Approximate Shared-Memory Counting Despite a Strong Adversary

James Aspnes* Keren Censor†

Abstract

A new randomized asynchronous shared-memory data structure is given for implementing an approximate counter that can be incremented up to n times. 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 sublinear solution to this problem that 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.

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 one incrementer overwrites values left by another. The simplest implementation, where each incrementer writes its increment to a separate location in an array of registers, requires n registers operations when collecting the array for the read operation. Even with more sophisticated primitives, there are still strong lower bounds on the time- and space-complexity for exact counting in a distributed system [9, 11].

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 [13, 14, 23], 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. 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 $(1-\delta)k \leq R \leq (1+\delta)K$, 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**.*

*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.

†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.

[1] This is a weaker condition than, say, linearizability [19], because it makes no consistency guarantees on the values returned by different read operations.

Our paper makes two main contributions: a sublinear-work implementation of an approximate counter for a strong-adversary model that is δ-accurate with high probability, and an application of this counter to obtained a protocol for **randomized consensus** 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 [12, 18, 21]. 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 one process (**per-process** or **individual work**).

Many randomized consensus protocols were designed for the asynchronous model under a strong adversary, The first of these, due to Abrahamson [1], required exponential work. Subsequent work (e.g., [3, 6, 24]) 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 [10]. Further progress in reducing total operations stalled at this point, although Aspnes and Waarts [7] 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 [4] (which implies an $\Omega(n/\log^2 n)$ lower bound on individual work) showed that significant further improvements were unlikely, although the polylogarithmic gap between the known upper and lower bounds remained a continuing annoyance.

This gap was closed for total work by Attiya and Censor [8], 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.* [5] 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 wait-free randomized consensus can be solved 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 [6] 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 output of the protocol is arbitrary; the adversary may determine the outputs of the processes or even arrange for them to disagree. The reduction in [6] 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 goal then is to construct a weak shared coin with constant agreement parameter and low cost.[2]

Essentially all known shared coins are based on random voting, with some variation in how votes are collected and how termination is detected. The particular version we use appeared first in the $O(n^2 \log n)$ total work protocol of Bracha and Rachman [10], 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 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 [7], 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 [5], 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 [5,7] that simplify the analysis (which is to appear in a full version of this paper).

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. 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.

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.

[2] Smaller agreement parameters do not appear to be helpful, although the consensus protocol of [3], which does not use the reduction of [6], effectively embeds a weak shared coin with agreement parameter $\Theta(1/n)$.

```
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] ← 1;
```

Procedure `SamplingIncrement`

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

Procedure `SamplingRead`

For the lower bound, we pretend that any bit not set by the start of the call to `SamplingRead` does not contribute to the total; this gives a fixed set of bits to consider, and allows us to apply standard Chernoff bounds. For the upper bound, the situation is more complicated, as 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 2.1. *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 at time t and let k_t be the number of processes that have executed the write operation in Line 2 of* `SamplingIncrement` *by time t. 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 time 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., [22, 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

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

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

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 time 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 $\sum_{r=1}^{N} a_t[r] \ge k_t - \sum_{i=1}^{n} X_i \ge k_t - \sum_{i=1}^{n} Y_i$.

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 2.2. *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`*. The 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 from a `SamplingRead` operation is too high.

LEMMA 2.3. *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 random variable equal to 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, 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 $|X_{i+1} - X_i| \leq 1$. If we can show in addition the supermartingale property $\mathrm{E}[X_{i+1}|X_1 \ldots X_i] \leq X_i$, we can apply the supermartingale version of Azuma's inequality [25, 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 $\mathrm{E}[X_{i+1} - X_i|X_1 \ldots X_i] \leq 0$ and thus $\mathrm{E}[X_{i+1}|X_1 \ldots X_i] \leq X_i$.

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

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

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 $S_s \geq X_s + (s/N)K_s$. The value returned by `SamplingIncrement` is $R = (s/N)S_s$ and therefore $\Pr[R \geq \alpha(N/s) + K_s] \leq \Pr[S_s \geq \alpha + (s/N)K_s] \leq \Pr[X_s \geq \alpha] \leq \exp\left(-\frac{\alpha^2}{2s}\right)$.

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

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

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 small (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 2.1. *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 2.1, 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 $k \geq n^{4/5}$ completed increments, we have $|A| \geq k - n^{4/5-\epsilon/4} \geq k(1-n^{-\epsilon/4}) \geq k(1-\delta/2)$. 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 2.2 is at most

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

again for sufficiently large n.

Finally, we consider the possibility that some call to `SamplingRead` returns a value that is too high. By Lemma 2.3 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 2.1 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 $k \leq K_{\max} = n^{3/4+2\epsilon}$. This data structure also uses an array of bits, which are now multi-writer. 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 scanning 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 [16] 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 [20] for a recent survey. We use an explicit construction of an **unbalanced bipartite expander** of Guruswami *et al.* [16]. 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 $\Gamma : [N] \times [D] \to [M]$ 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 [16] is the following theorem:

THEOREM 2.2. ([16]) *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 $\Gamma : [N] \times [D] \to [M]$ 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 scans 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 ← 1 to D do
2      b[Γ(pid, i)] ← 1
3  end
```

Procedure `ExpanderIncrement`.

It remains to choose the parameters of the expander. The value δ we will use in the expander is

```
1 return (1/D) Σ_{i=1}^{M} b[i]
```
Procedure `ExpanderRead`

```
1 SamplingIncrement();
2 ExpanderIncrement();
```
Procedure `ApproxIncrement`.

```
1 S ← ExpanderRead();
2 if S < (1 − δ)n^{4/5+ε/4} then
3     return S;
4 else
5     return SamplingRead();
6 end
```
Procedure `ApproxRead`

the same δ used for the sampling counter. We assume as in Theorem 2.1 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 the minimum accurate count $n^{4/5+\epsilon/4}$ for the sampling counter. We accomplish these goals by choosing α so that $(4/5+\epsilon/4)(1+\alpha) = 4/5+\epsilon$.

Set $N = n$, $K_{\max} = n^{4/5+\epsilon/4}$, and $\alpha = \frac{15\epsilon}{16+5\epsilon}$. Further, set $D = O\left((\log N)(\log K_{\max})/\delta\right)^{1+1/\alpha} = O\left((\log n)^{76/(25\epsilon)}(1/\delta)^{8/(5\epsilon)}\right)$ and $M \leq D^2 \cdot K_{\max}^{1+\alpha} = O\left(n^{4/5+\epsilon}(\log n)^{152/(25\epsilon)}(1/\delta)^{16/(5\epsilon)}\right)$.

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

LEMMA 2.4. *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 $(1-\delta)\min(k, n^{4/5+\epsilon/4}) \leq R \leq K$.*

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 $(1-\delta)\min(k, K_{\max})D \leq |S_0| \leq |S| \leq |S_1| \leq KD$. But then $(1-\delta)\min(k, K_{\max}) \leq \frac{1}{D}|S| \leq K$

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 2.1 and Lemma 2.4.

THEOREM 2.3. *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 2.1. 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 2.4 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 2.1 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.* [5] to match the $\Omega(n)$ lower bound of Attiya and Censor [8]. 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.

We use a standard reduction [6] 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`.

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 the full version of this paper. 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.

The full analysis of the shared coin protocol will appear in a full version of this paper. Below we describe the main tools used to obtain the time complexity and agreement parameter of the shared coin.

```
   shared data: array c[0..(2 log n)] of
                approximate counters with
                parameters δ = 1/2 and ε = 1/10,
                array votes[1..n] of single-writer
                registers, multi-writer bit done
 1 i ← 0;
 2 v_0 ← 0;
 3 varianceWritten ← 0;
 4 while v_i < 1 and not done do
 5     i ← i + 1;
 6     w_i = min(max(v_{i-1}, 1/n), 1/√n);
 7     v_i = v_{i-1} + w_i^2;
 8     vote = LocalCoin() · w_i;
 9     votes[pid] ← votes[pid] + vote;
10     if v_i ≥ 2^varianceWritten / n^2 then
11         ApproxIncrement(c[varianceWritten]);
12         varianceWritten ← varianceWritten + 1;
13         if Σ_{k=0}^{2 log n} (2^k · ApproxRead(c[k])) ≥ 3n^2
           then
14             break ;
15         end
16     end
17 end
18 done ← true;
19 return sgn(Σ_p votes[p]);
```

Procedure `SharedCoin`

First, we prove properties of the weight function, which allow us to bound the expected individual work of each process, as stated next.

LEMMA 3.1. *Procedure* `SharedCoin` *executes $O(n)$ local coin-flips and $O(n)$ register operations.*

Next, we prove that the shared coin protocol has a constant agreement parameter. 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 prove upper and lower bounds on the total variance of all the generated votes, as stated in the following lemma.

LEMMA 3.2. *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. $\Pr[V < 1] \leq n(2\log n+1)^2 \exp\left(\frac{-n^{1/20}}{8}\right)(1+o(1))$,

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

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 significantly 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 $X_1 \ldots X_{t-1}$, but because the adversary cannot predict the outcome of `LocalCoin`, the expectation of X_t is zero even conditioning on the previous votes. That $\mathrm{E}[X_t = 0 | X_1, \ldots X_{t-1}]$ is the defining property of a class of stochastic processes known as **martingales** (see [2, 15, 17]); 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, [17, Theorem 3.2] which we use in the proof of Lemma 3.3; and as with independent variables, the variance of S_t is equal to the sum of the variances of X_1 through X_t, [17, p. 8] a fact we use in the proof of Lemma 3.4.

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 [15, 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 [15, Theorem 12.4.5], as is the sequence $S''_t = \sum_{i=1}^{t} X_{\tau+i}$ [15, 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 . 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 $X_1 \ldots X_t$. 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 [17].

LEMMA 3.3. *For any fixed α and n sufficiently large, there is a constant probability p_α such that, for any adversary strategy, $\Pr[S_\tau \geq \alpha] \geq p_\alpha$.*

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 3.2, (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 3.4. *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 3.3 that S_τ is at least 19. Then for sufficiently large n and any adversary strategy, $\Pr[S_{\tau'} > 15] \geq (1/8)p_{19}$.*

This leads us to the final result:

THEOREM 3.1. *For sufficiently large n, Procedure `SharedCoin` implements a weak shared coin with constant agreement parameter.*

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 [5], and are tight due to the $\Omega(n^2)$ lower bound of [8].

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 [3,6,10,24] culminating in the $O(n^2)$ total work protocol of [8], 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.

Acknowledgements: The authors would like to thank Dana Angluin and David Eisenstat for useful discussions and Hagit Attiya for many helpful comments and suggestions.

References

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

[2] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. John Wiley & Sons, 1992.

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

[4] James Aspnes. Lower bounds for distributed coin-flipping and randomized consensus. *Journal of the ACM*, 45(3):415–450, May 1998.

[5] James Aspnes, Hagit Attiya, and Keren Censor. Randomized consensus in expected $O(n \log n)$ individual work. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing (PODC)*, pages 325–334, 2008.

[6] James Aspnes and Maurice Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3):441–461, September 1990.

[7] James Aspnes and Orli Waarts. Randomized consensus in expected $O(N \log^2 N)$ operations per processor. *SIAM Journal on Computing*, 25(5):1024–1044, October 1996.

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

[9] Hagit Attiya, Rachid Guerraoui, Danny Hendler, and Petr Kouznetsov. Synchronizing without locks is inherently expensive. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing (PODC)*, pages 300–307, 2006.

[10] Gabriel Bracha and Ophir Rachman. Randomized consensus in expected $O(n^2 \log n)$ operations. In Sam Toueg, Paul G. Spirakis, and Lefteris M. Kirousis, editors, *Distributed Algorithms, 5th International Workshop*, volume 579 of *Lecture Notes in Computer Science*, pages 143–150, Delphi, Greece, 7–9 October 1991. Springer, 1992.

[11] Faith Ellen Fich, Danny Hendler, and Nir Shavit. Linear lower bounds on real-world implementations of concurrent objects. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 165–173, 2005.

[12] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[13] Philippe Flajolet. Approximate counting: a detailed analysis. *BIT*, 25(1):113–134, 1985.

[14] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[15] G. R. Grimmet and D. R. Stirzaker. *Probability and Random Processes*. Oxford Science Publications, 2nd edition, 1992.

[16] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. In *Proceedings of the 22nd Annual IEEE Conference on Computational Complexity (CCC '07)*, pages 96–108, June 2007.

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

[18] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124–149, January 1991.

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

[20] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin (new series) of the American Mathematical Society*, 43(4):439–561, October 2006.

[21] Michael C. Loui and Hosame H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, pages 163–183, 1987.

[22] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

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

[24] Michael Saks, Nir Shavit, and Heather Woll. Optimal time randomized consensus—making resilient algorithms fast in practice. In *Proceedings of the 2nd annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 351–362, 1991.

[25] Nicholas C. Wormald. The differential equation method for random graph processes and greedy algorithms. In M. Karonski and H. J. Proemel, editors, *Lectures on Approximation and Randomized Algorithms*, pages 73–155. PWN, 1999.