Randomized Consensus in Expected $O(n^2)$ Total Work Using Single-Writer Registers

James Aspnes*

Yale University

Abstract. A new weak shared coin protocol yields a randomized wait-free shared-memory consensus protocol that uses an optimal $O(n^2)$ expected total work with single-writer registers despite asynchrony and process crashes. Previously, no protocol was known that achieved this bound without using multi-writer registers.

1 Introduction

The **consensus** problem is to get a group of n processes to agree on a bit. In a **wait-free randomized consensus** protocol, each process starts with an input bit and produces a decision bit; the protocol is correct if it satisfies **agreement**, where all processes that finish the protocol choose the same decision bit; **validity**, where every decision bit is equal to some process's input; and **probabilistic termination**, where every non-faulty process completes the protocol after a finite number of its own steps with probability 1. These conditions are required to hold despite asynchrony and up to $n-1$ crash failures.

We consider consensus in an asynchronous shared-memory system where the timing of events and failures is under the control of an **adaptive adversary**, one that can observe the complete state of the system—including the internal states of process—but that cannot predict future coin flips made by the processes.

Processes communicate by reading and writing a collection of **atomic registers**, which may be **single-writer** (only one particular process is allowed to write to each register) or **multi-writer** (any process can write to any register). In either case we assume that a register can be read by all processes. The cost of a protocol is measured by counting either the expected total number of operations carried out by all processes (the **total work** or **total step complexity**) or the maximum expected number of operations carried out by any single process (the **individual work** or **individual step complexity**). For either measure, a worst-case adversary is assumed.

Single-writer registers were used in most early shared-memory consensus protocols [1, 3, 6, 8, 11–14, 18]. More recent protocols [4, 7] have used multi-writer registers for increased efficiency. Though multi-writer registers can be implemented from single-writer registers (see, for example, [9, Theorem 10.9]), this imposes a linear blow-up in the costs, and the question of whether a stronger

* Supported in part by NSF grant CCF-0916389.

lower bound might apply to single-writer register protocols than to multi-writer register protocols has remained open [7]. We answer this question, by giving a wait-free randomized consensus protocol using only single-writer registers that matches the lower bound on total step complexity for multi-writer registers.

1.1 Prior Work

Wait-free randomized consensus can be solved using shared memory by reduction to a **weak shared coin** [5]. A weak shared coin is a protocol in which each process chooses a bit, with the property that there is some **agreement parameter** $\delta > 0$ such that for each possible value b, all processes choose b with probability at least δ for any adversary strategy. A construction of Aspnes and Herlihy [5], which uses only single-writer registers, shows that any weak shared coin protocol with expected total work $T(n)$ and agreement parameter δ can be used to solve consensus with $O\left(\left(n^2 + T\left(n\right)\right)/\delta\right)$ expected total work. In particular, finding a weak shared coin with total work $O(n^2)$ and constant agreement parameter gives consensus in $O(n^2)$ expected total work.

Though early randomized consensus protocols [1, 13] did not use a shared coin, much of the subsequent development of randomized consensus protocols for the adaptive-adversary model has been based on this technique [3, 4, 6–8, 11, 12, 14, 18]. The typical structure of a shared coin protocol is to have the processes collectively generate $\Theta(n^2)$ random ± 1 votes, and have each process choose what it sees as the majority value. The intuition is that after $\Theta(n^2)$ votes, the margin of the majority value is likely to be $\Omega(n)$, large enough that the adversary cannot disguise it by selectively delaying processes casting votes it doesn't like.

The main variation between protocols is in how they detect when enough votes have been cast. For many years, the best known protocol was that of Bracha and Rachman [12]. In this protocol, each process maintains in its own register both a count of how many votes it has generated so far and the sum of all of its votes. After every $n/\log n$ votes, a process collects all of the register values (by reading all n registers), and decides on the majority value if the sum of the counts exceeds n^2. The dominant cost is the cost of the collect, which amortizes at $O(\log n)$ register operations for each of the $O(n^2)$ total votes, giving $O(n^2 \log n)$ total work.

The reason for checking the vote count every $n/\log n$ steps is that it guarantees that at most $n^2/\log n$ **extra votes** can be generated once the initial n^2 **common votes** are cast. It can then be shown that (a) the common votes produce with constant probability a net majority at any fixed multiple of n, their standard deviation; while (b) using Hoeffding's inequality, the net extra votes seen by any one process have probability less than $\frac{1}{n^2}$ of exceeding $2n$, giving a low probability that any of the processes sees a large shift from the extra votes. Factoring in the additional shift of up to $n - 1$ votes that have been generated but not written still leaves a constant probability that all processes see the same majority value.

The main excess cost in the Bracha-Rachman protocol is the $\Theta(\log n)$ amortized cost per vote of doing a full collect every $n/\log n$ votes. This is needed

to keep the extra votes from diverging too much: if we collect only every $\Theta(n)$ votes, we would expect a constant probability for each process that the $\Theta(n^2)$ extra votes it sees change the majority value.

The goal of finding an $O(n^2)$ total-work shared coin with constant agreement parameter was finally achieved by Attiya and Censor using multi-writer registers [7] in a paper that also showed that $\Omega(n^2)$ total work was necessary for consensus. The key idea is to use Bracha-Rachman, modified to do collects every n votes, but add a single **termination bit** that shuts down voting immediately once some process detects termination. This makes all processes see essentially the same set of extra votes, meaning that it is no longer necessary to bound the effect of the extra votes separately for each process. However, a single multi-writer register is needed to implement the termination bit, even though the rest of the protocol uses only single-writer registers.

1.2 Our Approach

We show that the multi-writer bit is not needed: it can be replaced by an array of single-writer termination bits (one for each process) that propagate via a gossip protocol running in parallel with the voting mechanism at an amortized cost of $O(1)$ operations per vote. The intuition for why this works is that, as more processes detect termination, fewer processes are left voting. So while some processes may see a full $O(n^2)$ extra votes, later processes will see fewer; thus the probability that each process sees enough extra votes to shift the majority drops geometrically, giving a constant total probability that any process returns the wrong value. To make this work, a counting argument is used to show that the k-th process to detect termination sees at most $2n^2/k$ extra votes, and that this bound holds *simultaneously* for all k with constant probability. This avoids a blow-up that would otherwise occur using simple union bounds.

2 The Shared Coin Protocol

Code for the shared coin algorithm is given in Algorithm 1. The structure is similar to the protocol of Bracha and Rachman [12] as improved by Attiya and Censor [7], with the single termination bit of the Attiya-Censor protocol replaced by an array of termination bits that are sampled randomly. The essential idea is that a process repeatedly generates random ± 1 votes (using the `CoinFlip()` subroutine, which generates each value ± 1 with equal probability). These are added to the process's sum field in $a[\mathsf{pid}]$, while at the same time the number of votes the process has generated is written to the count field. Each process checks after every n votes to see if the sum of all the count fields exceeds a threshold $T = 64n^2$, and probes a random termination bit $\mathsf{done}[r]$ before every vote. If either enough votes have been generated or $\mathsf{done}[r]$ is set, the process exits the loop immediately, setting $\mathsf{done}[\mathsf{pid}]$ and returning the sign of the sum of all the sum fields.

shared data:
Register $a[p]$ for each process p, with fields $a[p].\mathsf{count}$ and $a[p].\mathsf{sum}$, both initially 0.
Boolean register $\mathsf{done}[p]$ for each process p, initially **false**.

```
1  for i ← 1...∞ do
2      if i mod n = 0 then
3          if Σ_{p=1}^{n} a[p].count ≥ T then break
4      end if
5      Choose r uniformly at random from {1...n} \ {pid}.
6      if done[r] = true then break
7      v ← CoinFlip()
8      a[pid] ← ⟨a[pid].count + 1, a[pid].sum + v⟩.
9  end for
10 done[pid] ← true
11 return sgn(Σ_{p=1}^{n} a[p].sum)
```

Algorithm 1: Shared coin protocol.

3 Analysis

For the analysis of the protocol, we fix an **adversary strategy**. This is a function that selects, after each initial prefix of the computation, which process will execute the next operation, leaving the results of the calls to `CoinFlip` and the random choices of done-bit probes as the only source of nondeterminism. We then wish to show that each outcome ± 1 is chosen by all processes with at least constant probability.

The essential idea is to show first that the sum of the votes generated before each process detects termination is likely to be large and then that the sums of the extra votes seen by each process p are likely to be small simultaneously for all p. In this case, no process's extra votes causes it to see a majority different from the common majority. This approach follows similar arguments used for previous protocols based on Bracha-Rachman [4,6,7,12]. The main new wrinkle is that we consider non-uniform error probabilities, where a process that sets $\mathsf{done}[p]$ early is more likely to return the wrong value than a process that sets it later.

We write that a process's i-th vote is **generated** when the process executes the call to `CoinFlip()` in Line 7. We will be more interested in when a vote is generated than when it is ultimately added to $a[p]$. We let $X_1, X_2, X_3, \ldots$ be random variables, with X_t representing the return value of the t-th call to `CoinFlip()` by any process, and let $S_t = \sum_{i=1}^{t} X_i$ be the sum of the first t generated votes. Because each X_i has expectation zero, the sequence $\{S_t\}$ is a

martingale, and we can use tools for bounding the evolution of martingales to characterize the total vote trajectory over time.[1]

Because we are examining votes when they are generated and not when they are written, we say that a process p **observes** a vote X_t generated by q if X_t is generated before p reads $a[q]$ during its final collect. The observed votes are not necessarily read by p or included in its final tally; but since at most one vote generated by q can be missing when p reads $a[q]$, the sum that p computes will differ from the sum it "observes" by at most $n-1$.

3.1 Overview of the Proof

We now give an outline of the structure of the proof:

1. We bound how far the values in the registers lag behind the generated votes (Lemma 1). This bound is used first to bound the total number of votes generated (in Lemma 2) and later to bound the gap between the generated votes observed by a process and the sum computed by that process during its final collect.
2. We show that the sum S_T of the first T votes is at least $8n$ with probability at least $1/8$ (Lemma 3). This $8n$ majority is our budget for losses in the later stages of the protocol.
3. If the preceding event holds, the probability that S_t ever drops below $4n$ during subsequent votes is shown to be less than $1/8$ (Lemma 4). This bound is obtained by combining Kolmogorov's inequality and the bound on the total number of votes from Lemma 2. A consequence is that it is likely that, for every process, S_t is above $4n$ when the process detects termination and begins its final collect.
4. While a process's final collect is in progress, extra votes may come in that reduce the value seen by the process below $4n$ (this does not contradict Lemma 4, because the adversary may be selective in when it allows the process to read particular registers). While different processes may see different numbers of extra votes (processes that detect termination later will have observe fewer other processes still generating votes), we can bound simultaneously the number of extra votes seen by each process as a function of the order in which they set their termination bit (Lemma 5).
5. The extra votes observed by each process also form a martingale, and thus the probability that they reduce the total by $3n$ or more can be bounded using Azuma's inequality [10]. Even though the different number of extra votes observed by each process varies, the resulting probability bounds form a geometric series that sums to less than $1/8$ (Lemma 6).
6. After subtracting the $3n$ bound on extra votes from the $4n$ bound of the previous step, we have a constant probability that the number of votes observed by every process is at least n. Since the actual total read by each process

[1] A good general reference on martingales (and other stochastic processes) is [15]. Discussion of applications of martingales to analysis of algorithms can be found in [2, 16, 17].

differs from the observed value by at most $n-1$, this gives that every process sees at least $+1$ votes, proving agreement (Theorem 1).

The choice of the thresholds $8n$ and $4n$ is somewhat arbitrary; these particular values happen to be convenient for the proof, but it is likely that further optimization is possible. The threshold n in the last step is needed because of the gap between generated votes and the values actually stored in the registers.

We now proceed with the details of the proof.

3.2 Deterministic Bounds on Error and Running Time

The following lemma bounds the difference between the generated votes and the values in the registers:

Lemma 1. *Let $\gamma_{pi} = 1$ if vote X_i is generated by p, and 0 otherwise. In any state of the protocol after exactly t calls to* `CoinFlip()` *have been made, we have:*

1. $t - n \leq \sum_{p=1}^{n} a[p].\mathsf{count} \leq t$.
2. *For all p,* $\left|\left(\sum_{i=1}^{t} \gamma_{pi} X_i\right) - a[p].\mathsf{sum}\right| \leq 1$.

Proof. Immediate from inspection of the protocol and the observation that for each process, there can be at most one vote that has been generated in Line 7 but not yet written in Line 8.

Lemma 2. *The total number of votes τ generated during any execution of the protocol is at least T and at most $T + n^2 + n$.*

Proof. Before a process p can finish, it must first write $\mathsf{done}[p]$ in Line 10. Consider the first process to do so. This process can only exit the loop after seeing $\sum_{p=1}^{n} a[p].\mathsf{count} \geq T$, which occurs only if at least T votes have already been generated (and written).

For the upper bound, suppose that at some time, $T + n$ votes have been generated; then from Lemma 1 we have $\sum_{p=1}^{n} a[p].\mathsf{count} \geq T$. Each process can generate at most n more votes before executing the test in Line 3 and exiting. Adding in these votes, summed over all processes, gives a total of at most $(T + n) + n^2 = T + n^2 + n$ votes.

It is easy to see from Lemma 2 that the total work for Algorithm 1 is $O(n^2)$; the main loop contributes an amortized $O(1)$ operations per vote (plus an extra $O(n)$ operations per process for the first execution of the collect in Line 3), while the assignment to $\mathsf{done}[\mathsf{pid}]$ and the final collect contributes $O(n)$ more operations per process. Summing these contributions gives the claimed bound.

3.3 Common Votes and Extra Votes

For each process p, let κ_p be the number of votes generated before p either observes a total number of votes greater than or equal to T in Line 3 or observes

a non-**false** value in done$[r]$ in Line 6. For each t, let ξ_{pt} be the indicator variable for the event that both $t > \kappa_p$ and a vote X_t is generated by some process q before p reads $a[q]$ in Line 11; formally, $\xi_{pt} = [t > \kappa_p] \wedge \gamma_{pt}$, where γ_{pt} is the random variable defined in Lemma 1. Since all votes $X_1 \ldots X_{\kappa_p}$ are generated before p executes Line 11, the sum over all q of votes generated by q before p reads $a[q]$ is given by $S_{\kappa_p} + \sum_t \xi_{pt} X_t$. We will refer to these votes as the votes **observed** by p, even though (by Lemma 1) up to one such vote for each process q may not be written to $a[q]$ before p reads it.

We will show that, with constant probability, the total votes observed by every process is bounded away from 0 by at least n in the same direction. The essentially idea is to show that $\left|S_{\kappa_p}\right|$ is likely to be large for all p, and that the extra votes $|\sum_t \xi_{pt} X_t|$ are likely to all be small. This argument essentially follows the structure of the proofs of correctness for the shared coins in [12] and subsequent work [4, 6, 7]. The new part is a trick for simultaneously bounding the number of extra votes observed by each process p after time κ_p, based on the effect of the random sampling of the done bits.

3.4 Bound on Common Votes

To simplify the argument, we concentrate on the case where all processes see a positive total vote; the negative case is symmetric. First we consider the effect of the pre-threshold votes:

Lemma 3. *For sufficiently large n,* $\Pr[S_T \geq 8n] \geq 1/8$.

Proof. Immediate from the normal approximation to the binomial distribution: $8n = \sqrt{T}$ is one standard deviation.

For the remainder of the proof we consider only events that occur after the T-th vote is generated. The bounds we obtain will thus hold independently of the value of S_T.

First, we use Kolmogorov's inequality (following the approach in [7]) to bound how far S_t can drop after S_T.

Lemma 4.

$$\Pr\left[\min_{t \geq T} (S_t - S_T) \leq -4n\right] \leq \frac{1}{8}.$$

Proof. From Lemma 2, there are at most $n^2 + n$ votes after T, giving a total variance of at most $n^2 + n$. Thus,

$$\begin{aligned}
\Pr\left[\min_{t \geq T} (S_t - S_T) \leq -4n\right] &\leq \Pr\left[\max_{t \geq T} |S_t - S_T| \geq 4n\right] \\
&\leq \frac{n^2 + n}{(4n)^2} \\
&= \frac{1}{16} + \frac{1}{16n} \\
&\leq \frac{1}{8},
\end{aligned}$$

where the second inequality follows from Kolmogorov's inequality and the last inequality holds for $n \geq 1$.

In particular, we have that, with probability at least 7/8, $S_{\kappa_p} \geq S_T - 4n$ for all p.

3.5 Bound on Extra Votes

We now consider the votes generated after a process detects termination but before it finishes its final collect in Line 11; i.e., those votes X_t for which $\xi_{pt} = 1$. Notice that for each p and t, the value of ξ_{pt} is determined before X_t is generated; formally, ξ_{pt} is measurable $\mathcal{F}_{t-1}$, where $\mathcal{F}_{t-1}$ records all events prior to the generation of X_t. It follows that $\mathrm{E}[\xi_{pt} X_t | \mathcal{F}_{t-1}] = \xi_{pt} \mathrm{E}[X_t | \mathcal{F}_{t-1}] = 0$, since $\mathrm{E}[X_t | \mathcal{F}_{t-1}] = 0$. This implies that extra votes observed by p form a martingale difference sequence and their sum can be bounded using Azuma's inequality [10]. If $n_p = \sum \xi_{pt}$, then $\Pr[\sum \xi_{pt} X_t \leq -2n] \leq \exp(-4n^2/n_p^2)$, and the probability that any process p observes $\sum \xi_{pt} X_t \leq -3n$ is bounded by $\sum_p \exp(-9n^2/n_p^2)$ by the union bound. The main trick is to show that, with constant probability, most n_p values are small enough that this sum is a small constant.

The basic idea behind this trick is that if a particular vote generated by some process q might be an extra vote for many processes, then these processes must all have written their done bits, making it more likely that q will read **true** on its next probe of the done array and finish. In particular, this means that the k-th process to write its done bit will observe at most n/k extra votes on average from q, and n^2/k extra votes on average from all processes. By itself, this would give a probability-(1/2) bound of $2n^2/k$ on the number of extra votes observed by p_k using Markov's inequality. But in fact we can show the stronger result that this bound holds simultaneously for all p_k with probability 1/2, by bounding the sum of the number of termination bits set when each vote is generated and arguing that each of p_k's extra votes contributes at least k to this sum. The result is the following:

Lemma 5. *Let p_k be the k-th process to write its* done *bit. With probability at least 1/2, it holds simultaneously for all k that $n_{p_k} = \sum_t \xi_{p_k t} \leq 2n^2/k$.*

Proof. For each vote X_t, let its **contribution** c_t be $\sum_k \xi_{p_k t}$. Consider the sequence of votes $X_{t_1}, X_{t_2}, \ldots$ generated by some single process q. If one of these votes X_{t_i} has contribution w_{t_i}, then q's next probe of the done array will find **true** with probability at least k/n; in other words, with probability at least k/n, a contribution k vote is the last vote q generates. Letting C be the expected total contribution of q's votes, we get the recurrence

$$C \leq k + (1 - k/n)C,$$

which has the solution

$$C \leq n,$$

no matter how the adversary chooses k. It follows that the expected total contribution of all votes cast by q is at most n, and thus that the expected total contribution $\sum_t c_t$ of all votes cast by all processes is at most n^2. By Markov's inequality, we have

$$\Pr\left[\sum_t c_t \leq 2n^2\right] \geq 1/2.$$

Now observe that process p_k only observes extra votes of contribution k or greater, since all other votes were generated before p_k wrote its done bit. So $\xi_{p_k t} = 1$ implies $c_t \geq k$, and thus

$$\begin{aligned}\sum_t \xi_{p_k t} &= \frac{1}{k}\sum_t \xi_{p_k t} k \\ &\leq \frac{1}{k}\sum_t \xi_{p_k t} c_t \\ &\leq \frac{1}{k}\sum_t c_t.\end{aligned}$$

This holds for all k. So in the event that $\sum_t c_t \leq 2n^2$, which occurs with probability at least $1/2$, we have $\sum_t \xi_{p_k t} \leq 2n^2/k$ for all k.

We now bound the extra votes for each process. To avoid dependencies, we define a truncated version of the extra votes for each process p_k, capped at $\lfloor 2n^2/k \rfloor$ votes; when the bound in Lemma 5 holds, this will be equal to the actual extra votes. Let $\xi'_{p_k t} = \xi_{p_k t}$ if $\left(\sum_{s<t} \xi_{p_k s}\right) < \lfloor 2n^2/k \rfloor$ and 0 otherwise. Then $\xi'_{p_k t}$ is predictable and the sequence $\left\{\sum_{i \leq t} \xi'_{p_k i} X_i\right\}$ is a martingale. If we consider just the sequence of values $X_{i_1}, X_{i_2}, \ldots$ for which $\xi'_{p_k i} = 1$, we have a bounded martingale difference sequence of length at most $2n^2/k$. It follows from Azuma's inequality that

$$\begin{aligned}\Pr\left[\sum_t \xi'_{p_k t} X_t \leq -3n\right] &\leq \exp\left(-\frac{(3n)^2}{2(2n^2/k)}\right) \\ &= e^{-(9/4)k}. \qquad (1)\end{aligned}$$

Summing this quantity for all k gives

Lemma 6. $\Pr\left[\exists k \sum_t \xi'_{p_k t} X_t \leq -3n\right] < 1/8$.

Proof. Using the union bound and (1):

$$\begin{aligned}
\Pr\left[\exists k \sum_t \xi'_{p_k t} X_t \leq -3n\right] &\leq \sum_{k=1}^{n} \Pr\left[\sum_t \xi'_{p_k t} X_t \leq -3n\right] \\
&\leq \sum_{k=1}^{n} e^{-(9/4)k} \\
&= \sum_{k=1}^{n} \left(e^{-9/4}\right)^k \\
&\leq \frac{e^{-9/4}}{1 - e^{-9/4}} \\
&\approx 0.1178\ldots \\
&< 1/8.
\end{aligned}$$

3.6 Full Result

We can now state the full result:

Theorem 1. *Algorithm 1 implements a shared coin with agreement parameter* $1/32$.

Proof. From Lemma 3, the probability that $S_T \geq 8n$ is at least $1/8$. Suppose that this event occurs; we now consider the probability that subsequent events involving $X_{T+1}\ldots$ cause some process to compute a non-positive total vote.

Recall that each process p observes a total vote $S_{\kappa_p} + \sum_t \xi_{pt} X_t$, and that the actual vote computed by the process may be as low as $S_{\kappa_p} + \sum_t \xi_{pt} X_t - (n-1)$ (the $n-1$ is from the unwritten votes described in Lemma 1). Lemma 4 gives a probability of at most $1/8$ that any S_{κ_p} is less than $S_T - 4n$. Lemma 5 gives a probability of at most $1/2$ that $\sum_t \xi_{pt} \neq \sum_t \xi'_{pt}$ for any p. Finally, Lemma 6 gives a probability of at most $1/8$ that $\sum_t \xi'_{pt} \leq -3n$ for any p. These probabilities sum to $3/4$; so there is a probability of at least $1/4$ that none of these bad events occur, in which case the total vote computed by p is at least $S_T - 4n - 3n - (n-1) = S_T - 8n + 1$.

When $S_T \geq 8n$, this quantity is at least 1, and thus all processes return $+1$ with probability at least $(1/8)(1/4) = 1/32$. That all processes return -1 with at least the same probability follows from symmetry of the algorithm.

Corollary 1. *There is a wait-free randomized shared-memory consensus protocol using only-single writer registers that executes* $O(n^2)$ *expected total operations against an adaptive adversary.*

4 Conclusions

We have shown that the expected total work needed for randomized shared-memory consensus with an adaptive adversary is $O(n^2)$ using only single-writer

registers, matching the upper and lower bounds for multi-writer registers of Attiya and Censor [7]. This leaves the question of what happens with individual work. Here the best known lower bound is $\Omega(n)$, which matches the Attiya-Censor lower bound and which is immediate from the need for a process in a solo execution to read at least one register belonging to each other process before deciding. The best known upper bound is given by an $O(n \log^2 n)$ algorithm of Aspnes and Waarts [6] that modifies the Bracha-Rachman protocol by having processes generate votes of increasing weight. We suspect that the gossip-based termination detector used here might be able to reduce this upper bound to $O(n \log n)$, but that closing the gap completely will likely require new techniques.

5 Acknowledgments

I would like to thank Hagit Attiya and Keren Censor for suggesting the possibility of faster single-writer consensus and for several useful discussions.

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 and Keren Censor. Approximate shared-memory counting despite a strong adversary. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 441–450, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.
5. James Aspnes and Maurice Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3):441–461, September 1990.
6. 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.
7. Hagit Attiya and Keren Censor. Tight bounds for asynchronous randomized consensus. *Journal of the ACM*, 55(5):20, October 2008.
8. Hagit Attiya, Danny Dolev, and Nir Shavit. Bounded polynomial randomized consensus. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, pages 281–293, Edmonton, Alberta, Canada, 14–16 August 1989.
9. Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. John Wiley & Sons, second edition, 2004.
10. Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tôhoku Mathematical Journal*, 19(3):357–367, 1967.
11. Gabriel Bracha and Ophir Rachman. Approximated counters and randomized consensus. Technical Report 662, Technion, 1990.

12. 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.
13. Benny Chor, Amos Israeli, and Ming Li. Wait-free consensus using asynchronous hardware. *SIAM J. Comput.*, 23(4):701–712, 1994.
14. Cynthia Dwork, Maurice Herlihy, Serge Plotkin, and Orli Waarts. Time-lapse snapshots. *SIAM Journal on Computing*, 28(5):1848–1874, 1999.
15. Geoffrey R. Grimmett and David R. Stirzaker. *Probability and Random Processes.* Oxford University Press, 2001.
16. Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.
17. Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
18. 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*, pages 351–362, 1991.