Technion - Israel Institute of Technology
Computer Science Department

Space-Efficient Asynchronous Consensus
without Shared Memory Initialization

by

M.J. Fischer, S. Moran, G. Taubenfeld

Technical Report #707
December 1991
Space-Efficient Asynchronous Consensus without Shared Memory Initialization

Michael J. Fischer* Shlomo Moran † Gadi Taubenfeld ‡

December 22, 1991

Abstract

We study the consensus problem in a shared memory model where all processes are programmed alike, there is no global synchronization, it is not possible to simultaneously reset all parts of the system to a known initial state, and processes may be faulty. We present a consensus protocol for \( n \) processes which can tolerate up to \( \lfloor n/2 \rfloor - 1 \) failures and which uses a single \( (2\lceil 1.5n - 1 \rceil) \)-valued shared register.

1 The Consensus Problem

The consensus problem is to design a protocol in which all correct processes reach a common decision based on their initial opinions [Abr88, DDS87, DLS88, Fis83, FLM86, FLP85, LA88]. While various decision rules can be considered such as “majority consensus”, the problem is interesting even where the decision value is constrained only when all processes are unanimous in their opinions, in which case the decision value must be the common opinion. More formally the problem is defined as follows. There are \( n \) processes \( p_1, p_2, \ldots, p_n \). Each process \( p_i \) has an input value \( x_i \in \{0, 1\} \). It decides on a value \( v \) if it writes \( v \) to its output register. It may decide at most once. The requirements of the consensus problem are that there exist a decision value \( v \) such that:

1. Each non-faulty process eventually decides on \( v \).

2. \( v \in \{x_1, x_2, \ldots, x_n\} \).

In particular, if all input values are the same, then that value must be the decision value.

The problem is fundamental in distributed computing and is at the core of many algorithms for fault-tolerant distributed applications. Achieving consensus is straightforward in the absence of faults, for all processes can share their initial values and then all apply
a uniform decision rule to the resulting vector of initial values. However, such a simple algorithm cannot tolerate even a single process crash, for its correctness depends on all processes obtaining the same initial value vector.

Much is known about the consensus problem in other models. For example, Fischer, Lynch, and Paterson show that no algorithm can tolerate even a single crash failure in a particular asynchronous message-passing model [FLP85]. In this note we consider an asynchronous shared memory model with arbitrary initialization faults and process crash failures. We show that the consensus problem is solvable in our model using a single \((2\lceil 1.5n - 1 \rceil)\)-valued shared register, as long as a majority of the processes are non-faulty.

2 The Model

Our model of computation consists of a fully asynchronous collection of \(n\) identical anonymous deterministic processes that communicate via a single finite sized shared register. Access to the shared register is via atomic "read-modify-write" instructions which, in a single indivisible step, read the value in the register and then write a new value that can depend on the value just read.

The shared register is subject to initialization failure in which an arbitrary unknown value is placed in the shared register before the protocol begins. Processes are subject to crash failures in which a process at an arbitrary time simply ceases to participate further in the protocol. We assume that a majority of the processes are non-faulty. This model captures important properties of real systems and is similar to the model considered in [FMRT90].

Initialization failure relates to the notion of self-stabilizing systems defined by Dijkstra [Dij74]. However, Dijkstra considers only non-terminating control problems such as the mutual exclusion problem, whereas we show how to solve the consensus problem in which a process makes an irrevocable decision after a finite number of steps.

3 Related Work

In [FMRT90] the wakeup problem is defined and solved in a model similar to ours. The following results follow easily from those presented in [FMRT90].

- In the absence of faults, it is possible to reach consensus on one of \(k\) values using a single 5-valued shared register.
- There is no consensus protocol that can tolerate \([n/2]\) failures.
- Let \(P\) be a consensus protocol that can tolerate \([n/2] - 1\) failures, and let \(V\) be the set of shared memory values used by \(P\). Then \(|V| = \Omega(n^{0.63})\).
- There is a consensus protocol that can tolerate \([n/2] - 1\) failures, which uses a single 8\(n\)-valued register. (In fact, the result is that for any \(1 \leq t \leq [n/2] - 1\) there is a consensus protocol that can tolerate \(t\) failures, which uses a single \(O(t)\)-valued register.)

In this note we improve the above upper bound.
4 The consensus protocol

In this section we show how to solve the consensus problem using small shared space.

Consider the n-process protocol for \( n \geq 2 \) given in Figure 1. It uses a shared register whose values range over the set \( B \times C \), where \( B = \{0,1\} \) and \( C = \{0, \ldots, \lceil 1.5n \rceil - 2 \} \). We use \( b \) and \( c \) to designate the first and second components, respectively, of the ordered pair stored in the shared register. Thus, \( b \) ranges over \( B \), \( c \) ranges over \( C \), \( |B| = 2 \), and \( |C| = \lceil 1.5n \rceil - 1 \). Let \( k = \lceil n/2 \rceil \). We call the the interval \( [0, k - 1] \) the 0-interval and the interval \( [k, 2k - 1] \) the 1-interval. These two intervals are disjoint subsets of \( C \) since \( 2k \leq |C| \).

The protocol uses the function \( d(a_1, a_2) = 1 + ((a_2 - a_1 - 1) \mod |C|) \), defined for \( a_1, a_2 \in C \). If we think of the numbers in \( C \) as being arranged clockwise in a circle, then \( d(a_1, a_2) \) is the distance one must travel clockwise around the circle starting from \( a_1 \) before reaching \( a_2 \). (In the special case that \( a_1 = a_2 \), one must travel all the way around the circle.) Thus, \( 1 \leq d(a_1, a_2) \leq |C| \), and \( a_1 + d(a_1, a_2) \equiv a_2 \pmod{|C|} \).

Figure 1 uses constructs lock and unlock to mark the beginning and end of exclusive access to the shared register. A use of lock is valid if every possible execution path beginning at the lock eventually reaches an unlock. A valid use of lock can be replaced by equivalent code which firsts tests local registers, then, depending on the outcome of those tests, performs a read-modify-write operation on the shared register, and finally, depending also on the value returned by the read-modify-write operation, updates the local registers. This code can be constructed by tracing out all execution paths in the original code for all possible values of the local registers, starting from the lock and stopping when an unlock is encountered. It is easily seen by inspection that all uses of lock in Figure 1 are valid.

Thus, the protocol could in principle be translated into equivalent code that uses only read-modify-write operations on a single shared register. We assume that a process does not fail between executing lock and the next unlock. This corresponds to the assumption in our underlying model that a read-modify-write operation is atomic, so a process is assumed not to crash in the middle of such an operation.

Each process, when it starts to operate, reads the value of the shared register, stores its components in local memory, and increments \( c \) by one (modulo \(|C|\)). At any later point, a process becomes the master if it learns (by inspecting \( c \)) that more than half of the processes have waked up but the value of \( b \) has still not changed. The master decides on its own input value, and this value \( v \) will eventually become everyone's decision value. If the master chooses zero, it sets \( c \) to the beginning of the 0-interval; otherwise it sets \( c \) to the beginning of the 1-interval. Simultaneously, it changes the value of \( b \), thereby indicating to processes that are already awake that a decision has been made. When any other processes first learns that the decision has been made, it makes its own decision according to the interval in which \( c \) lies. A process can learn that a decision has been made either by seeing \( b \) change value, or by seeing that \( c \) appears to have been incremented by more than \( n \) (modulo \(|C|\)). Every process, after making its decision, continues forever setting \( c \) to the beginning of the interval corresponding to its decision.

**Theorem** There exists a consensus protocol for \( n \) processes which can tolerate arbitrary initialization failures and as many as \( \lceil n/2 \rceil - 1 \) process crash failures and which uses a single \( (2\lceil 1.5n - 1 \rceil)\)-valued shared register.
Protocol for process $p_i$ with input value $x_i$:

constant $B = \{0, 1\}$
constant $C = \{0, \ldots, \lfloor 1.5n \rfloor - 2\}$
constant $k = \lceil n/2 \rceil$

shared register $(b, c)$ ranges over $B \times C$
local register $(b_i, c_i)$ ranges over $B \times C$
output register $v_i$ ranges over $\{0, 1, \lambda\}$, initially $\lambda$
lock

$(b_i, c_i) := (b, c)$
$c := (c + 1) \mod |C|$
unlock

while $v_i = \lambda$ do
  lock
  case $b$ of
    $b_i$: if $d(c_i, c) > n$ then $v_i := \lfloor c/k \rfloor$ /* make decision */
  else if $d(c_i, c) > n/2$ then $v_i := x_i$; $b := 1 - b$; $c := v_i \ast k$ /* become master */
  $1 - b_i$: $v_i := \lfloor c/k \rfloor$ /* make decision */
  unlock
  end-case
  end-lock
end-while
repeat forever
  lock
  $c := v_i \ast k$
  unlock
end-repeat

Figure 1: $n$-process consensus using $2\lceil 1.5n - 1 \rceil$ values
Proof: The case \( n = 1 \) is trivial. Assume now \( n \geq 2 \). We show that the protocol given in Figure 1 satisfies all of the conditions of the theorem.

We consider separately the cases \( n = 2 \) and \( n > 2 \). If \( n = 2 \), then both \( b \) and \( c \) are binary registers and we assume no failures. Then for all \( a_1, a_2 \in C \), \( d(a_1, a_2) = 1 \) if \( a_1 \neq a_2 \) and \( d(a_1, a_2) = 2 \) if \( a_1 = a_2 \). By inspection of Figure 1, the first process \( p_1 \) to wake up will make \( c_i \neq c \) and will wait until \( c_i = c \). This will become true when the second process, \( p_2 \), wakes up and increments \( c \). On \( p_1 \)'s next step, it will become master and decide on its value \( x_i \), and it will set \( b := 1 - b \) and \( c := x_i \). When \( p_2 \) next takes a step, it will notice that \( b = 1 - b \) and will decide on \( c = x_i \).

Assume now that \( n \geq 3 \). We use the notation \([a_1, a_2]\) to denote the set of values in \( C \) that lie on the circle between \( a_1 \) and \( a_2 \), inclusive, i.e.,

\[
[a_1, a_2] = \{a_1, a_2\} \cup \{a : d(a_1, a) + d(a, a_2) < |C|\}.
\]

We extend this notation to arbitrary integers by first reducing \( a_1 \) and \( a_2 \) modulo \(|C|\), i.e.,

\[
[a_1, a_2] = ([a_1 \mod |C|], [a_2 \mod |C|]).
\]

To see that this protocol works, one must verify several properties. In every run with fewer than \( n/2 \) faulty processes, there is exactly one master. The value of \( b \) is changed exactly once during the run, and that change is made by the master. The step at which \( b \) is changed is called the decision step. Before the decision step, no process decides on an output value, \( b = \overline{b} \) and \( c = (\overline{c} + w) \mod |C| \), where \((\overline{b}, \overline{c})\) is the initial value of the shared register and \( w \) is the number of processes that have waked up so far. After the decision step, \( b = 1 - \overline{b} \) and \( vk \leq c \leq vk + w' < vk + k - 1 \), where \( v \) is the master's decision value and \( w' \) is the number of processes that have waked up since the decision step. Every process that decides after the decision step chooses \( v \) as its decision value, and every non-failing process eventually decides. This implies the correctness of the protocol.

We now substantiate these claims. A formal proof would verify them by simultaneous induction on the length of the run. We instead argue informally for their correctness.

Any process \( p_i \) that wakes up before the decision step will have \( b_i = \overline{b} \) and \( c_i \in [\overline{c}, \overline{c} + n - 1] \). Thereafter, until the decision step, \( p_i \) will see \( b = b_i \) and \( c \in [c_i + 1, \overline{c} + n] \), so \( d(c_i, c) \leq n \). Thus, no process \( p_i \) will decide before the decision step.

Some process will become master, and hence there will be a decision step. Suppose not. At least \([n/2] + 1 > n/2 \) process are non-failing and eventually wake up. Let \( p_i \) be the first non-failing process to wake up. Then eventually \( c = c_i + |n/2| + 1 \) will become true. Thereafter, it will always be true that \( n/2 < d(c_i, c) \leq n \). But then \( p_i \) becomes master on its next step, a contradiction.

We next show that after the decision step, no process becomes master or changes the value of \( b \), and \( c \) always lies in the \( v \)-interval, where \( v \) is the decision value of the master. Hence, any process that makes a decision chooses \( v \).

The master sets \( c \) to \( vk \), the start of the \( v \)-interval, when it makes its decision. Thereafter \( c \) is incremented as new processes wake up and is reset to \( vk \) by processes that have already decided on \( v \). Since at most \( k - 1 \) processes wake up after the decision step, \( c \) is confined to the \( v \)-interval \([vk, (v + 1)k - 1]\). Therefore, any process that makes its decision during this time on the basis of the interval in which \( c \) lies will decide on the master's value \( v \).

Any process \( p_i \) that wakes up before the decision step has \( b_i = \overline{b} \), so the next time that it takes a step after the decision step, it will see \( b = 1 - \overline{b} = 1 - b_i \) and will decide on \( v \). In
particular, it will not itself become a master. Any process $p_j$ that wakes up after the decision step will set $c_j$ to a value in the interval $[v_k, (v + 1)k - 2]$. This is because $c$ is confined to the $v$-interval, as noted above, and $c_j \neq (v + 1)k - 2$ since at most $k - 2$ other processes woke up and incremented $c$ after the decision step and before $p_j$ wakes up. Thereafter, $p_j$ will always see a value of $c$ in the $v$-interval. It follows that either $d(c_j, c) \leq n/2$, in which case $p_j$ does nothing, or $d(c_j, c) > n$, in which case $p_j$ decides on $v$. To see this, we consider three cases: If $c_j = c$, then $d(c_j, c) = |C| > n$. If $c_j \neq c$ and $c \in [c_j, (v + 1)k - 1]$, then $d(c_j, c) \leq d(v_k, (v + 1)k - 1) = k - 1 < n/2$. If $c_j \neq c$ and $c \in [v_k, c_j]$, then $d(c_j, c) \geq d((v + 1)k - 2, v_k) = |C| + v_k - ((v + 1)k - 2) = ([1.5n] - 1) - k + 2 = n + 1 > n$.

It remains to show that every non-failing process eventually makes a decision. First, at most $k - 1$ processes are faulty, and at least $|n/2| + 1 > n/2 > k - 1$ processes wake up before or at the decision step. Hence, at least one of these is non-faulty and is either the master or will make its decision on the first step it takes after the decision step. Forever after, it repeatedly resets $c$ to $v_k$, so $c$ will eventually stabilize to $v_k$. Any process $p_j$ that has not already decided by that time will decide on $v$ at its next step, either because it sees $b \neq b_j$ or because it sees $d(c_j, c) > n$. Thus, all non-failing processes decide on $v$, the value chosen by the master.

References


