the *total* number of operations carried out by all processes, our is the first protocol that guarantees that this work is in fact evenly distributed among all the processes.

As described previously, we use a standard reduction [Aspnes and Herlihy 1990] of randomized consensus to the problem of implementing a **weak shared coin**. The code for each process's actions in the shared coin implementation is given as Procedure `SharedCoin`, in which each process outputs either +1 or -1.

```
shared data: array c[0..(2 log n)] of approximate counters with parameters
             $\delta = 1/2$ and $\epsilon = 1/10$, array votes[1..n] of single-writer
             registers, multi-writer bit done
1  $i \leftarrow 0$;
2  $v_0 \leftarrow 0$;
3  varianceWritten $\leftarrow 0$;
4  while $v_i < 1$ and not done do
5      $i \leftarrow i + 1$;
6      $w_i = \min(\max(v_{i-1}, 1/n), 1/\sqrt{n})$;
7      $v_i = v_{i-1} + w_i^2$;
8      vote = LocalCoin() $\cdot w_i$;
9      votes[pid] $\leftarrow$ votes[pid] + vote;
10     if $v_i \geq 2^{\text{varianceWritten}}/n^2$ then
11         ApproxIncrement(c[varianceWritten]);
12         varianceWritten $\leftarrow$ varianceWritten + 1;
13         if $\sum_{k=0}^{2\log n} \left(2^k \cdot \text{ApproxRead}(c[k])\right) \geq 3n^2$ then
14             break;
15         end
16     end
17 end
18 done $\leftarrow$ true;
19 return $\text{sgn}(\sum_p \text{votes}[p])$;
```

Procedure `SharedCoin`

We now give a high-level description of the shared coin algorithm, which will be followed by a formal proof. Each process generates votes whose sum is recorded in an array of n single-writer registers, and whose variance is recorded in $2 \log n$ approximate counters. A process terminates and outputs the majority of votes when the total variance of the votes reaches a certain threshold, which is small enough to guarantee the claimed step complexity, and at the same time large enough to have a good probability for the votes to have a distinct majority.

In order to reduce the individual step complexity, the votes generated by a process have increasing weights. This allows fast processes running alone to cast heavier votes and reach the variance threshold after generating fewer votes.

The weight w_i of the i-th vote is a function of the total variance v_{i-1} of all previous votes, as computed in Line 6; we discuss the choice of this formula in more detail in Section 3.1.1. The voting operation consists of lines 6 through 9; each time the process votes, it computes the weight w_i of the next vote, updates the total variance v_i, generates a random vote with value $\pm w_i$ with equal probability, and adds this vote to the pool votes[pid], where pid is the current process id.

Termination can occur in one of three ways:

(1) The process by itself produces enough variance to cross the threshold (first clause of while loop test in Line 4).

(2) All processes collectively produce enough variance for the threshold test to succeed in (Line 13).

(3) The process observes that some other process has written done (second clause of while loop test in Line 4). This last case can only occur if some other process previously observed sufficient total variance to finish.

We use $2 \log n$ counters, since our counters can be incremented at most once by each process. Having approximate sublinear counters allows incrementing and reading them not very frequently, namely, only when increasing amounts of variance are generated by the process, which gives the linear complexity.

We show that the counters give a good approximation of the total variance, which when large enough has constant probability for the votes having a distinct majority, even in spite of small differences between the votes that different processes read, which may be caused by the asynchrony of the system.

3.1 Analysis

The proof of correctness for the shared coin protocol proceeds in several steps. In Section 3.1.1 we prove some properties of the weight function. These will allow us to bound the expected individual work of each process, and later will also be used to analyze the agreement parameter. In Section 3.1.2 we bound the individual work (Lemma 11), and prove bounds on the probabilities of terminating with a total variance of votes which is too low or too high. Finally, in Section 3.1.3 we analyze the sums of votes in different phases of Procedure 7, which allows us to prove in Theorem 18 that it implements a weak shared coin with a constant agreement parameter.

3.1.1 *Properties of the weight function.* The weight of the i-th vote is given by the formula $w_i = \min(\max(v_{i-1}, 1/n), 1/\sqrt{n})$, where $v_{i-1} = \sum_{j=1}^{i-1} w_j^2$ is the total variance contributed by all previous votes.

The cap of $1/\sqrt{n}$ keeps any single vote from being too large, which will help us show in Section 3.1.3 that the core votes are normally distributed in the limit. The use of $\max(v_{i-1}, 1/n)$ bounds the weight of all unwritten votes in any state by the total variance of all written votes, plus a small constant corresponding to those processes that are still casting the minimum votes of weight $1/n$. This gives a bound on the **bias** that the adversary can create by selectively stopping a process after it generates its i-th vote in Line 8 but before it writes it in Line 9.

LEMMA 9. *For any values $i_j \geq 0$, we have $\sum_{j=1}^{n} w_{i_j} \leq 1 + \sum_{j=1}^{n} v_{i_j-1}$.*

PROOF. Sum $w_{i_j} \leq \max(v_{i_j-1}, 1/n) \leq \left(v_{i_j-1} + 1/n\right)$ over all j. □

Despite wanting to keep w_i small relative to v_i, we still want to generate variance quickly. The following lemma states that any single process can generate a total variance $v_i \geq 1$ after only $i = 4n$ votes. It follows immediately that the loop in Procedure 7 is executed at most $4n$ times.

LEMMA 10. *All of the following conditions hold:*

(*1*) $v_1 = 1/n^2$

(*2*) $v_{i+1} \leq 2v_i \qquad [i \geq 1]$

(*3*) $v_{4n} \geq 1$.

PROOF. We observe that the following recurrence holds for v_i:

$$v_i = v_{i-1} + w_i^2 = v_{i-1} + \left(\min(\max(v_{i-1}, 1/n), 1/\sqrt{n})\right)^2,$$

with a base case of $v_0 = 0$. We can immediately compute $v_1 = 1/n^2$, giving (1).

It also follows that $v_i \geq v_1 = 1/n^2$ for all $i \geq 1$. Let $i \geq 1$ and consider the possible values of v_{i-1}. If $v_{i-1} \leq 1/n$ then $w_i^2 = 1/n^2$, therefore $v_i = v_{i-1} + 1/n^2 \leq 2v_{i-1}$. Otherwise, if $1/n \leq v_{i-1} < 1/\sqrt{n}$ then $w_i^2 = v_{i-1}^2 < 1/n$, therefore $v_i \leq v_{i-1} + 1/n \leq 2v_{i-1}$. Finally, if $v_{i-1} \geq 1/\sqrt{n}$ then $w_i^2 = 1/n$ and we have $v_i = v_{i-1} + 1/n \leq 2v_{i-1}$. So (2) holds for all $i \geq 1$.

To prove (3), we consider three phases of the increase in v_i, depending on whether $w_i = 1/n$, $w_i = v_{i-1} \geq 1/n$, or $w_i = 1/\sqrt{n}$.

In the first phase, we have that for any $i > 0$, $v_i \geq v_{i-1} + 1/n^2$, and thus $v_i \geq i/n^2$. In particular, for $i = n$ we have $v_i \geq 1/n$.

For the second phase, suppose that $v_{i-1} \leq 1/\sqrt{n}$. We then have $v_i \geq v_{i-1} + v_{i-1}^2$. If this holds, and there is some $x \geq 1$ such that $v_i \geq 1/x$, then

$$\begin{aligned} v_{i+1} \geq v_i + v_i^2 \geq \frac{1}{x} + 1/x^2 &= \frac{x+1}{x^2} = \frac{(x+1)(x-1/2)}{x^2(x-1/2)} \\ &= \frac{x^2 + x/2 - 1/2}{x^2(x-1/2)} = \frac{1 + 1/(2x) - 1/(2x^2)}{x - 1/2} \geq \frac{1}{x-1/2}. \end{aligned}$$

By iterating this calculation, we obtain that $v_{i+t} \geq \frac{1}{x-t/2}$, so long as $v_{i+t-1} \leq 1/\sqrt{n}$. Starting with $v_n \geq 1/n$, we thus get $v_{n+t} \geq 1/(n - t/2)$, which gives $v_i \geq 1/\sqrt{n}$ for some $i \leq (n + (2n - \sqrt{n})) \leq 3n$.

At this point, w_i is capped by $1/\sqrt{n}$; the increment to v_i is thus $w_i^2 = 1/n$, so after a further $(n - \sqrt{n}) \leq n$ votes, we have $v_i \geq 1$. The total number of votes is bounded by $4n$, as claimed. □

3.1.2 *Termination.* We begin analyzing the situation of termination, i.e., when no more votes are generated, by bounding the running time of the protocol.

LEMMA 11. *Procedure* `SharedCoin` *executes $O(n)$ local coin-flips and $O(n)$ register operations, including those incurred by $O(\log^2 n)$* `ApproxRead` *operations on approximate counters.*

Proof. Lemma 10 implies that each process terminates after casting at most $4n$ votes. This gives an $O(n)$ bound on the number of iterations of the main loop. Each iteration requires one call to `LocalCoin` and two register operations (the read of done in Line 4 and the write to votes[pid] in Line 9, assuming the previous value of votes[pid] is cached in an internal variable), plus whatever operations are needed to execute the threshold test in Lines 11 through 13. These lines are executed at most $1 + 2\log n$ times (since varianceWritten rises by 1 for each execution), and their cost is dominated by the $1 + 2\log n$ calls to `ApproxRead` at a cost of $O\left(n^{4/5+\epsilon}\left((1/\delta)\log n\right)^{O(1/\epsilon)}\right)$ each, where $\epsilon = 1/10$ and $\delta = 1/2$ are both constants. The cost of the at most $(1 + 2\log n)^2$ total calls to `ApproxRead` is thus $O\left(n^{9/10}\log^{O(1)} n\right) = O(n)$. $\square$

Consider the sequence of votes generated by all processes, ordered by the interleaving of execution of the `LocalCoin` procedure. Write X_t for the random variable representing the value of the t-th such vote (or 0 if there are fewer than t total votes); we thus have a sequence of votes $X_1, X_2, \ldots$.

We wish to bound any sum computed in Line 13 according to the total variance of the votes that have been generated, where for a given number of votes t their total variance is $\sum_{i=1}^{t} X_i^2$.

For a given t, consider the state of the counters when the t-th vote is generated. For each process j, let k_j^t be the maximum index of any counter in c for which j has completed an `ApproxIncrement` operation, and let ℓ_j^t be the maximum index of any counter for which j has started an `ApproxIncrement` operation. If there is no such index, set k_j^t or ℓ_j^t to -1. Let i_j^t be the total number of votes generated by process j among the first t votes, i.e., $\sum_{i=1}^{t} X_i^2 = \sum_{j=1}^{n} v_{i_j^t}$. We first bound k_j^t and ℓ_j^t in terms of $v_{i_j^t}$.

Lemma 12. *For every t, we have $2^{\ell_j^t} \leq v_{i_j^t} n^2 + 1/2$ and $2^{k_j^t+1} \geq v_{i_j^t} n^2$.*

Proof. We begin with an upper bound on $2^{\ell_j^t}$. Observe that the test in Line 10 means that `ApproxIncrement(c[`k`])` can have started only if $v_{i_j^t} \geq 2^k/n^2$; it follows that either $\ell_j^t = -1$ or $2^{\ell_j^t} \leq v_{i_j^t} n^2$; in either case we have

$$2^{\ell_j^t} \leq v_{i_j^t} n^2 + 1/2.$$

Getting a lower bound on $2^{k_j^t}$ is slightly harder, since we can't rely solely on the test in Line 10 succeeding but must also show that varianceWritten is large enough that $2^{\text{varianceWritten}}$ is in fact close to v_i^2. We do so by proving, by induction on i, that at the end of each iteration of the main loop in Procedure `SharedCoin`, $v_i \leq 2^{\text{varianceWritten}}/n^2$. To avoid ambiguity (and excessive text), we will write W_i for the value of varianceWritten at the end of the i-th iteration.

The base case is $i = 1$, where inspection of the code reveals $v_1 = 1/n^2$ and $W_1 = 1$; in this case $v_1 \leq 2^{W_1}/n^2 = 2/n^2$. For larger i, suppose that it holds that $v_{i-1} \leq 2^{W_{i-1}}/n^2$. Then $v_i \leq 2v_{i-1} \leq 2^{W_{i-1}+1}/n^2$ (the first inequality follows from (2) of Lemma 10). It is possible that v_i is much smaller than this bound, indeed, small enough that $v_i < 2^{W_{i-1}}/n^2$; in this case $W_i = W_{i-1}$ and the invariant continues to hold. If not, Line 12 is executed, and so we have $W_i = W_{i-1} + 1$. But then $v_i \leq 2^{W_{i-1}+1}/n^2 = 2^{W_i}/n^2$, so the invariant holds here as well.

In bounding $2^{k_j^t}$, the worst case (for $k_j^t \geq 0$) is when $k_j^t = W_{i_j^t - 1}$, the value of varianceWritten at the end of the previous iteration of the loop. In this case we have $v_{i_j^t} \leq 2v_{i_j^t - 1} \leq 2 \cdot 2^{k_j^t}/n^2 = 2^{k_j^t+1}/n^2$. For $k_j^t = -1$, we have $i_j^t \leq 1$, so $v_{i_j^t} \leq 1/n^2 = 2^{-1+1}/n^2 = 2^{k_j^t+1}/n^2$. In either case we get

$$2^{k_j^t+1} \geq v_{i_j^t} n^2.$$

□

We now consider the interaction between `ApproxIncrement` and `ApproxRead` operations in order to bound any sum computed in Line 13. The next lemma shows a small upper bound on the probability that the sum is too large.

Lemma 13. *If S is a sum computed in Line 13, where the first* `ApproxRead` *operation is started after t total votes are generated, then*

$$\Pr\left[S \leq \frac{1}{2} n^2 \sum_{i=1}^{t} X_i^2 - n/2\right] \leq (2\log n + 1)\left(\exp\left(-n^{1/20}/8\right)(1 + o(1))\right).$$

Proof. For each k let $r[k]$ be the value returned by the `ApproxRead(c[`k`])` operation included in the sum, and let $c[k]$ be the number of calls to `ApproxIncrement(c[`k`])` that have finished before the summation starts. Applying Theorem 8 with $\epsilon = 1/10$ and $\delta = 1/2$ gives

$$\Pr\left[r[k] \leq \frac{1}{2} c[k]\right] \leq \exp\left(-n^{1/20}/8\right)(1 + o(1)),$$

This implies that with probability at least $1-(2\log n+1)\cdot\left(\exp\left(-n^{1/20}/8\right)(1 + o(1))\right)$ we have $r[k] > \frac{1}{2}c[k]$ for every k. In this case

$$\begin{aligned} S = \sum_{k=0}^{2\log n} 2^k r[k] &> \frac{1}{2}\sum_{k=0}^{2\log n} 2^k c[k] = \frac{1}{2}\sum_{j=1}^{n}\sum_{m=0}^{k_j^t} 2^m = \frac{1}{2}\sum_{j=1}^{n}\left(2^{k_j^t+1} - 1\right) \\ &\geq \frac{1}{2}\sum_{j=1}^{n} v_{i_j^t} n^2 - n/2 = \frac{1}{2} n^2 \sum_{i=1}^{t} X_i^2 - n/2, \end{aligned}$$

where the fourth inequality follows from Lemma 12. This completes the proof. □

Similarly, the next lemma shows a small upper bound on the probability that the sum is too small.

Lemma 14. *If S' is a sum computed in Line 13, where the last* `ApproxRead` *operation is completed before t' total votes are generated, then*

$$\Pr\left[S' \geq 3n^2 \sum_{i=1}^{t'} X_i^2\right] \leq (2\log n + 1)\exp\left(-n^{1/20}/8\right)(1 + o(1)).$$

Proof. For each k let $r'[k]$ be the value returned by the `ApproxRead(c[`k`])` operation included in the sum, and let $c'[k]$ be the number that start before the summation finishes. Applying Theorem 8 with $\epsilon = 1/10$ and $\delta = 1/2$ gives

$$\Pr\left[r'[k] \geq \frac{3}{2}c'[k]\right] \leq \exp\left(-n^{1/20}/8\right)(1+o(1)).$$

This implies that with probability at least $1-(2\log n+1)\cdot\left(\exp\left(-n^{1/20}/8\right)(1+o(1))\right)$ we have $r'[k] < \frac{3}{2}c'[k]$ for every k. In this case

$$S' = \sum_{k=0}^{2\log n} 2^k r'[k] < \frac{3}{2}\sum_{k=0}^{2\log n} 2^k c'[k] = \frac{3}{2}\sum_{j=1}^{n}\sum_{m=0}^{\ell_j^{t'}} 2^m = \frac{3}{2}\sum_{j=1}^{n}\left(2^{\ell_j^{t'}+1}-1\right)$$
$$\leq \frac{3}{2}\sum_{j=1}^{n}\left(2\left(v_{i_j^{t'}}n^2+1/2\right)-1\right) = 3\sum_{j=1}^{n} v_{i_j^{t'}}n^2 = 3n^2\sum_{i=1}^{t'} X_i^2.$$

where the fourth inequality follows from Lemma 12. This completes the proof. □

Using the two previous lemmas, we now prove upper and lower bounds on the total variance of all the generated votes.

Lemma 15. *Let T be the total number of votes generated by all processes during an execution of the shared coin protocol, and let $V = \sum_{i=1}^{T} X_i^2$ be the total variance of these votes. Then we have*

(*1*) $\Pr[V < 1] \leq n(2\log n+1)^2\exp\left(-n^{1/20}/8\right)(1+o(1))$,
(*2*) $\Pr\left[V > 13 + \frac{4}{n}\right] \leq n(2\log n+1)\exp\left(-n^{1/20}/8\right)(1+o(1))$.

Proof. Termination with $V < 1$ cannot occur as the result of some process failing the main loop test $v_i < 1$, as if this test fails, that process alone gives $V \geq 1$. So the only possibility is that the threshold test in Line 13 succeeds for some process despite the low total variance. But since the total variance of all votes is less than 1, for any particular sum of observed counter values S' we have from Lemma 14 that $\Pr[S' \geq 3n^2] \leq (2\log n+1)\exp\left(-n^{1/20}/8\right)(1+o(1))$, from which it follows, by summing over the at most $n(2\log n+1)$ executions of Line 13 by all processes, that the probability of premature termination is at most $n(2\log n+1)^2\exp\left(-n^{1/20}/8\right)(1+o(1))$.

For (2), suppose that after t_1 votes we have $\sum_{i=1}^{t_1} X_i^2 \geq 6 + 1/n$. If there is no such t_1, then $V < 13 + \frac{4}{n}$; otherwise, let t_1 be the smallest value with this property. Because t_1 is least, we have $\sum_{i=1}^{t_1} X_i^2 < 6 + 1/n + X_{t_1}^2 \leq 6 + 2/n$.

From Lemma 13 we have that, for any execution of Line 13 that starts after these t_1 votes, the return value S satisfies $\Pr\left[S \leq \frac{1}{2}n^2(6+1/n) - n/2\right] = \Pr[S \leq 3n^2] \leq (2\log n+1)\exp\left(-n^{1/20}/8\right)(1+o(1))$. The probability that there exists any process j for which the *first* execution of the threshold test (if any) starting after the t_1-th vote returns a value less than or equal to $3n^2$ is at most n times this bound, or $n(2\log n+1)\exp\left(-n^{1/20}/8\right)(1+o(1))$.

Assuming this improbably event does not hold, every process that executes the threshold test after t_1 total votes will succeed, and as a result will cast no more votes. So we must bound the amount of additional variance each process can add before it reaches this point. Recall that $i_j^{t_1}$ is the number of votes cast by process j among the first t_1 votes, and let i'_j be the total number of votes cast by process j before termination. Then under the assumption that j's next threshold test succeeds, we have $v_{i'_j} < 2v_{i_j^{t_1}} + 1/n$, as j can at most double its variance and cast one additional vote before seeing $v_i \geq 2^{\mathsf{varianceWritten}}$. So now we have

$$V = \sum_{i=1}^{T} X_i^2 = \sum_{j=1}^{n} v_{i'_j} < \sum_{j=1}^{n} (2v_{i_j^{t_1}} + 1/n) = 1 + 2\sum_{j=1}^{n} v_{i_j^{t_1}}$$

$$= 1 + 2\sum_{i=1}^{t_1} X_i^2 < 1 + 2(6 + 2/n) = 13 + \frac{4}{n}.$$

Thus (2) holds. □

3.1.3 *Core votes and extra votes.* We will assume for convenience that the adversary scheduler is deterministic, in particular that the choice of which process generates vote X_t is completely determined by the outcomes of votes X_1 through X_{t-1}; this assumption does not constrain the adversary's behavior, because any randomized adversary strategy can be expressed as a weighted average of deterministic strategies. Under this assumption, we have that the weight $|X_t|$ of X_t is a constant conditioned on $X_1 \dots X_{t-1}$, but because the adversary cannot predict the outcome of `LocalCoin`, the expectation of X_t is zero even conditioning on the previous votes. That $\mathrm{E}[X_t = 0 | X_1, \dots X_{t-1}]$ is the defining property of a class of stochastic processes known as **martingales** (see [Grimmet and Stirzaker 1992; Alon and Spencer 1992; Hall and Heyde 1980]); in particular the X_t variables form a **martingale difference sequence** while the variables $S_t = \sum_{i=1}^{t} X_t$ form a martingale proper.

Martingales are a useful class of processes because for many purposes they act like sums of independent random variables: there is an analog of the Central Limit Theorem that holds for martingales [Hall and Heyde 1980, Theorem 3.2], which we use in the proof of Lemma 16; and as with independent variables, the variance of S_t is equal to the sum of the variances of X_1 through X_t [Hall and Heyde 1980, p. 8], a fact we use in the proof of Lemma 17.

Martingales can also be neatly sliced by **stopping times**, where a stopping time is a random variable τ which is finite with probability 1 and for which the event $[\tau \leq t]$ can be determined by observing only the values of X_1 through X_t (see [Grimmet and Stirzaker 1992, Section 12.4]); the process $\{S'_t = \sum_{i=1}^{t} X'_i\}$ obtained by replacing X_t with $X'_t = X_t$ for $t \leq \tau$ and 0 otherwise, is also a martingale [Grimmet and Stirzaker 1992, Theorem 12.4.5], as is the sequence $S''_t = \sum_{i=1}^{t} X_{\tau+i}$ [Grimmet and Stirzaker 1992, Theorem 12.4.11]. We will use a stopping time to distinguish the core and extra votes.

Define τ as the least value such that either (a) $\sum_{t=1}^{\tau} X_t^2 \geq 1$ or (b) the protocol terminates after τ votes. Observe that τ is always finite, because if the protocol does not otherwise terminate, any process eventually generates 1 unit of variance on its own (as shown in Lemma 10). Because the weights of votes vary, τ is in general a random variable; but for a fixed adversary strategy, the condition $\tau = t$ can be detected by observing the values of $X_1 \dots X_t$. Thus τ is a stopping time relative to the X_t. The quantity S_τ will be called the **core vote** of the protocol. The remaining votes $X_{\tau+1}, X_{\tau+2}, \ldots$ form the **extra votes**.

First, we show a constant probability of the core vote being at least a constant. This will follow by an application of the martingale Central Limit Theorem, particularly in the form of Theorem 3.2 from [Hall and Heyde 1980]. This theorem considers a zero-mean **martingale array**, which is a sequence of tuples $\{S_{mt}, \mathcal{F}_{mt}, 1 \le t \le k_m, m \ge 1\}$ parameterized by m, where for each fixed m the sequence of random variables $\{S_{mt}\}$ is a zero-mean martingale with respect to the corresponding sequence of σ-algebras $\{\mathcal{F}_{mt}\}$, with difference sequence $X_{mt} = S_{mt} - S_{m,t-1}$. Specializing the theorem slightly, if it holds that:

(1) $\max_t |X_{mt}| \xrightarrow{p} 0$,
(2) $\sum_t X_{mt}^2 \xrightarrow{p} 1$,
(3) $\mathrm{E}\left[\max_t X_{mt}^2\right]$ is bounded in m, and
(4) $\mathcal{F}_{m,t} \subseteq \mathcal{F}_{m+1,t}$ for $1 \le t \le k_m, m \ge 1$,

then $S_{mt} \xrightarrow{d} N(0,1)$, where $N(0,1)$ has a normal distribution with zero mean and unit variance. Here $\xrightarrow{p}$ denotes convergence in probability and $\xrightarrow{d}$ denotes convergence in distribution.

Lemma 16. *For any fixed α and n sufficiently large, there is a constant probability p_α such that, for any adversary strategy,* $\Pr[S_\tau \ge \alpha] \ge p_\alpha$.

Proof. We construct our martingale array by considering, for each number of processes n, the set of all deterministic adversary strategies for scheduling Procedure `SharedCoin`. The first rows of the array correspond to all strategies for $n = 1$ (in any fixed order); subsequent rows hold all strategies for $n = 2$, $n = 3$, and so forth. Because each set of strategies is finite (for an execution with n process, each choice of the adversary chooses one of n processes to execute the next coin-flip in response to some particular pattern of $O(n^2)$ preceding coin-flips, giving at most $n^{O(n^2)}$ possible strategies), every adversary strategy eventually appears as some row m in the array. We will write n_m as the value of n corresponding to this row and observe that it grows without bound.

For each row in the array, we set k_m to include all possible votes, but truncate the actual set of coin-flips at time τ. Formally, we define $X_{mt} = X_t$ for $t \le \tau$, but set $X_{mt} = 0$ for larger t. This ensures that $S_{mk_m} = S_\tau$, the total core vote from each execution, while maintaining the martingale property and the fixed-length rows required by the theorem. We ensure the nesting condition (4) by using the same random variable to set the sign of each vote at time t in each row; in effect, we imagine that we are carrying out an infinite collection of simultaneous executions for different values of n and different adversary strategies using the same sequence of random local coin-flips.

Let's knock off the remaining requirements of the theorem in turn. For (1), we have that $\max_t |X_{mt}| \le 1/\sqrt{n_m}$, which converges to 0 absolutely (and thus in probability as well). For (2), by construction of τ and Lemma 15, we have that $1 \le \sum_t X_{mt}^2 \le 1 + X_{m\tau}^2 \le 1 + 1/n_m$, except for an event of probability at most $n_m(2\log n_m + 1)^2 \exp\left(-n_m^{\epsilon/2}/2\right)(1 + o(1))$, which goes to 0 in the limit.

Thus $\sum_t X_{mt}^2$ converges in probability to 1. For (3), we again use the fact that $X_{mt}^2 \leq 1/n_m$ for all t.

It follows that S_{mt} converges in distribution to $N(0,1)$. In particular, for any fixed α, we have that $\lim_{m\to\infty} \Pr[S_{mt} \geq \alpha] = \Pr[N(0,1) \geq \alpha]$, which is a constant. By choosing p_α strictly less than this constant, we have that for sufficiently large m (and thus for sufficiently large $n = n_m$), $\Pr[S_\tau \geq \alpha] = \Pr[S_{mt} \geq \alpha] \geq p_\alpha$. $\square$

By symmetry, we also have $\Pr[S_\tau \leq -\alpha] \geq p_\alpha$.

We now consider the effect of the extra votes. Our goal is to bound the probability that the total extra vote is too large using Chebyshev's inequality, obtaining a bound on the variance of the extra votes from a bound on the sum of the squares of the weights of all votes as shown in Lemma 15, (2). There is a complication in that (with very small probability), this latter quantity may be too big; we deal with this by truncating the process early (for now) and handling the rare case that the protocol runs too long later.

LEMMA 17. *Define τ' to be the maximum index such that (a) $X_{\tau'} \neq 0$ and (b) $\sum_{i=1}^{\tau'} X_i^2 \leq 13+4/n$. Let p_{19} be the probability from Lemma 16 that S_τ is at least 19. Then for sufficiently large n and any adversary strategy,* $\Pr[S_{\tau'} > 15] \geq (1/8)p_{19}$.

PROOF. From Lemma 16, the probability that the sum of the core votes S_τ is at least 19 is at least p_{19}. We wish to show that, conditioning on this event occurring, adding the extra votes up to τ' does not drive this total below 15.

Observe that τ' is a stopping time. For the rest of the proof, all probabilistic statements are conditioned on the values of $X_1 \dots X_\tau$.

Define $Y_i = X_{\tau+i}$ for $\tau + i \leq \tau'$ and 0 otherwise. Let $U_i = \sum_{j=1}^{i} Y_j$. Then $\{U_i\}$ is a martingale and $\mathrm{E}[U_i] = 0$ for all i. Let $i_{\max}$ be such that $Y_i = 0$ for $i > i_{\max}$ with probability 1 ($i_{\max}$ exists by Lemma 11). Then

$$\begin{aligned}\mathrm{Var}[U_{i_{\max}}] &= \mathrm{Var}\left[\sum_{i=1}^{i_{\max}} Y_i\right] = \sum_{i=1}^{i_{\max}} \mathrm{Var}\left[Y_i\right] = \sum_{i=1}^{i_{\max}} \mathrm{E}\left[Y_i^2\right] \\ &= \mathrm{E}\left[\sum_{i=1}^{i_{\max}} Y_i^2\right] \leq \mathrm{E}[13+4/n] = 13+4/n.\end{aligned}$$

So by Chebyshev's inequality,

$$\Pr\left[|U_{i_{\max}}| \geq 4\right] \leq \frac{13+4/n}{4^2} = 13/16 + 1/4n \leq 7/8,$$

when $n \geq 4$. But if $|U_{i_{\max}}| < 4$, we have $S_{\tau'} = S_\tau + U_{i_{\max}} \geq 19 - 4 = 15$. As the event $|U_{i_{\max}}| < 4$ occurs with conditional probability at least 1/8, the total probability that $S_{\tau'} \geq 15$ is at least $(1/8)p_{19}$. $\square$

3.1.4 *Full result.* We are now ready to prove the main theorem of having a constant agreement parameter.

THEOREM 18. *For sufficiently large n, Procedure* `SharedCoin` *implements a weak shared coin with constant agreement parameter.*

PROOF. Let T be the total number of votes generated.

The total vote Z_i computed by any process in Line 19 is equal to S_T minus at most one vote for each process because of the termination bit. From Lemma 9, these unwritten votes have total size bounded by $1 + \sum_{i=1}^{T} X_i^2$. We show there is at least a constant probability that both $1 + \sum_{i=1}^{T} X_i^2 \leq 14 + 4/n$ and $S_T > 15$, which implies that for sufficiently large n there is a constant probability for having $Z_i > 0$ for all i, and therefore all processes agree on the value $+1$.

From Lemma 15, the probability that $\sum_{i=1}^{T} X_i^2 > 13 + 4/n$ is $o(1)$. From Lemma 17, the probability that $S_{\tau'} \leq 15$ is at most $1 - (1/8)p_{19}$. The probability that neither of the above events hold is at least $(1/8)p_{19} - o(1)$, which is at least some constant δ, and in this case we have $S_T \geq S_{\tau'} > 15$ and $1 + \sum_{i=1}^{T} X_i^2 \leq 14 + 4/n$.

This proves that there is a constant probability of all processes deciding $+1$; the same results hold for -1 by symmetry. □

4. DISCUSSION

We have constructed an approximate counter for a shared-memory system, that works under a strong adversary which can decide upon its scheduling adaptively, by observing the execution so far, including the results of local coin-flips. Incrementing and reading the counter require sublinear work, and any read operation has a high probability of returning a value which is at most a fraction of δ less than the number of increments that have finished before the read started, and at most a fraction of δ more than the number of increments that have started before the read finished.

We have shown an application of this approximate counter in a shared coin protocol with $O(n)$ individual work, and hence $O(n^2)$ total work. This implies a randomized consensus protocol with the same complexities, which improve upon the best previously known protocol of $O(n \log n)$ individual work [Aspnes et al. 2008], and are tight due to the $\Omega(n^2)$ lower bound of [Attiya and Censor 2007].

While the approximate counter of Section 2 is highly specialized for our particular application, the underlying techniques seem fairly general. We believe that further improvements could give an approximate counter with much better complexity and fewer restrictions on its use.

The analysis of our shared coin protocol is asymptotic. While the $O(n)$ individual work bound holds with a reasonably small constant for all values of n, the bound on the agreement parameter is proved only for values of n that are quite large, and the agreement parameter itself is very small. This is in contrast to the many shared coin protocols based on *unweighted* voting [Aspnes and Herlihy 1990; Aspnes 1993; Bracha and Rachman 1991; Saks et al. 1991] culminating in the $O(n^2)$ total work protocol of [Attiya and Censor 2007], where both the protocols and their proofs are relatively simple, work even for small n, and give highly respectable agreement parameters. Though the theoretical question of the asymptotic individual work complexity of randomized wait-free consensus is now settled, the resulting algorithm is still likely to be quite expensive, and it is an intriguing open question whether a *practical* algorithm with linear individual work can be obtained.

ACKNOWLEDGMENTS

The authors would like to thank Dana Angluin and David Eisenstat for useful discussions and Hagit Attiya for many helpful comments and suggestions. Comments on an earlier version by the anonymous referees are appreciated.

REFERENCES

Abrahamson, K. 1988. On achieving consensus using a shared memory. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. 291–302.

Alon, N. and Spencer, J. H. 1992. *The Probabilistic Method.* John Wiley & Sons.

Aspnes, J. 1993. Time- and space-efficient randomized consensus. *Journal of Algorithms 14,* 3 (May), 414–431.

Aspnes, J. 1998. Lower bounds for distributed coin-flipping and randomized consensus. *45,* 3 (May), 415–450.

Aspnes, J., Attiya, H., and Censor, K. 2008. Randomized consensus in expected $O(n \log n)$ individual work. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing (PODC)*. 325–334.

Aspnes, J. and Herlihy, M. 1990. Fast randomized consensus using shared memory. *Journal of Algorithms 11,* 3 (Sept.), 441–461.

Aspnes, J. and Waarts, O. 1996. Randomized consensus in expected $O(N \log^2 N)$ operations per processor. *SIAM J. Comput. 25,* 5 (Oct.), 1024–1044.

Attiya, H. and Censor, K. 2007. Tight bounds for asynchronous randomized consensus. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing (STOC)*. 155–164.

Attiya, H., Guerraoui, R., Hendler, D., and Kouznetsov, P. 2006. Synchronizing without locks is inherently expensive. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing (PODC)*. 300–307.

Bracha, G. and Rachman, O. 1991. Randomized consensus in expected $O(n^2 \log n)$ operations. In *Distributed Algorithms, 5th International Workshop*, S. Toueg, P. G. Spirakis, and L. M. Kirousis, Eds. Lecture Notes in Computer Science, vol. 579. Springer, 1992, Delphi, Greece, 143–150.

Fich, F. E., Hendler, D., and Shavit, N. 2005. Linear lower bounds on real-world implementations of concurrent objects. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 165–173.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. 1985. Impossibility of distributed consensus with one faulty process. *32,* 2 (Apr.), 374–382.

Flajolet, P. 1985. Approximate counting: a detailed analysis. *BIT 25,* 1, 113–134.

Flajolet, P. and Martin, G. N. 1985. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci. 31,* 2, 182–209.

Grimmet, G. R. and Stirzaker, D. R. 1992. *Probability and Random Processes*, 2nd ed. Oxford Science Publications.

Guruswami, V., Umans, C., and Vadhan, S. 2007. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. In *Proceedings of the 22nd Annual IEEE Conference on Computational Complexity (CCC '07)*. 96–108.

Hall, P. and Heyde, C. 1980. *Martingale Limit Theory and Its Application.* Academic Press.

Herlihy, M. 1991. Wait-free synchronization. *ACM Trans. Program. Lang. Syst. 13,* 1 (January), 124–149.

Herlihy, M. P. and Wing, J. M. 1990. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems 12,* 3 (July), 463–492.

Hoory, S., Linial, N., and Wigderson, A. 2006. Expander graphs and their applications. *Bulletin (new series) of the American Mathematical Society 43,* 4 (Oct.), 439–561.

Loui, M. C. and Abu-Amara, H. H. 1987. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 163–183.

Mitzenmacher, M. and Upfal, E. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press.

Morris, R. 1978. Counting large numbers of events in small registers. *Commun. ACM 21,* 10, 840–842.

SAKS, M., SHAVIT, N., AND WOLL, H. 1991. Optimal time randomized consensus—making resilient algorithms fast in practice. In *Proceedings of the 2nd annual ACM-SIAM symposium on Discrete algorithms (SODA)*. 351–362.

WORMALD, N. C. 1999. The differential equation method for random graph processes and greedy algorithms. In *Lectures on Approximation and Randomized Algorithms*, M. Karonski and H. J. Proemel, Eds. PWN, 73–155.