Randomized Consensus in Expected O(n log n) Individual Work

James Aspnes*
Department of Computer Science, Yale University
New Haven, CT
aspnes@cs.yale.edu

Hagit Attiya†
Department of Computer Science, Technion
Haifa, Israel
hagit@cs.technion.ac.il

Keren Censor†‡
Department of Computer Science, Technion
Haifa, Israel
ckeren@cs.technion.ac.il

ABSTRACT

This paper presents a new randomized algorithm for achieving consensus among asynchronous processes that communicate by reading and writing shared registers, in the presence of a strong adversary. The fastest previously known algorithm requires a process to perform an expected $O(n \log^2 n)$ read and write operations in the worst case. In our algorithm, each process executes at most an expected $O(n \log n)$ read and write operations. It is shown that shared-coin algorithms can be combined together to yield an algorithm with $O(n \log n)$ individual work and $O(n^2)$ total work.

Categories and Subject Descriptors

D.1.3 [**Software**]: Programming Techniques—*Concurrent programming*; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*Stochastic processes*

General Terms

Algorithms, Theory

Keywords

distributed computing, shared memory, randomized algorithms, consensus

1. INTRODUCTION

Coordinating the actions of processes is crucial for virtually all distributed applications, especially in asynchronous systems. At the core of many coordination problems is the need to reach *consensus* among processes, despite the possibility of process failures. A consensus algorithm is a distributed algorithm where n processes collectively arrive at a common decision value starting from individual process inputs. It must satisfy *agreement* (all processes decide on the same value), *validity* (the decision value is an input to some process), and *termination* (all processes eventually decide). Consensus is a fundamental task in asynchronous systems, which can be employed to implement arbitrary concurrent objects [9]; consensus is also a key component of the state-machine approach for replicating services [10, 14].

There is no deterministic consensus algorithm in an asynchronous system, if one process may fail [8, 9, 11]. However, reaching consensus becomes possible using randomization with the termination condition relaxed to hold with probability 1. (The agreement and validity properties remain the same.) The complexity of solving consensus is measured by the expected number of register operations performed by *all* processes (*total work*) or by any one process (*per-process* or *individual work*). Many randomized consensus algorithms have been suggested, in different communication models and under various assumptions about the adversary (see [3]).

We study the consensus problem in 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' events is governed by a *strong* adversary that observes the results of the local coin flips before scheduling the next event; in particular, it may observe a coin-flip and, based on its outcome, choose whether or not that process may proceed with its next shared-memory operation.

Many randomized consensus algorithms were designed for this model, e.g., [1, 2, 4, 7, 13]. Attiya and Censor [6] have recently presented an algorithm with $O(n^2)$ total work; they also proved that $\Omega(n^2)$ is a lower bound on the total work. In their algorithm, however, a process running alone may have to perform all this work by itself, meaning that the individual work is also $\Theta(n^2)$. This is significantly higher than the $O(n \log^2 n)$ individual work achieved by the algorithm of Aspnes and Waarts [5], using only single-writer registers.

We show that wait-free randomized consensus can be solved using only atomic multi-writer multi-reader registers with $O(n \log n)$ expected individual work and $O(n^2)$ expected total work; the last figure matches the lower bound [6], while the first leaves a logarithmic gap.

*Supported in part by NSF grant CNS-0435201.
†Supported in part by the *Israel Science Foundation* (grant number 953/06).
‡Supported in part by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

PODC'08, August 18–21, 2008, Toronto, Ontario, Canada.

Our upper bound is based on a new weak shared-coin algorithm that requires each process to do at most $O(n \log n)$ operations. A *shared-coin* algorithm with *agreement parameter* δ allows processes to collectively "flip" a shared coin with probability at least δ for agreeing on each value. A shared-coin algorithm with a constant agreement parameter immediately yields a consensus algorithm with essentially the same complexities [4].

The algorithm is based on the weighted voting approach of the $O(n \log^2 n)$ individual-work algorithm of Aspnes and Waarts [5]. The improved complexity comes from applying the termination bit technique from the recent $O(n^2)$ total work algorithm [6].

While the individual work of our algorithm improves over the algorithm of Attiya and Censor [6], the $O(n^2 \log n)$ total work of our algorithm is inferior to its $O(n^2)$ total work.

We prove, however, that shared-coin algorithms can be *interleaved* in a way that obtains the best of their complexity figures. Given the power of the adversary to observe both protocols and adjust their scheduling, it is not obvious that this is the case, and indeed recent work of Lynch *et al.* [12] shows that undesired effects may follow from the interaction of an adaptive adversary and composed probabilistic protocols. Nonetheless, we can show that in the particular case of weak shared-coin algorithms, two algorithms with agreement parameter δ_A and δ_B can be composed with sufficient independence such that the combined protocol terminates with the minimum of the individual protocols' complexities in *both* individual and total work, while obtaining an agreement parameter of $\delta_A \cdot \delta_B$. Thus, by combining our shared coin with the $O(n^2)$ total work shared coin of [6], we obtain a protocol with both $O(n \log n)$ individual work and $O(n^2)$ total work.

The next section describes some previous shared-coin algorithms while Section 3 presents our new shared-coin algorithm. Section 4 shows how shared-coin algorithms can be interleaved. We conclude, in Section 5, with a discussion of our results and directions for future research.

2. PREVIOUS SHARED-COIN ALGORITHMS

A *weak shared coin* with *agreement parameter* δ is a distributed algorithm with the following properties:

(a) against any adversary strategy, with probability at least δ, every process returns -1,

(b) against any adversary strategy, with probability at least δ, every process returns $+1$.

The rest of the time the output of the algorithm is arbitrary; the adversary may determine the outputs of the processes or even arrange for them to disagree. A standard reduction [4] shows that a weak shared-coin algorithm with agreement parameter δ, expected individual work I, and expected total work T, yields a consensus algorithm with expected individual work $O(n + I/\delta)$ and expected total work $O(n^2 + T/\delta)$.

The question is how to construct a weak shared coin with constant agreement parameter and low cost.

All known weak shared coins for the strong-adversary model use some variant of the following voting scheme. The particular termination rule described below appeared first in the $O(n^2 \log n)$ total work algorithm of Bracha and Rachman [7]. The use of non-constant weights was introduced by Aspnes and Waarts [5].

1. Each process p_i generates random votes $X_{i,t} = \pm w_{i,t}$, where the two possible signs occur with equal probability. We say $w_{i,t}$ is the *weight* of the t-th vote generated by process p_i. Each process writes both its cumulative variance $\sum_t w_{i,t}^2$ and the total vote $\sum_t X_{i,t}$ to a single-writer register.

2. The algorithm starts terminating once the total variance exceeds some fixed threshold K. The votes generated (but not necessarily written) so far are called the *core votes*. The role of the core votes is to produce a large majority in favor of one value or the other.

3. Depending on the mechanism used to detect termination, there may be additional divergence between processes due to votes generated by processes that have not yet detected crossing the threshold, as well as votes that are missed because the write operations recording them are delayed by the adversary.

For example, in the Bracha-Rachman algorithm, $w_{i,t} = 1$ for all i and t and $K = \Theta(n^2)$. This gives a standard deviation of the core votes of $\Theta(n)$. The divergence among processes is bounded by having each process collect all n registers every $O(n/\log n)$ steps. It follows that the maximum divergence observed by any process is small relative to the core votes with at least constant probability. The total work of the Bracha-Rachman algorithm is $O(n^2 \log n)$, with the extra $\log n$ factor coming from the need to perform a $\Theta(n)$-work collect operation every $\Theta(n/\log n)$ steps. The individual work is the same as the total work in the worst case: a single process can be forced to generate all $\Theta(n^2)$ core votes.

The recent algorithm of Attiya and Censor [6] improves on the Bracha-Rachman algorithm by adding a single multi-writer bit that allows any process that detects termination to signal this to all other processes immediately. This reduces the divergence between processes since all check the termination bit at each step and thus see almost the same set of votes. In consequence, it suffices for the additional votes to have variance $O(n^2)$, since we can now tolerate a constant probability that the divergence exceeds the magnitude of the core votes. This small but significant change means that it is only necessary to check for termination every $\Theta(n)$ steps. The resulting total work is thus $\Theta(n^2)$; this bound is optimal because it matches the tight lower bound on randomized consensus proved in the same paper. As in the Bracha-Rachman algorithm, a single process may be forced to generate most of the votes, so the individual work is also $\Theta(n^2)$.

To reduce the individual work, Aspnes and Waarts [5] modified the Bracha-Rachman algorithm by having processes cast increasingly heavy votes over time. In the Aspnes-Waarts algorithm, $w_{i,t} = t^a$ where $a = \frac{\log n - 1}{2}$; the odd-looking exponent is chosen so that the total variance contributed by a process's first m votes is $\Theta\left(m^{\log n}\right)$. Termination is tested by doing an n-read collect operation every $\Theta(n/\log n)$ steps, as in the Bracha-Rachman algorithm. A careful analysis of the rate of increase of the weights shows that the increased weights late in the algorithm still allow the core votes to dominate with constant probability; the

intuition is that not many of the processes can be casting very large votes (since otherwise the algorithm would have terminated sooner). However, because the size of votes continue to increase after crossing the threshold, it is necessary to increase the variance of the core votes to compensate for possibly larger divergence; this adds an extra log factor to the total work bound of Bracha-Rachman, raising it to $O(n^2 \log^2 n)$. The payoff is that the individual work is now bounded by only $O(n \log^2 n)$, since a fast process running alone quickly generates enough variance on its own to reach the core-vote variance threshold.

3. A SHARED-COIN ALGORITHM WITH $O(n \log n)$ INDIVIDUAL WORK

We combine the weighted votes mechanism of Aspnes-Waarts with the termination bit of Attiya-Censor, in order to reduce divergence and hence, individual work. The reduced divergence also simplifies the proof. Our shared-coin algorithm provides a constant agreement parameter and requires $O(n \log n)$ individual work.

In the algorithm, each process generates votes, whose variance and sums are recorded in an array of n single-writer multi-reader registers.

The processes generate votes until the variance of votes reaches a certain threshold, which is small enough to guarantee that not too many steps are taken, but large enough to give a good probability for the votes to have a distinct majority.

Each process reads the array and decides on the majority of the votes it reads. In order to agree on the same majority value, processes should read similar sets of votes; this is achieved by bounding the total number of votes that are generated (by any process) after some process observes that the threshold was exceeded.

A very simple way to guarantee this property is to have processes frequently read the array in order to quickly detect that the threshold was reached. This, however, increases the individual and total work. Instead, we use a multi-writer multi-reader bit, called *done*, that serves as a binary termination flag; it is initialized to false. A process that detects that enough votes were generated, sets *done* to true. This allows a process to read the array only once in every $O(n)$ of its votes, but check the register *done* before each vote.

Increasing weights for the votes as in [5] are used to reduce the individual work, i.e., if not many processes take steps, then they generate votes with increasing weights so fewer coins are needed to reach the threshold. The threshold is now based on the variance of the votes, rather than on their number.

The pseudocode appears in Algorithm 1. In addition to the binary register *done*, it uses an array V of n single-writer multi-reader registers, each with the following components (all initially 0):

***variance*:** the total variance of the votes generated by the process so far.

***sum*:** the weighted sum of votes so far.

Each process keeps a local copy v of the array V. The collect in lines 6 and 8 is an abbreviation for n read operations of the array V.

We take the weight function to be

$$w(t) = t^a,$$

Algorithm 1 Shared coin algorithm: code for process p_i.

```
local integer t, initially 0
array v[1..n]
1: while not done do
2:     t++
3:     vote = random(−1,+1) · w(t)        // a fair local coin
4:     V[i].⟨variance, sum⟩=
           ⟨V[i].variance + vote², V[i].sum + vote⟩
                                          // atomically
5:     if t = 0 mod c then          // check if time to terminate
6:         v = collect V                    // n read operations
7:         if v[1].variance + ... + v[n].variance > K then
               done = true                 // raise termination flag
   end while
8: v=collect V                              // n read operations
9: return sign(∑_{j=1}^{n} v[j].sum)        // return +1 or −1,
           // depending on the majority value of the votes
```

where $a = \frac{1}{2}(\log n - 1)$, and the threshold to be

$$K = (64n \log n)^{\log n} \frac{n}{\log n} .$$

The collect operation is performed every $c = n - 3$ local votes.

For convenience, we will also denote $A = 2a + 1 = \log n$, and define $T_K = \left(\frac{AK}{n}\right)^{1/A}$; this represents the maximum number of votes that a process generates until the total variance first exceeds K, if the execution is synchronous. The choice of parameters implies that

$$T_K = \left(\frac{\log n (64n \log n)^{\log n} \cdot n/\log n}{n}\right)^{1/\log n} = 64n \log n.$$

Finally, we define $g = 1 + \frac{c+3}{T_K} = 1 + \frac{1}{64 \log n}$, and $\Delta = n(gT_K)^a$.

The following lemma is used to prove that certain functions are concave.

LEMMA 1. *Let $f_{p,q,r,s}(x) = ((rx)^p + s)^q$, where p, q, r, s, and x are all non-negative, $pq \leq 1$ and $p \leq 1$. Then $f_{p,q,r,s}(x)$ is concave.*

PROOF. To show that $f_{p,q,r,s}(x)$ is concave, we show that its second derivative is non-positive.

$$\begin{aligned}
\frac{d^2}{dx^2}((rx)^p + s)^q &= \frac{d}{dx} q((rx)^p + s)^{q-1} p(rx)^{p-1} r \\
&= q(q-1)((rx)^p + s)^{q-2} p^2 (rx)^{2p-2} r^2 \\
&+ q((rx)^p + s)^{q-1} p(p-1)(rx)^{p-2} r^2 \\
&= [q((rx)^p + s)^{q-2} p(rx)^{p-2} r^2] \\
&\cdot [(q-1)p(rx)^p + ((rx)^p + s)(p-1)] \\
&= [q((rx)^p + s)^{q-2} p(rx)^{p-2} r^2] \\
&\cdot [(rx)^p (pq - 1) + s(p-1)]
\end{aligned}$$

If p, q, r, s and x are all non-negative, then the first term above is non-negative. If furthermore, $pq \leq 1$, and $p \leq 1$, then the second term above is non-positive. In this case the second derivative is non-positive and hence the function $f_{p,q,r,s}(x) = ((rx)^p + s)^q$ is concave. □

We denote by U_i, $1 \leq i \leq n$, the random variable describing the maximum number of votes that process p_i generates

during an execution. We will later use Lemma 1 in order to apply the next lemma:

LEMMA 2 (LEMMA 5.4 FROM [5]). *Let $\psi(x) = x^A/A$ and let χ be any strictly increasing function such that $\chi(\psi^{-1}(x) + c + 1)$ is concave in x. Then, $\sum_{i=1}^{n} \chi(U_i) \leq n\chi(T_K + c + 1)$.*

In order to bound the individual work and the agreement parameter, we fix an execution α of the algorithm. First, we bound the individual work, in a manner similar to Theorem 5.10 from [5].

LEMMA 3. *Every process executes at most $O(n \log n)$ steps during the execution.*

PROOF. Consider some process p_i. After $(AK)^{1/A}$ votes of p_i, the total variance of its votes is:

$$\sum_{x=1}^{(AK)^{1/A}} x^{2a} > \int_{0}^{(AK)^{1/A}} x^{2a} dx = \frac{((AK)^{1/A})^A}{A} = K.$$

This implies that after at most c additional votes, p_i performs the collect of line 6 and notices that the total variance has exceeded K. Therefore p_i generates at most $(AK)^{1/A}+c$ votes. Each vote costs 1 write operation in line 4, and one read operation in line 1, and every c votes cost n read operations in line 6. In addition there may be one write operation in line 7, and there are n read operations in line 8. Therefore the total number of operations that process p_i performs is at most

$$((AK)^{1/A}+c)\left(1+\left\lceil\frac{n}{c}\right\rceil\right)+1+n \leq (AK)^{1/A}\left(2+\frac{n}{c}\right)+2c+3n.$$

Substituting the parameters, we have that the number of operations that any process executes is at most:

$$(AK)^{1/A}\left(2+\tfrac{n}{c}\right)+2c+3n =$$

$$\left(\log n(64n\log n)^{\log n}\frac{n}{\log n}\right)^{\frac{1}{\log n}}(2+\frac{n}{n-3})+2(n-3)+3n =$$

$$\left(n^{\frac{1}{\log n}}\cdot 64n\log n\right)(2+\frac{n}{n-3})+2(n-3)+3n =$$

$$(2\cdot 64n\log n)\,O(1)+O(n),$$

which is $O(n \log n)$. □

We now show that all processes that terminate agree on the value 1 with constant probability; by symmetry, the same holds for -1, implying that the algorithm has a constant agreement parameter.

We model the execution α as a stochastic process, by considering the votes that are generated during the execution as a sequence of random variables $X_1, X_2, \cdots$. The value of X_i represents the i-th vote that is generated by some process in line 3, or 0 if less than i votes occur during the execution α.

In order to prove a constant agreement parameter, we need to bound the partial sums of votes during different times in the execution. Similar to [6], we partition the execution into three phases (see Figure 1). The first phase ends when a total variance of K is generated. The second phase ends when the termination bit *done* is set for the first time. In the third phase, each process collects the array in line 8, and returns a value for the shared coin.

For a set of votes F, we let $Sum(F)$ be the sum of the votes in F. Denote by F_C the set of votes that are generated by the first time that the termination bit *done* is set. We further denote by F_i the set of votes read by the collect of process p_i in line 8. This is the set according to which the process p_i decides on its output, i.e., p_i returns $\text{sign}(Sum(F_i))$. Since each process generates at most one more vote after the termination bit *done* is set, there can be no more than n additional votes in F_i that are not in F_C. This allows us to bound the sum of these additional votes (Lemma 4).

Since F_C is the set of votes generated when *done* is set, then it is exactly the set of votes generated in the first and second phases. Let F_{first} be the first votes that are generated until the total variance reaches K, and $F_{second} = F_C \setminus F_{first}$ (see Figure 1). This implies that $Sum(F_C) = Sum(F_{first}) + Sum(F_{second})$. We later bound $Sum(F_C)$ by bounding $Sum(F_{first})$ (Lemma 5) and $Sum(F_{second})$ (Lemma 7).

We begin with the next lemma, which bounds the sum of additional votes that a process p_i may observe.

LEMMA 4. *For every i, $1 \leq i \leq n$,*

$$|Sum(F_i) - Sum(F_C)| \leq \Delta.$$

PROOF. Let X_j be a vote in F_C, i.e., X_j is generated by some process p_k before the termination bit *done* is set for the first time.

If the vote X_j is not included in F_i, then it is not written to the register of process p_k before it is read during the collect of p_i in line 8, and therefore no other vote of p_k is generated yet, hence there can be no other vote $X_{j'}$ by process p_k that is in F_C but not in F_i.

Recall that the process p_k generates at most U_k votes, and hence the vote X_j has weight of at most U_k^a, and therefore

$$|Sum(F_i) - Sum(F_C)| \leq \sum_{\ell=1}^{n} U_\ell^a.$$

Let $\chi(y) = y^a$ and $\psi(x) = \frac{x^A}{A}$, then we have

$$\chi(\psi^{-1}(x) + c + 1) = ((Ax)^{1/A} + c + 1)^a.$$

Using Lemma 1 with $p = (1/A), q = a, r = A$ and $s = c + 1$, we have that this function is concave in x, since all the parameters are non-negative, $pq = \frac{1/2(\log n - 1)}{\log n} \leq 1$, and $p = \frac{1}{\log n} \leq 1$. Therefore, by Lemma 2 we have that

$$\sum_{\ell=1}^{n} U_\ell^a \leq n(T_K + c + 1)^a,$$

and hence

$$\begin{aligned} |Sum(F_i) - Sum(F_C)| &\leq \sum_{\ell=1}^{n} U_\ell^a \leq n(T_K + c + 1)^a \\ &\leq n(gT_K)^a = \Delta. \end{aligned}$$

□

Having bounded the sum of coins of the third phase by Δ, we now show that there is at least a constant probability

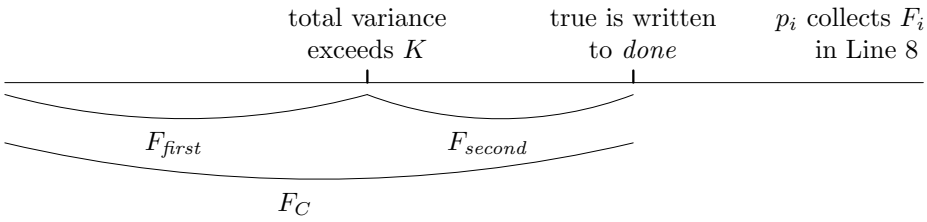

Figure 1: Phases of the shared-coin algorithm.

that $Sum(F_C) \geq \Delta$. In this case, by Lemma 4 all processes that terminate have $Sum(F_i) > 0$, and therefore agree on the value 1.

Recall that $Sum(F_C) = Sum(F_{first}) + Sum(F_{second})$. Therefore, we can provide a lower bound the probability that $Sum(F_C) \geq \Delta$ by bounding from above the probabilities that $Sum(F_{first}) \leq \sqrt{K}$ or $Sum(F_{second}) \leq \Delta - \sqrt{K}$.

Lemma 5.6 from [5] gives a bound on the probability that the sum of the votes of the first phase is too small. Our parameters satisfy the conditions of this lemma, and it remains the same:

LEMMA 5 (LEMMA 5.6 FROM [5]). *If*

$$\frac{4A^2}{n^{1/A}T_K} \leq 1,$$

then

$$Pr\left[Sum(F_{first}) \leq \sqrt{K}\right] \leq \Phi(1) + C_1 \cdot \left(\frac{A^2}{n^{1/A}T_K}\right)^{1/5},$$

where C_1 is a constant, and $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{1}{2}y^2}dy$ is the normal distribution function.

In our case, the condition of Lemma 5.6 from [5] is satisfied since:

$$\frac{4A^2}{n^{1/A}T_K} = \frac{4\log^2 n}{n^{1/\log n}64n\log n} = \frac{\log n}{32n} \leq 1,$$

and therefore

$$\begin{aligned}\Pr\left[Sum(F_{first}) \leq \sqrt{K}\right] &\leq \Phi(1) + C_1 \cdot \left(\frac{A^2}{n^{1/A}T_K}\right)^{1/5} \\ &= \Phi(1) + C_1 \cdot \left(\frac{\log n}{128n}\right)^{1/5}.\end{aligned}$$

As n grows, this probability tends to $\Phi(1) = 0.841$, and therefore is at most 0.842 for large enough n.

Lemma 5.7 from [5] bounds the probability that the sum of the votes of the second phase is too small.

LEMMA 6 (LEMMA 5.7 FROM [5]). *If*

$$g^a \leq \frac{1}{2}\sqrt{\frac{T_K}{nA}},$$

and

$$g^A \leq 1 + \frac{1}{8\log(10n)},$$

then

$$Pr\left[Sum(F_{second}) \leq \Delta - \sqrt{K}\right] \leq \frac{1}{10n}.$$

This lemma has to be modified in order to handle the fact that there can be more votes in the second phase than there are in [5] since c is larger and therefore the array is scanned less frequently. The first condition of Lemma 5.7 is $g^a \leq \frac{1}{2}\sqrt{\frac{T_K}{nA}}$, and is still satisfied by the parameters since:

$$g^a = \left(1 + \frac{1}{64\log n}\right)^{\frac{1}{2}(\log n - 1)} \leq e^{\frac{1/2(\log n-1)}{64\log n}} \leq e^{1/128},$$

and

$$\frac{1}{2}\sqrt{\frac{T_K}{nA}} = \frac{1}{2}\sqrt{\frac{64n\log n}{n\log n}} = 4.$$

The second condition of Lemma 5.7 is $g^A \leq 1 + \frac{1}{8\log(10n)}$, and is not satisfied since:

$$g^A = \left(1 + \frac{c+3}{T_K}\right)^{\log n} = \left(1 + \frac{1}{64\log n}\right)^{\log n},$$

which is not smaller than $1 + \frac{1}{8\log(10n)}$, for large n.

However, once the termination bit is set, every process notices that the threshold has been reached in at most one more step. This allows the bound on the probability $\Pr\left[Sum(F_{second}) \leq \Delta - \sqrt{K}\right]$ in this lemma to be constant and not $O(1/n)$. We therefore prove a new lemma which only uses the fact that g^A is bounded by a constant, and the statement is weaker but sufficient because of the termination bit.

For every i, we define a random variable Y_i, which is equal to X_i if X_i is a vote in F_{second}, and 0 otherwise. By definition we have that $Sum(F_{second}) = \sum_{j=1}^{\infty} Y_j$, which allows us to prove the lemma that bounds the sum of the votes of the second phase.

LEMMA 7.

$$Pr\left[Sum(F_{second}) \leq \Delta - \sqrt{K}\right] \leq 4(e^{\frac{1}{64}} - 1) \leq 0.063.$$

PROOF. Define $S'_i = \sum_{j=1}^{i} Y_j$. In order to bound the sums S'_i, we first bound the maximum index of Y_i, as follows. The maximum index of X_i that can be generated is $\sum_{i=1}^{n} U_i$, which is at most $M = n(T_K + c + 1)$ by Lemma 2, when using the identity function for χ. Specifically, M is the maximum index of X_i that can be in F_C, and therefore $Sum(F_{second}) = S'_M$, which is the sum of the variables Y_i (i.e., for any index $i \geq M$, we have $Y_i = 0$).

Our proof applies Chebyshev's Inequality[1] to S'_M, hence we need to bound its variance. The random variables Y_i

[1]The sums S'_i define a martingale, for which we can apply an Azuma-type inequality [5] to get a slightly better constant. We use Chebyshev's Inequality in order to simplify the proof.

are not independent, since the adversary has some control over the weight of votes by choosing the schedule. However, they are uncorrelated because the expected value of Y_i is 0, even conditioned over values of other votes. Formally, for every $i \neq j$ we have $E[Y_i \cdot Y_j] = E[E[Y_i \cdot Y_j | Y_j]] = 0$. Since $E[Y_i] = E[Y_j] = 0$ this implies that

$$Cov[Y_i, Y_j] = E[(Y_i - E[Y_i])(Y_j - E[Y_j])] = E[Y_i \cdot Y_j] = 0,$$

and therefore we have

$$Var[S'_M] = Var\left[\sum_{i=1}^{M} Y_i\right] = \sum_{i=1}^{M} Var[Y_i].$$

Since $Var[Y_i] = Y_i^2$, in order to bound the variance of S'_M we show that $\sum_{i=1}^{M} Y_i^2 \leq K(e^{1/64} - 1)$. Notice that

$$\sum_{X_i \in F_{first}} X_i^2 \geq K - t^{2a}$$

for some t which is no more than U_i for some process p_i, otherwise adding the next vote would still keep the variance below the threshold and therefore should also be in F_{first}. Therefore we have

$$\begin{aligned}
\sum_{i=1}^{M} Y_i^2 &= \sum_{X_i \in F_{second}} X_i^2 = \sum_{X_i \in F_C} X_i^2 - \sum_{X_i \in F_{first}} X_i^2 \\
&\leq \sum_{i=1}^{n} \sum_{j=1}^{U_i} j^{2a} - (K - t^{2a}) \\
&\leq \sum_{i=1}^{n} \sum_{j=1}^{U_i+1} j^{2a} - K \leq \sum_{i=1}^{n} \frac{(U_i+2)^A}{A} - K.
\end{aligned}$$

Let $\chi(y) = \frac{(y+2)^A}{A}$ and $\psi(x) = \frac{x^A}{A}$, then we have

$$\chi(\psi^{-1}(x) + c + 1) = \frac{((Ax)^{1/A} + c + 3)^A}{A}.$$

Using Lemma 1 with $p = (1/A), q = A, r = A$ and $s = c+3$, we have that $((Ax)^{1/A} + c + 3)^A$ is concave in x, since all the parameters are non-negative, $pq = 1$, and $p = \frac{1}{\log n} \leq 1$. Hence, $\chi(\psi^{-1}(x) + c + 1) = \frac{((Ax)^{1/A} + c + 3)^A}{A}$ is also concave in x. Therefore, by Lemma 2 we have that

$$\sum_{i=1}^{n} \frac{(U_i+2)^A}{A} \leq \frac{n(T_K + c + 3)^A}{A},$$

and hence

$$\begin{aligned}
\sum_{i=1}^{M} Y_i^2 &\leq \sum_{i=1}^{n} \frac{(U_i+2)^A}{A} - K \leq \frac{n(T_K + c + 3)^A}{A} - K \\
&= \frac{n(gT_K)^A}{A} - K = \frac{n(g((AK/n)^{1/A}))^A}{A} - K \\
&= K(g^A - 1) \leq K(e^{1/64} - 1),
\end{aligned}$$

where the last inequality holds since

$$g^A = (1 + \frac{c+3}{T_K})^{\log n} = (1 + \frac{1}{64 \log n})^{\log n} \leq e^{1/64}.$$

Applying Chebyshev's Inequality to S'_M gives

$$\begin{aligned}
\Pr\left[|S'_M| \geq \frac{\sqrt{K}}{2}\right] &\leq \frac{Var[S'_M]}{\left(\frac{\sqrt{K}}{2}\right)^2} \leq \frac{K(e^{1/64} - 1)}{\frac{K}{4}} \\
&= 4(e^{1/64} - 1) \leq 0.063.
\end{aligned}$$

Finally, in order to get the claimed probability, we bound Δ in terms of K. Notice that

$$\frac{1}{2}\sqrt{\frac{T_K}{nA}} = \frac{1}{2}\sqrt{\frac{64n \log n}{n \log n}} = 4$$

and hence $g^a \leq g^A \leq e^{1/64} \leq \frac{1}{2}\sqrt{\frac{T_K}{nA}}$. Therefore,

$$\begin{aligned}
\Delta &= n(gT_K)^a \leq n\left(\frac{1}{2}\sqrt{\frac{T_K}{nA}}\right) T_K^{\frac{1}{2}(\log n - 1)} \\
&= \frac{n}{2}\frac{1}{\sqrt{nA}} T_K^{\frac{1}{2}\log n} = \frac{n}{2}\frac{1}{\sqrt{nA}}\sqrt{\frac{AK}{n}} = \frac{\sqrt{K}}{2},
\end{aligned}$$

which implies that

$$\Pr\left[Sum(F_{second}) \leq \Delta - \sqrt{K}\right] \leq \Pr\left[S'_M \leq -\frac{\sqrt{K}}{2}\right]$$

$$\leq \Pr\left[|S'_M| \geq \frac{\sqrt{K}}{2}\right] \leq 0.063.$$

□

We can now calculate the agreement parameter.

LEMMA 8. *Algorithm 1 is a shared-coin algorithm with a constant agreement parameter $\delta = 0.095$.*

PROOF. We show that the probability that all processes that terminate decide upon 1 is at least $\delta = 0.095$. The result for -1 follows by symmetry.

By Lemmas 5 and 7 we have

$$\begin{aligned}
&\Pr\left[\left(Sum(F_{first}) \leq \sqrt{K}\right) \vee \left(Sum(F_{second}) \leq \Delta - \sqrt{K}\right)\right] \\
&\leq \Pr\left[\left(Sum(F_{first}) \leq \sqrt{K}\right)\right] \\
&\quad + \Pr\left[\left(Sum(F_{second}) \leq \Delta - \sqrt{K}\right)\right] \\
&\leq \Phi(1) + C_1 \cdot \left(\frac{A^2}{n^{1/A} T_K}\right)^{1/5} + 0.063 \\
&\leq 0.842 + 0.063 = 0.905.
\end{aligned}$$

Therefore,

$$\Pr\left[\left(Sum(F_{first}) > \sqrt{K}\right) \wedge \left(Sum(F_{second}) > \Delta - \sqrt{K}\right)\right]$$

$$\geq 1 - 0.905 = 0.095.$$

If this event occurs, then

$$\begin{aligned}
Sum(F_C) &= Sum(F_{first}) + Sum(F_{second}) \\
&> \Delta - \sqrt{K} + \sqrt{K} = \Delta.
\end{aligned}$$

By Lemma 4 we have that $Sum(F_i) \geq Sum(F_C) - \Delta$, and therefore $Sum(F_i) > 0$. Hence with probability at least 0.095, all the processes that terminate will decide on the value 1. □

Lemmas 3 and 8 complete the proof of the algorithm, and we have the main theorem.

THEOREM 9. *Algorithm 1 is a shared-coin algorithm with constant agreement parameter and $O(n \log n)$ individual work.*

4. INTERLEAVING SHARED-COIN ALGORITHMS

Our goal in this section is to obtain a shared-coin algorithm that has both $O(n^2)$ total work and $O(n \log n)$ individual work. We do this by *interleaving* the algorithm from [6] and Algorithm 1 (from Section 3).

Interleaving two algorithms A and B is done by performing a loop in which the process executes one step of each algorithm. When one of the algorithms terminates, returning a value v, the interleaved algorithm terminates as well, returning the same value v.

Let A and B be two shared-coin algorithms. We denote by $T_A(n)$ and $I_A(n)$ the total and individual work, respectively, of algorithm A, and its agreement parameter by δ_A. Similarly, we denote $T_B(n)$, $I_B(n)$ and δ_B for algorithm B.

We first argue that the total and individual step complexities of the interleaved algorithm are the minimum between the respective complexities of algorithms A and B.

LEMMA 10. *The interleaved algorithm of algorithms A and B, has an expected total work of $2\min\{T_A(n), T_B(n)\} + n$, and an expected individual work of $2\min\{I_A(n), I_B(n)\} + 1$.*

PROOF. We begin by proving the total work. After at most $2T_A(n)+n$ total steps where executed by the adversary, at least $T_A(n)$ of them where in algorithm A, and hence all the processes have terminated Algorithm A, and have therefore terminated the interleaved algorithm. The same applies to Algorithm B. Therefore the interleaved algorithm has a total work of $2\min\{T_A(n), T_B(n)\} + n$.

We now prove the bound on the individual work. Consider any process p_i. After at most $2I_A(n) + 1$ total steps of p_i where executed by the adversary, at least $I_A(n)$ of them were in algorithm A, and hence the process p_i has terminated Algorithm A, and has therefore terminated the interleaved algorithm. The same applies to Algorithm B. This is true for all the processes, therefore the interleaved algorithm has an individual work of $2\min\{T_A(n), T_B(n)\} + 1$. □

The next lemma shows that the agreement parameter of the interleaved algorithm is the product of the agreement parameters of algorithms A and B. The idea behind the proof is that since different processes may choose a value for the shared coin based on any of the two algorithms, for all process to agree on some value v we need all processes agreeing on v in both algorithms. In order to deduce an agreement parameter which is the product of the two given agreement parameters, we need to show that the executions of the two algorithms are independent, in the sense that the adversary cannot gain any additional power out of running two interleaved algorithms.

In general, it is not obvious that the agreement parameter of the interleaved algorithm is the product of the two given agreement parameters. In each of the two algorithms it is only promised that there is a constant probability that the adversary cannot prevent a certain outcome, but in the interleaved case the adversary does not have to decide in advance which outcome it tries to prevent from a certain algorithm, since it may depend on how the other algorithm proceeds.

Notice that the lemma assumes that the algorithms always terminate within some fixed bound on the number of steps, and not only with probability 1, which is indeed the case in Algorithm 1 and the algorithm from [6]. This assumption is needed since there are cases that do not satisfy it in which such a claim does not hold (see [12]).

LEMMA 11. *If both algorithms A and B always terminate within some fixed bound on the number of steps, then the interleaving of algorithms A and B has agreement parameter $\delta_A \cdot \delta_B$.*

PROOF. Since the algorithms always terminate within some fixed bound on the number of steps, we define the probability of reaching agreement on the value v for every configuration C in one of the algorithms, by backwards induction, as follows.

In the configurations we consider, all the processes have flipped their local coins and are now pending to access the shared memory.

With every configuration C, we associate a value s that is the maximal number of steps taken by all the processes from configuration C, over all possible adversaries and all results of the local coin flips. Since the algorithms always terminate within some fixed bound on the number of steps, s is well defined.

Consider algorithm A. For clarity, we denote configurations for algorithm A with a subindex A. For a configuration C_A we define the probability $\Pr_v^A[C_A]$ for agreeing on the value v in algorithm A by induction on s, as follows. In a configuration C_A for which $s = 0$, all processes terminate in the algorithm. We define $\Pr_v^A[C_A]$ to be 1 if all the processes agree on the value v, and 0 otherwise. Let C_A be any other configuration, then:

$$\Pr_v^A[C_A] = \min_{p_i} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v^A[(C_A, p_i, y)],$$

where (C_A, p_i, y) is the resulting configuration after p_i takes one step including coin-flips (y is the random variable representing the results of the local coin flips, in the probability space X^i). We define $\Pr_v^B[C_B]$ similarly for algorithm B.

We now consider the interleaved algorithm. Each configuration C consists of the local states of all the processes, and the values of the shared registers. We denote by $C|_A$ the projection of the configuration C on algorithm A, i.e., $C|_A$ consists of the local states of all the processes regarding algorithm A, and the values of the shared registers of algorithm A. Similarly we denote by $C|_B$ the projection of C on algorithm B, and therefore use the notation $C = (C|_A, C|_B)$ to describe the configuration C.

We define probabilities for agreeing in the interleaved algorithm as follows. We extend the definitions of agreeing in Algorithm A and Algorithm B by defining $\Pr_v^A[C] = \Pr_v^A[C|_A]$, and $\Pr_v^B[C] = \Pr_v^B[C|_B]$. We now define the probability $\Pr_v[C]$ for agreeing on the value v in both algorithms in the interleaved algorithm by induction on s, as follows.

In a configuration C for which $s = 0$, all processes terminate in both algorithms A and B. We define $\Pr_v[C]$ to be 1 if all the processes decide v in both algorithms, and 0 otherwise. Let C be any other configuration, then:

$$\Pr_v[C] = \min_{p_i} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v[(C, p_i, y)],$$

where (C, p_i, y) is the resulting configuration after p_i took one step and its coin has been flipped. Denote

$$C' = (C, p_i, y) = (C'|_A, C'|_B),$$

and notice that if p_i took a step in algorithm A, then $C|_B = C'|_B$, and if p_i took a step in algorithm B, then $C|_A = C'|_A$.

We now claim that for every configuration C, $\Pr_v[C] = \Pr_v^A[C] \cdot \Pr_v^B[C]$; the proof is by induction on s.

Base case: If $s = 0$, then all processes have terminated in both algorithms A and B. Processes agree on v if and only if they agree on v in both algorithms, that is, $\Pr_v[C] = \Pr_v^A[C] \cdot \Pr_v^B[C]$.

Induction step: Assume the claim holds for any configuration C' with at most $s-1$ steps remaining to termination under any adversary. Let C be a configuration with at most s steps until termination under any adversary. For every i, $1 \leq i \leq n$, let C^i be a random variable representing the configuration reached from C after process p_i takes a step, including flipping its local coins $y \in X^i$. By definition of $\Pr_v[C]$, we have:

$$\Pr_v[C] = \min_{p_i} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v[C^i]$$

$$= \min \left\{ \min_{p_i \in A} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v[C^i], \min_{p_i \in B} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v[C^i] \right\}$$

where $p_i \in A$ and $p_i \in B$ are abbreviations for a process whose next step is taken in algorithm A or B, respectively.

By the induction hypothesis on the configuration C^i, with one less step to termination, we get:

$$\Pr_v[C] = \min \left\{ \min_{p_i \in A} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v^A[C^i] \cdot \Pr_v^B[C^i], \min_{p_i \in B} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v^A[C^i] \cdot \Pr_v^B[C^i] \right\}$$

$$= \min \left\{ \min_{p_i \in A} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v^A[C^i|_A] \cdot \Pr_v^B[C^i|_B], \min_{p_i \in B} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v^A[C^i|_A] \cdot \Pr_v^B[C^i|_B] \right\}$$

where the second equality is by definition of $\Pr_v^A[C]$ and $\Pr_v^B[C]$. If the step taken from C by p_i is in algorithm A, then $C^i|_B = C|_B$, and if the step taken from C by p_i is in algorithm B, then $C^i|_A = C|_A$. Thus,

$$\Pr_v[C] = \min \left\{ \min_{p_i \in A} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v^A[C^i|_A] \cdot \Pr_v^B[C|_B], \min_{p_i \in B} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v^A[C|_A] \cdot \Pr_v^B[C^i|_B] \right\}$$

$$= \min \left\{ \Pr_v^B[C|_B] \left(\min_{p_i \in A} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v^A[C^i|_A] \right), \Pr_v^A[C|_A] \left(\min_{p_i \in B} \sum_{y \in X^i} \Pr[y] \cdot \Pr_v^B[C^i|_B] \right) \right\}$$

$$= \min \left\{ \Pr_v^B[C] \cdot \Pr_v^A[C], \Pr_v^A[C] \cdot \Pr_v^B[C] \right\}$$

$$= \Pr_v^A[C] \cdot \Pr_v^B[C],$$

which completes the proof of the claim that $\Pr_v[C] = \Pr_v^A[C] \cdot \Pr_v^B[C]$.

The lemma follows by applying the claim to the initial configuration, where $\Pr_v^A[C] = \delta_A$ and $\Pr_v^B[C] = \delta_B$. Hence, $\delta = \Pr_v[C] = \Pr_v^A[C] \cdot \Pr_v^B[C] = \delta_A \cdot \delta_B$, for every $v \in \{0, 1\}$, which completes the proof. □

By Lemmas 10 and 11, interleaving Algorithm 1 and the algorithm from [6], gives the following.

THEOREM 12. *There is a shared-coin algorithm with a constant agreement parameter, with $O(n^2)$ total work and $O(n \log n)$ individual work.*

5. SUMMARY

We presented a shared-coin protocol with $O(n \log n)$ individual work and $O(n^2)$ total work; this implies a randomized consensus protocol with the same complexities. It is an intriguing open question whether an algorithm with linear individual work can be designed.

Our shared-coin protocol uses multi-writer registers, while the $O(n \log^2 n)$ individual-work protocol of Aspnes and Waarts [5] uses only single-writer registers. This is because the absence of the multi-writer termination bit allows larger drifts between the sets of votes that different processes observe. As in the case of the total work, the question of whether it is possible to obtain a better bound for single-writer registers remains open.

Acknowledgements. The authors would like to thank Dana Angluin, David Eisenstat and Roberto Segala for useful discussions.

6. REFERENCES

[1] K. 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] J. Aspnes. Time- and space-efficient randomized consensus. *J. Algorithms*, 14(3):414–431, 1993.

[3] J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–176, Sept. 2003.

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

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

[6] H. Attiya and K. Censor. Tight bounds for asynchronous randomized consensus. In *Proceedings of the 39th annual ACM symposium on Theory of computing (STOC)*, pages 155–164, 2007.

[7] G. Bracha and O. Rachman. Randomized consensus in expected $O(n^2 \log n)$ operations. In *Proceedings of the 5th International Workshop on Distributed Algorithms (WDAG)*, pages 143–150, 1991.

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

[9] M. Herlihy. Wait-free synchronization. *ACM Trans. Prog. Lang. Syst.*, 13(1):124–149, January 1991.

[10] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

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

[12] N. A. Lynch, R. Segala, and F. W. Vaandrager. Observing branching structure through probabilistic contexts. *SIAM J. Comput.*, 37(4):977–1013, 2007.

[13] M. Saks, N. Shavit, and H. 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.

[14] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, Dec. 1990.