Time- and Space-Efficient Randomized Consensus

James Aspnes

School of Computer Science

Carnegie-Mellon University

Pittsburgh, PA 15213

February 10, 1992

Abstract

A protocol is presented which solves the *randomized consensus problem*[9] for shared memory. The protocol uses a total of $O(p^2+n)$ worst-case expected increment, decrement and read operations on a set of three shared $O(\log n)$-bit counters, where p is the number of active processors and n is the total number of processors. It requires less space than previous polynomial-time consensus protocols[6, 7], and is faster when not all of the processors participate in the protocol. A modified version of the protocol yields a *weak shared coin* whose bias is guaranteed to be in the range $1/2 \pm \epsilon$ regardless of scheduler behavior, and which is the first such protocol for the shared-memory model to guarantee that all processors agree on the outcome of the coin.

1. Introduction

Consensus is a decision problem in which n processors, each starting with a value (0 or 1) not known to the others, must collectively agree on a single value. A *consensus protocol* is a distributed protocol for solving consensus. It is *correct* if it meets the following conditions:

- Consistency. All processors decide on the same value.
- Termination. Every processor decides on some value in finite expected time.
- Validity. If every processor starts with the same value, every processor decides on that value.

We are interested in consensus protocols in which the processors communicate through some shared concurrent data structure (for example, a collection of single-writer atomic registers). We would like our protocols to be *wait-free*: every processor terminates after a finite number of its *own* steps, regardless of other processors' halting failures or relative speeds[15]. Wait-free consensus is fundamental to synchronization without mutual exclusion. It can be used to obtain wait-free implementations of arbitrary abstract data types with atomic operations[16, 19, 15]. Consensus is also *complete* for *distributed decision tasks*[10] in the sense that it can be used to solve all such tasks that have a wait-free solution.

Consensus is often viewed as a game played between a set of processors and an adversarial scheduler, where the powers available to the scheduler are chosen to simulate the conditions under which the protocol is required to operate. It has been known for some time that the ability of the scheduler to stop even a single processor is sufficient to prevent consensus from being solved by a deterministic algorithm using only atomic registers[17, 14, 11, 9, 15]. Chor, Israeli, and Li[9]

showed that it is possible to solve consensus using globally-visible atomic coin flips as primitive actions. Since then a number of protocols have been put forward[1, 6, 7] which achieve consensus with only local coin-flip operations. These protocols work even against an adversary which has complete knowledge of the processors' programming and internal states, and total control over the scheduling of reads and writes.

The protocol of Aspnes and Herlihy[6] is of particular interest because it is the first such algorithm which runs in polynomial expected time. Unfortunately it makes extensive use of unbounded registers, a defect that is corrected by Attiya, Dolev, and Shavit[7], who describe the first known protocol that runs in both polynomial time and bounded space.

These protocols work on the same basic principle: the processors repeatedly execute a *weak shared coin* protocol until all are in agreement. The weak shared coin abstraction has the property that all processors that participate in it agree on its outcome with high probability, and the scheduler has only limited control over what that outcome will be. The Aspnes and Herlihy weak shared coin protocol works by having the processors collectively move a counter through a random walk, each repeatedly flipping a local coin to decide whether to move it down or up. If a processor notices that the counter has moved a predesignated distance from the origin, it returns with a value determined by the sign of the counter. The key to this protocol is that the scheduler's control is in effect limited to "withholding" local coin flips by stopping processors before they can write the local flips' results. Since the effect of any small number of local coin flips on the outcome of the protocol can be made arbitrarily small, the scheduler's power over the outcome can be made arbitrarily weak.

These weak shared coin protocols, however, are not *robust*: it is possible for the scheduler to

arrange that some processor decides 1 and then have others decide 0 with nonzero probability. Thus they cannot be used by themselves to solve consensus— instead, it is necessary to provide space for many successive shared coins and a round-based superstructure for detecting agreement and invalidating the outcome of older shared coins. The present work eliminates the need for multiple rounds by using a robust weak shared coin which guarantees that all processors agree on the outcome of the flip. Consensus is handled as a degenerate case of the weak shared coin with a preamble attached to ensure validity. The result is a simpler consensus protocol which uses less space, less time, and is more amenable to optimization than its predecessors.

The contributions of the paper are two-fold. First, it describes a weak shared coin protocol which uses both bounded space and is robust in the sense of guaranteeing agreement on its outcome, which may have applications as a source of *semi-random* bits[21]. Second, it describes the first polynomial-time consensus protocol whose running time depends primarily on the number of active processors, rather than the total number of processors. Thus the protocol is likely to be of use in situations where typically only a few of many processors may participate, such as when implementing a shared abstract data type that only a fraction of the processors in the system might access concurrently.

1.1. Model

We use the *probabilistic I/O automaton* model[6], a probabilistic extension of the *I/O automaton* model[18]. Details of the model are omitted here. Informally, the system is assumed to consist of n processors, each of which can execute as a step either a single operation of some shared primitive object (such as an atomic register or a counter with atomic increment, decrement, and read operations) or a local coin flip. The sequencing of the processors' actions is controlled by a

scheduler, a function which selects a processor to run for each possible state of the system. Implicit in the definition of the scheduler are its ability to exploit total knowledge of the system's state (including the internal state of the processors) and its inability to predict the outcome of future local coin flips.

It is not clear how best to measure the time performance of a wait-free algorithm. Previous work on wait-free consensus has considered only the total number of primitive operations used, without regard to how they are distributed among the processors or when they occur, an approach that is adopted in this paper. The primary virtues of this measure are both its simplicity and the fact that it adds no new timing assumptions to the wait-free model. Note that the total number of operations does depend on the choice of primitive object; in most of the paper it will be assumed that an increment, decrement, or read of an atomic counter counts as a single operation. Implementing these operations using the weaker primitive of single-writer atomic registers as described in Section 4 will generally increase the stated running times by a factor of $O(n^2)$.

2. Probabilistic Preliminaries

Let us begin by stating a few simple lemmas about the behavior of random walks.

Lemma 1 *Consider a symmetric random walk with step size* 1 *running between absorbing barriers at* a *and* b *and starting at* x, $a < x < b$. *Then:*

1. *The expected number of steps until one of the barriers is reached is given by* $(x-a)(b-x)$, *and is always less than or equal to* $\left(\frac{a-b}{2}\right)^2$.

2. The probability that the random walk hits b before a is $\frac{x-a}{b-a}$.

Lemma 2 *Consider a symmetric random walk with step size* 1 *running between a reflecting barrier at a and an absorbing barrier at b, starting at position x, $a < x < b$. Then the expected time until b is reached is $(x - (a - (b - a)))(b - x) \leq (b - a)^2$*

Proof: This random walk can be obtained from the random walk with absorbing barriers at b and $a - (b - a)$ by the transformation $x \mapsto a + |x - a|$. ∎

The following critical lemma describes a modified random walk that will be of great importance in analyzing the weak shared coin and consensus protocols:

Lemma 3 *Consider a symmetric random walk with absorbing barriers at a and b with the following twist: a point c, $a < c < b$ is designated as the* center *of the random walk. Before each step, an adversary chooses between moving randomly in either direction with probability* 1/2, *or moving away from c with probability* 1. *The adversary may also choose the starting position x of the random walk to be anywhere in the range from a to b. No matter what choices the adversary makes, The expected number of steps until one of the barriers is reached is less than or equal to $(b - a)^2$*

Proof: The game described can be thought of as a *controlled Markov process* in which the adversary is trying to maximize the expected time. As it is played over a finite set of states, it is a standard result of the theory of controlled Markov processors[13] that the maximum time can be achieved with a *simple* strategy, one which chooses the same option from each state at all times.

Such a strategy can be specified by listing the points where the adversary chooses to force the particle to move away from c. These points divide the random walk into distinct regions: the region containing c acts like a random walk with two absorbing barriers, and the others act like random walks with one absorbing barrier (on the side away from c) and one reflecting barrier (on the side toward c). Because each barrier can only be crossed away from c, once the particle leaves a region it can never return. Now, suppose the particle starts in a region with width w_1. After at most w_1^2 steps on average (by Lemmas 2 and 1) it will pass into a new region with width w_2; after an additional w_2^2 steps it will pass into a new region of width w_3, and so on until either a or b is reached. Since the regions all fit between a and b, $\sum w_i \leq b - a$, and thus (since each $w_i > 0$) $\sum w_i^2 \leq (b-a)^2$. ∎

Though the bound in Lemma 3 is proved in terms of a very powerful adversary that is always allowed to choose between a random move and a deterministic move at each step, the bound applies equally well to a weaker adversary whose choices are more constrained, as the stronger adversary could always choose to operate within the weaker adversary's constraints. This technique, of proving bounds for a strong adversary that then carry over to a weaker one, has great simplifying power. It will be used extensively in the analysis of the shared coin and consensus protocols.

3. The Weak Shared Coin Protocol

A *weak shared coin protocol* with bias ϵ is a distributed protocol in which n processors agree on a value (0 or 1) with high probability. A weak shared coin protocol is correct if it meets the following conditions:

```
Shared data:
  counter c with range [−K − 3n, K + 3n], initialized to 0

procedure SharedCoin()
repeat
  c := read(counter)
  if c ≤ −(K + n) then decide 0
  elseif c ≥ (K + n) then decide 1
  elseif c ≤ −K then decrement(counter)
  elseif c ≥ K then increment(counter)
  else
    if LocalCoin() = 0 then decrement(counter)
    else increment(counter)
    fi
  fi
end end
```

Figure 1: Robust weak shared coin protocol.

- Termination. Every processor decides on some value in finite expected time.

- Bounded bias. The probability that at least one processor decides on a given value is at most $1/2 + \epsilon$ where $0 \leq \epsilon < 1/2$.

A weak shared coin protocol is *robust* if, in addition, it satisfies the consistency condition:

- Consistency. All processors decide on the same value.

Figure 1 shows pseudocode for each processor's behavior in the robust weak shared coin protocol. The coin is constructed using an *atomic counter*, which supports atomic increment, decrement, and read operations. In this section, these operations are assumed to take unit time.

A processor's behavior in the protocol is represented in pictorial form in Figure 2. While a processor reads values in the central range from $-K$ to K (where K is a parameter of the protocol)

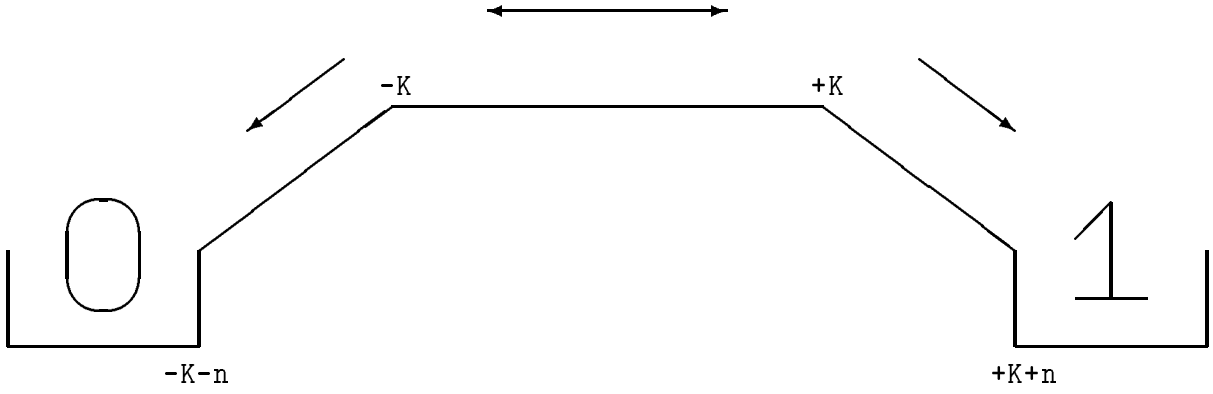

Figure 2: Pictorial representation of robust weak shared coin protocol.

it flips a local fair coin to decide whether to increment or decrement the counter. This part of the protocol is essentially the same as the random-walk-based weak shared coin of Aspnes and Herlihy[6]. What is new is the addition of a "slope" at either side of the random walk. On these slopes, a processor does not move the counter randomly but instead always moves it away from the center. When a processor reads a counter value in one of the "buckets" beyond the slopes, it decides either 0 or 1 depending on the sign of the counter.

If the slopes are wide enough, once any processor has seen a value that causes it to decide, all other processors will see values that cause them to push the counter toward the same decision. This mechanism eliminates the possibility that delayed writes might move the counter out of the decision range and allow the random walk (with small but non-negligible probability) to wander over to the other side. This fact is stated formally in the following lemma:

Lemma 4 *If any processor reads a counter value $v \geq (K+n)$, then all subsequent reads will return values $\geq K+1$; in the symmetric case where $v \leq -(K+n)$, all subsequent reads will be $\leq -(K+1)$.*

Proof: Suppose that a processor has read $v \geq (K + n)$; then it immediately terminates leaving $n - 1$ running processors. Thus the number d of processors that will execute a decrement before their next read is at most $n - 1$. Let $l = c - d$ where c is the value stored in the counter. Since $c \geq (K + n)$, it must be the case that $l \geq K + 1$. Now consider the effect of the actions the scheduler can take. If it allows a decrement to proceed, c and d both drop by 1 and l remains constant. If it allows an increment to occur, c increases and l increases with it. If it allows a read, the value read is $c \geq l \geq K + 1$, and thus d is unaffected. In each case l remains at least $K + 1$, and the claim follows since $c \geq l$. The proof of the symmetric case is similar. ∎

The consistency property follows immediately from Lemma 4. A similar argument shows that the counter will not overflow:

Lemma 5 *The counter value never leaves the range* $[K - 3n, K + 3n]$ *in any execution of the weak shared coin protocol.*

Proof: Suppose that the counter reaches $K + 2n$ at some point. Then each processor will execute at most one increment or decrement operation it reads the counter, at which point it will decide 1 and execute no additional operations. Thus the counter cannot exceed $K + 2n + n = K + 3n$. The full result follows by symmetry. ∎

Proving the termination and bounded bias properties of the shared coin requires some additional machinery. Define the *true position* t of the random walk to be the value in the counter, c, plus 1 for each processor that will increment the counter before its next read, and minus 1 for each processor that will decrement the counter before its next read. The following Lemma relates the value read by a processor to the true position of the random walk:

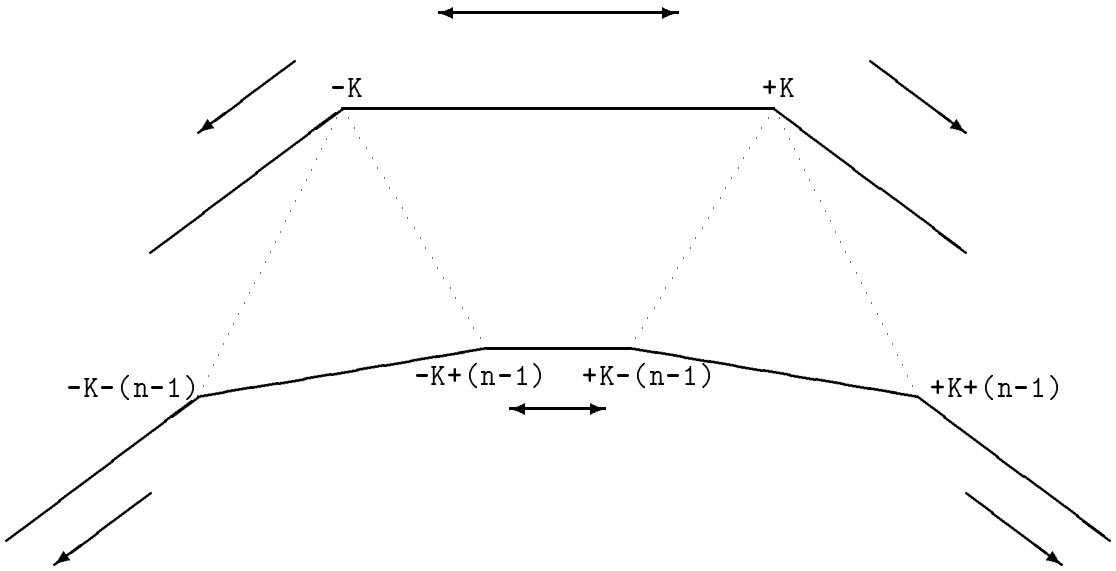

Figure 3: The protocol as a controlled random walk.

Lemma 6 *Let c be a value read from the counter by some processor and t the true position of the random walk in the state preceding the read. Then $|c - t| \leq n - 1$.*

Proof: There can be at most $n - 1$ processors with pending increments or decrements. ∎

Let us assume hereafter that the scheduler can cause a processor to read any value between $t-(n-1)$ and $t+(n-1)$; as such a scheduler is more powerful than any scheduler the protocol will actually face, any "good" statement we can prove with the assumption will carry over *a fortiori* to the situation without. The advantage of this simplifying assumption is that it allows us to forget about the vagaries of the counter value. Instead we can treat the protocol as a controlled random walk using the true position t.

Consider the lower part of Figure 3 (the upper part simply repeats Figure 2 without the buckets.) If the true position t is in the central region between $-K + (n-1)$ and $K - (n-1)$, then Lemma

6 says that any processor which reads the counter will see a value between $-K$ and K and move t randomly. In the two immediately adjacent regions a processor will either read a value between $-K$ and K, and move t randomly, or read a value that causes it to move t away from 0. Finally, any processor that reads a value in the outermost regions where $|t| > K + (n-1)$ will either make a decision or move t away from 0. In all of these cases, the scheduler is never allowed to force that true position toward 0; and if K is large relative to n much of the execution of the protocol will be spent in the central region where the scheduler's control is ineffective. These two properties of the protocol are the basis of the proof of its termination and bounded bias, as shown below.

Lemma 7 *The robust weak shared coin protocol executes an expected $O((K+n)^2)$ total counter operations when $K \geq n$.*

Proof: If we consider the true position t, Lemma 6 implies that the scheduler can only force t up if $t \geq K - (n-1) \geq 1$ and down if $t \leq -K + (n-1) \leq -1$. Furthermore if $|t|$ ever exceeds $K+n+(n-1)$, each processor will decide after its next read. Thus the movement of the true position is a controlled random walk in the sense of Lemma 3 with center 0 and barriers at $\pm(K+2n-1)$. The expected number of steps until a barrier is reached is at most $4(K+2n-1)^2$ steps, which will be followed by at most $2n$ operations as the processors each decide. Since each step takes a constant number of counter operations the expected number of operations required is $O((K+n)^2)$.

∎

The time bound of Lemma 7 shows that every processor terminates in finite expected time when $K \geq n$. The bounded bias property is a consequence of the following lemma:

Lemma 8 *Against any scheduler, the probability that the processors in the robust weak shared coin protocol will decide* 1 *is between* $\frac{K-(n-1)}{2K}$ *and* $\frac{K+(n-1)}{2K}$.

Proof: Suppose the scheduler is trying to maximize the probability of deciding 1. Under the simplifying assumption it can force a decision of 1 as soon as $t = K-(n-1)$; however, if it allows t to slip below $-K-(n-1)$ the protocol will eventually decide 0. When $-K-(n-1) < t \leq K-(n-1)$ the scheduler may choose between moving t randomly or forcing t toward $-K-(n-1)$. Clearly, forcing the counter toward $-K-(n-1)$ can only increase the probability of deciding 0, so choosing to move t randomly maximizes the problem of deciding 1. But if the scheduler makes this choice, the movement of the true position becomes a simple random walk with absorbing barriers at $-K-(n-1)$ and $K-(n-1)$. By Lemma 1, the probability that t reaches $K-(n-1)$ first is $\frac{K+(n-1)}{2K}$. The lower bound follows by symmetry. ∎

Combining the lemmas we obtain:

Theorem 9 *When* $K > n$, *the protocol of Figure 1 implements a robust weak shared coin.*

Proof: Consistency follows from Lemma 4, termination from Lemma 7, and bounded bias from Lemma 8. ∎

Lemma 8 allows K to be chosen to obtain arbitrarily small non-negative bias.[1] Let the bias of the shared coin be $\frac{1}{2}+\epsilon$, then

$$\epsilon \leq \frac{n-1}{2K}$$

[1] A wait-free coin with zero bias is known to be impossible in the shared-memory model[6].

which gives

$$K \geq \frac{n-1}{2\epsilon}.$$

Combining this inequality with Lemma 7 gives a bound on the worst-case expected running time for the protocol of $O((n/\epsilon)^2)$ total operations. This time is comparable to the worst-case expected running times of the protocol's non-robust ancestors. The protocol thus achieves robustness without paying a significant cost in speed.

4. Implementing a Bounded Counter with Atomic Registers

The robust weak shared coin assumes the presence of a shared counter supporting atomic increment, decrement, and read operations, with the restriction that no operation will be applied that will move the counter out of some fixed range $[-r, r]$. In practice such a counter is not likely to be available as a hardware primitive. Fortunately it is not difficult to implement a shared counter using atomic registers. However, some care must be taken to guarantee that the counter uses only a bounded amount of space.

Both Aspnes and Herlihy[6] and Attiya et al[7] describe shared counter implementations. The two counter implementations both assign a register to hold the net increment due to each processor, so that the counter's value is simply the sum of the values in these registers. Both algorithms use simple atomic snapshot protocols to allow the entire set of registers to be read in a single atomic action.

Alas, neither implementation does quite what we would like. Even though the value stored in

```
Shared data:
  scannable array count[0 ... n − 1], initialized to 0

procedure increment()
  v := read(count[ProcessorId])
  write((v + 1) mod m, count[ProcessorId])
end

procedure decrement()
  v := read(count[ProcessorId])
  write((v − 1) mod m, count[ProcessorId])
end

function read()
  scan(count)
  v := Σ_{i=0}^{n−1} count[i]
  return v' where −r ≤ v' ≤ r and v' ≡ v (mod m)
end
```

Figure 4: Pseudocode for Counter Operations

the counter will never exceed the range $[-r, r]$, the net increment due to an individual processor is potentially unbounded. The Aspnes and Herlihy protocol ignores this difficulty by assuming the presence of unbounded registers (which it also uses to implement the atomic scan.) The Attiya et al protocol uses only bounded registers, but enforces the bounds by prematurely terminating the weak shared coin protocol if any processor's register wanders out of a limited range. This premature termination occurs infrequently, and is acceptable in a weak shared coin which does not need to guarantee consistency. But it is not acceptable for a robust coin, as it may allow the scheduler to force some processor to choose one value (through premature termination) after another has already chosen a different value (through the normal workings of the shared coin protocol.)

A simple alternative to premature termination that still allows for bounded registers is to store the remainder of each processor's contribution relative to some convenient modulus m greater than the total range $2r + 1$. The counter value can then be reconstructed as the unique v in the range

$[-r, r]$ that is congruent to the sum of the registers, modulo m. Pseudocode for the three counter operations using this technique is shown in Figure 4; it assumes the presence of a scannable array of registers, which can be built in a number of ways[5, 3, 2]. The algorithm of Afek et al[2] in particular allows an atomic scan operation to be implemented with $O(n^2)$ bits of extra space and a maximum of $O(n^2)$ primitive reads and writes per operation.

5. The Randomized Consensus Protocol

Figure 5 shows pseudocode for each processor's behavior in the randomized consensus protocol. The protocol uses three shared counters: the first two maintain a total of the number of participating processors that started with inputs 0 and 1, respectively, and the last is used as the counter for a modified version of the robust weak shared coin protocol. All of the counters start with an initial value of 0.

The protocol is optimized for the case where few processors participate. We will define a processor to be *active* if it takes at least one step before some processor decides on a value, and denote by p the total number of active processors in a given execution. The protocol uses the counters a_0 and a_1 to keep track of the number of active processors by having each processor increment one or the other of these counters as it starts the protocol.

The protocol depends on being able to take an atomic snapshot of the counters. Since the first two counters are never decremented, such a snapshot can be obtained as described in Figure 6. Though the operation there defined is not wait-free, the values of a_0 and a_1 change at most p times during the execution of the consensus protocol and thus any processor will go through at most

```
Shared data:
  counter a_0 with range [0, n], initialized to 0
  counter a_1 with range [0, n], initialized to 0
  counter c with range [−4n, 4n], initialized to 0

procedure consensus(input)
increment(a_input)
repeat
  read(a_0, a_1, c)
  if c ≤ −2n then decide 0
  elseif c ≥ 2n then decide 1
  elseif c ≤ −(a_0 + a_1) or a_1 = 0 then decrement(c)
  elseif c ≥ (a_0 + a_1) or a_0 = 0 then increment(c)
  else
    if coin() = 0 then decrement(c)
    else increment(c)
    fi
  fi
end
end
```

Figure 5: The Consensus Protocol

p extra passes during the entire duration of the protocol. If the counters are not primitives but are instead constructed using an atomic scan operation as described in Section 4, the overhead of Figure 6 can be avoided completely by simple reading all three counters in a single atomic scan of the arrays that implement them.

Several features of the protocol are worth noting. First of all, the same "slopes" that ensured consistency for the robust weak shared coin ensure consistency for the consensus protocol, for the same reasons. Second, counters a_0 and a_1 allow the protocol to guarantee validity, as the random walk is only invoked if both have non-zero values. These counters are also used to minimize the range of the random walk, by taking advantage of the fact stated in the following lemma, a modification of Lemma 6:

procedure $ScanCounters()$
repeat
  $a_0 = read(a_0)$
  $a_1 = read(a_1)$
  $c = read(c)$
  $a_0' = read(a_0)$
  $a_1' = read(a_1)$
until $a_0' = a_0$ **and** $a_1' = a_1$
return a_0, a_1, c
end

Figure 6: Counter Scan for Randomized Consensus Protocol

Lemma 10 *Let a_0, a_1, c be the values read from the counters by some processor and t the true position of the random walk in the state preceding the read. Then $|c - t| \leq a_0 + a_1 - 1$.*

Proof: There are at most $a_0 + a_1 - 1$ processors with pending increments or decrements. ∎

To prove that the consensus protocol is correct, we must establish that it is consistent, that it terminates, and that it is valid. The proof of consistency is a straightforward modification of the proof of Lemma 4:

Lemma 11 *If any processor reads a counter value $v \geq 2n$, then all subsequent reads will return values $\geq n + 1$; in the symmetric case where $v \leq -2n$, all subsequent reads will be $\leq -(n + 1)$.*

Proof: Apply the proof of Lemma 4 with $K = n$. ∎

Similarly, the proof that the counter c does not overflow is a straightforward modification of Lemma 5:

Lemma 12 *The value of c never leaves the range $[-4n, 4n]$ in any execution of the consensus*

protocol.

Proof: Apply the proof of Lemma 5 with $K = n$. ∎

Termination is trickier to demonstrate. As for the weak shared coin, the key to the proof of the consensus protocol's termination is the fact that the scheduler's only alternative to moving the true position randomly is to move it away from the origin. In the weak shared coin protocol, this condition depends on fixing the parameter $K \geq n$. In the consensus protocol the situation is more complicated, as the protocol uses its knowledge of the number of currently active processors to set the inner boundaries of the slope close to the origin while still preventing the scheduler from being able to force the true position to move toward the origin.

Lemma 13 *Let n be the total number of processors and p be the number of processors that take at least one step before some processor decides on a value. Then the worst-case expected running time of the consensus protocol is $O(p^2 + n)$ total counter operations.*

Proof: We will show that the consensus protocol terminates in $O(p^2 + n)$ time by reducing it to a controlled random walk of the true position t. Divide the execution of the protocol into two phases. In the first phase, at most one of a_0, a_1 is nonzero; if the execution does not leave the first phase before $2n$ increments or decrements have occurred the protocol will terminate after $O(n)$ additional steps by Lemma 11.

In the second phase, both a_0 and a_1 are nonzero. Let v be a value read by some processor from the counter c. By Lemma 10 we know that $|t - v| \leq a_0 + a_1 - 1 \leq p - 1$. Now, to force an increment during the second phase the scheduler must show a processor a counter value v that is

at least $a_0 + a_1$, possibly by withholding local coin-flips to raise the value of c or by withholding increments to lower the value of $a_0 + a_1$. In either case Lemma 10 applies and t must be greater than 0. The case of the scheduler attempting to force a decrement is symmetric, and thus in either case the scheduler can only force the true position to move away from 0.

Furthermore, since p is both an upper bound on the distance between c and t and on the value of $a_0 + a_1$, if $|t| \geq 2p$ then $|v| \geq a_0 + a_1$ and the true position will move away from 0 thereafter. Thus the second phase of the execution can be modeled as a controlled random walk in the sense of Lemma 3 with center 0, barriers at $\pm 2p$, and a starting position of equal to the true position at the end of the first phase. By Lemma 3, this random walk will take an expected $O(p^2)$ steps, each consisting of a constant number of counter operations; to this value must be added $O(n)$ steps until termination, up to $O(n)$ steps from the first phase, and $O(p^2)$ extra read operations due to extra passes through the loop in $ScanCounters()$. The total expected number of counter operations is thus $O(p^2 + n)$. ∎

Lemma 14 *The protocol of Figure 5 satisfies the validity condition.*

Proof: Suppose every processor starts with the input 1. Then a_0 is never incremented and thus retains its initial value of 0 throughout the execution of the protocol. Thus each processor will increment c until it reads a value $v \geq 2n$ at which point it will decide 1. The case where every processor has input 0 is symmetric. ∎

Combining the lemmas gives:

Theorem 15 *Figure 5 implements a consensus protocol.*

Protocol	Running Time (total operations)	Space Used (total bits)
[6]	$O(n^2)$	Unbounded
[6] (using atomic registers)	$O(n^4)$	Unbounded
[7]	$O(n^2)$	$O(n^2)$
[7] (using atomic registers)	$O(n^4)$	$O(n^2)$
Figure 5	$O(p^2 + n)$	$O(\log n)$
Figure 5 (using atomic registers)	$O(n^2(p^2 + n))$	$O(n^2)$

Table 1: Comparison of Consensus Protocols

Proof: Lemmas 11, 13, and 14. ∎

6. Discussion

Table 1 compares the time and space used by the consensus protocol described in this paper with the corresponding statistics for the protocols of Aspnes and Herlihy[6] and Attiya et al[7]. All values are expressed as a function of the total number of processors n and the number of active processors p. Two cases are considered for each algorithm. In the first, atomic counters are available as primitive objects. In the second, only atomic registers are available, and we assume that the atomic counters (and whatever other structures are used by each protocol) are constructed from them using the atomic snapshot protocol of Afek et al[2]. The running time in each case is the worst-case expected running time expressed as the total number of operations on the available primitives. The space is expressed as the total number of bits of state in the shared objects.

Table 1 shows three stages of evolution of what is essentially the same basic technique, as all three protocols are based on the technique of using a random walk to minimize the effects of local coin-flips trapped within stopped processors. The principal accomplishment of the present work is to extend the structure of the random walk to include the functions of detecting agreement and guaranteeing validity. Once these functions are handled within the weak shared coin itself,

limiting scheduler control over the outcome of the weak shared coin is no longer necessary to achieve consensus. Thus the parameter K of the weak shared coin protocol can be set to minimize the time taken in the random walk without regard to its effect on the bias ϵ. For example, the weak shared coin embedded in the consensus protocol of Figure 5 has an effective bias of $\frac{p-1}{2p}$, as high as is possible without setting $K < p$.

At the same time, the simplicity of the protocol allows the number and size of the shared counters to be very small. Unfortunately, when the available primitives are limited to atomic registers this reduction is lost in the $\Theta(n^2)$ space overhead of the atomic scan operation. It is not immediately clear that this overhead is a necessary feature of an atomic counter implementation; much work remains to be done in this area.

6.1. Recent Developments

Since the appearance of the first version of this paper[4], a number of advancements have been made in the state of the art of randomized consensus protocols. A brief summary will help to indicate the paths along which consensus protocols have developed.

Saks, Shavit, and Woll[20] have constructed a consensus protocol with several interesting features. The most noteworthy is the revival of the technique of using a non-robust weak shared coin in a multi-round framework. Their non-robust coin is a hybrid of several coins, some of which are optimized for a situation where nearly all processors participate in the protocol and run at bounded relative speeds. The coin which is used in the general case is similar to the random walk based coins of other polynomial-time consensus protocols; however, instead of terminating the walk when a counter moves out of a given range, the walk is stopped after $O(n^2)$ "votes" are collected

regardless of the total value, and the outcome of the coin is determined by the majority of the votes. The non-robustness of the coin arises from the possibility of different processors reading different totals when the walk stops close to the origin. The main advantage of the structure is that the proper functioning of the coin is much less dependent on detecting termination quickly; Bracha and Rachman[8] have recently shown that using similar techniques it is possible to build a weak shared coin which requires only $O(n^2 \log n/\epsilon^2)$ atomic register operations. Unfortunately both the Saks et al protocol and the Bracha and Rachman protocol use an unboundedly large set of registers.

An alternative approach is that taken by Dwork et al[12]. They show that by replacing the atomic snapshot with a weaker "time-lapse snapshot", the protocol of Figure 5 can be made to run in an expected $O(n(p^2+n))$ total atomic register operations. As their time-lapse snapshot requires only bounded-sized registers, this result gives the currently fastest consensus protocol that does not require unbounded space.

6.2. Open Problems

All of the wait-free consensus protocols known to date are based on a weak shared coin which combines local coin flips by taking a majority, either when computing the sign of a total vote or when computing the sign of the position of a random walk. Because the scheduler can choose to withhold up to $n-1$ local coin flips it can control the outcome of such a coin with high probability unless the total of the local coin flips is at least $n-1$ away from even. This property alone means that such a protocol requires at least $\Omega(n^2)$ primitive operations to obtain a constant-bias weak shared coin, even if the protocol uses more powerful deterministic primitives than single-writer

atomic registers. As the protocol of Bracha and Rachman comes very close to this bound it is unlikely that current techniques will lead to much improvement without some new breakthrough. Two questions that should be considered are:

1. Does every wait-free consensus protocol for the shared-memory model contain a weak shared coin?
2. Can a constant-bias weak shared coin be built that requires asymptotically fewer than $\Theta(n^2)$ local coin flips?

7. Acknowledgments

The author would like to thank Hagit Attiya, Maurice Herlihy, Orli Waarts, and Xia Zeqing for their comments on earlier versions of this paper.

References

[1] K. Abrahamson. On achieving consensus using a shared memory. In *Proceedings of the Seventh ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1988.

[2] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. In *Proceedings of the Ninth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 1–14, August 1990.

[3] James H. Anderson. Composite registers. In *Proceedings of the Ninth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 15–29, August 1990.

[4] James Aspnes. Time- and space-efficient randomized consensus. In *Proceedings of the Ninth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 325–331, August 1990.

[5] James Aspnes and Maurice Herlihy. Wait-free synchronization in the asynchronous pram model. In *Second Annual ACM Symposium on Parallel Algorithms and Architectures*, July 1989.

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

[7] H. Attiya, D. Dolev, and N. Shavit. Bounded polynomial randomized consensus. In *Proceedings of the Eighth ACM Symposium on Principles of Distributed Computing*, pages 281–294, August 1989.

[8] Gabi Bracha and Ophir Rachman. Randomized consensus in expected $O(n^2 \log n)$ operations. In *Proceedings of the Fifth Workshop on Distributed Algorithms*, 1991.

[9] B. Chor, A. Israeli, and M. Li. On processor coordination using asynchronous hardware. In *Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing*, pages 86–97, 1987.

[10] B. Chor and L. Moscovici. Solvability in asynchronous environments. In *30th Annual Symposium on Foundations of Computer Science*, pages 422–427, October 1989.

[11] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.

[12] Cynthia Dwork, Maurice Herlihy, Serge Plotkin, and Orli Waarts. Time-lapse snapshots. Unpublished manuscript.

[13] E.B. Dynkin and A.A. Yushkevich. *Controlled Markov Processes*. Springer-Verlag, 1975.

[14] M. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed commit with one faulty process. *Journal of the ACM*, 32(2), April 1985.

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

[16] Maurice P. Herlihy. Impossibility and universality results for wait-free synchronization. In *Proceedings of the Seventh ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1988.

[17] Michael C. Loui and Hosame H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. In Franco P. Preparata, editor, *Advances in Computing Research*, volume 4. JAI Press, 1987.

[18] Nancy Lynch. I/O automata: A model for discrete event systems. Technical Report MIT/LCS/TM-351, MIT Laboratory for Computer Science, March 1988.

[19] S. Plotkin. Sticky bits and universality of consensus. In *Proceedings of the Eighth ACM Symposium on Principles of Distributed Computing*, pages 159–176, August 1989.

[20] Michael Saks, Nir Shavit, and Heather Woll. Optimal time randomized consensus — making resilient algorithms fast in practice. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 351–362, 1991.

[21] M. Santha and U.V. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33:75–87, 1986.