Fast Adaptive Multiple-Shot Consensus  
(Research Note)

Roy Friedman  
Computer Science Department  
Technion  
Haifa, 32000 Israel  
E-mail: roy@cs.technion.ac.il  
May 9, 2002

Abstract

This paper propose a simple optimization that enhances the performance of multiple invocations of the Chandra and Toueg protocol for solving consensus using a $\omega W$ failure detector. This optimization is especially useful for consensus-based total ordering and group membership protocols.

1 Introduction

Distributed consensus is one of the most fundamental problems in distributed computing. In particular, it was shown in [5, 7] how to solve many related distributed agreement problems, such as group membership, atomic multicast, etc. based on a consensus service. While it is known that consensus cannot be solved in a completely asynchronous system prone to a single crash failure [4], it was shown in [2] that consensus can be solved in an asynchronous system augmented with an unreliable failure detector. The significance of that work is two fold: First, it means that consensus can be solved in the existence of very minimal synchrony assumptions. Second, it cleanly abstracts away all timing assumptions from consensus protocols. Specifically, Chandra and Toueg presented such protocol in [2], which always maintains safety, even if the failure detector does not behave according to its specification, although it might not terminate under such conditions. This protocol also has the nice property that if no failures occur, the protocol terminates within three communication steps, although a failure of the coordinator can lead to additional four communication steps. However, the identity of the coordinator is hard-coded in this protocol. This presents a problem in multiple invocations of the protocol, such as when consensus is used as a building block for atomic delivery and group communication [5, 7]. Specifically, if the coordinator fails, consecutive invocations of the protocol will always take at least four communication steps plus the time for the failure detector to suspect the coordinator.

In a recent paper, Boichat, Dutta and Guerraoui [1] introduced the concept of asynchronous leasing as a way to overcome this shortcoming in the protocol of Chandra and Toueg. The idea behind that work is that a process can obtain a lease from the other nodes for being a coordinator for a certain number of invocations of the consensus protocol. Thus, if a coordinator fails, only the rounds to which that coordinator had a lease would require four steps, but most incarnations of the protocol would terminate in three. Obtaining an asynchronous lease, however, requires
two communication steps, which presents the following tradeoff: The longer the lease is, the more efficient are failure free consecutive invocations of the consensus protocol. On the other hand, all invocations of the consensus protocol after a failure and before the lease expires will be slow, which encourages short leases.

In this work, we propose a much simpler solution to the same problem that only requires $cW$ ($cS$), and is more efficient than the solution proposed in [1]. Specifically, in the scheme we propose here, each invocation of the Chandra-Toueg consensus protocol [2] also runs a consensus on the coordinator for the next invocation. Thus, a failure of a coordinator, in the worst case, slows down the next invocation of consensus, but afterwards, all instances of the consensus protocol are fast. Moreover, this scheme only requires constant space overhead, and does not impose any new messages.

The optimization presented here was also mentioned very informally in [7] for the Paxos algorithm [6]. On the other hand, here we make the details of such an optimization precise, and work with the Chandra-Toueg protocol [2]. The latter protocol has the nice property that it relies on the abstract notion of a failure detector, which eliminates spurious leaders as in the Paxos protocol. Finally, our work assumes no message loss. It is a trivial to incorporate the changes we made into the variant of the Chandra-Toueg protocol that was presented in [3] and is tolerant to message loss.

## 2 The Protocol

We assume here the same asynchronous distributed model as in CT91,BDG02, augmented with an unreliable failure detector from the $cS$ class. The protocol is essentially the same protocol of Chandra and Toueg, with the following addition. The estimate maintained by each node includes two fields, $val$ – the proposed value to decide on, and $coord$ – the proposed coordinator for the next invocation of the algorithm. In each invocation, $val$ is initialized to the initial value of the process and $coord$ is initialized to the process id. Also, the initial coordinator for Invocation $i$ is the coordinator decided on in Invocation $i-1$ except for the first invocation in which it is node 0.

Each invocation of the protocol runs in multiple rounds, where a new round starts if the coordinator of the previous round was suspected to have failed. Each round consists of four phases, except for the first round that skips the initial phase. The parameter $m_p$ is the invocation number, the parameter $v_p$ is the initial value for $p$ for this invocation, and the local variable $r_p$ counts the round number inside the invocation.

### 2.1 Correctness

The correctness proof for the original protocol can be used for the protocol in Figure 1 as well, since it does not rely on the contents of the value being decide on. Thus, each invocation of the protocol achieves agreement on the value and on the initial coordinator for the next invocation. Moreover, the elected coordinator that is decided on in an invocation $m_p$ has to participate in Invocation $m_p$, since each process proposes itself in its initial $estimate_p$ and the decision is always one of the values proposed.

### 2.2 Performance

The performance measure we use is the number of communication phases. Specifically, a communication step is a phase in the algorithm in which a process needs to send a message to all other processes, or a process has to wait for messages from a majority of processes.
Each process maintains a global variable

\[ \text{elected\_coordinator}_p \rightarrow \text{array of integers}\]

initially \[ \text{elected\_coordinator}_p[0] = 0, \text{elected\_coordinator}_p[i] = -1 \text{ for } i > 0 \]

To decide on message number \( m_p > 0 \), each process \( p \) executes the following code:

**procedure** propose \((r_p, m_p)\)

\[ \text{estimate}_p, \text{val} \rightarrow r_p; \text{estimate}_p, \text{coord} \rightarrow p \]
\[ \text{state}_p \rightarrow \text{undecided} \]
\[ r_p \rightarrow 0 \]
\[ ts_p \rightarrow 0 \]

{ Rotate through coordinators, starting with last elected coordinator, until decision is reached }

while \( \text{state}_p = \text{undecided} \)

\[ c_p \rightarrow (\text{elected\_coordinator}_p[m_p - 1] + r_p) \mod n \{ c_p \text{ is the coordinator for this round } \} \]
\[ r_p \rightarrow r_p + 1 \]

if \( r_p = 1 \) then

if \( c_p = p \) then

send \((p, m_p, r_p, \text{estimate}_p)\) to all
goto Phase 3

Phase 1: \{ each process sends its estimation to the coordinator \}

send \((p, m_p, r_p, \text{estimate}_p, ts_p)\) to \( c_p \)

Phase 2: \{ current coordinator collects majority estimates and proposes a new

estimate corresponding to largest timestamp \}

if \( p = c_p \) then

wait until \([\frac{n+1}{2}] \text{ processes } q : \text{ received } (q, m_p, r_p, \text{estimate}_q, ts_q) \text{ from } q \]

\[ msg_p[r_p, m_p] \leftarrow \{(q, m_p, r_p, \text{estimate}_q, ts_q) \text{ from } q \} \]

\[ t \leftarrow \text{largest } ts_q \text{ such that } (q, m_p, r_p, \text{estimate}_q, ts_q) \in msg_p[r_p, m_p] \]

\[ \text{estimate}_p \leftarrow \text{select one } \text{estimate}_q \text{ such that } (q, m_p, r_p, \text{estimate}_q, t) \in msg_p[r_p, m_p] \]

send \((p, m_p, r_p, \text{estimate}_p, ts_p)\) to all

Phase 3: \{ all processes wait for new estimate from coordinator \}

wait until \([\text{received } (c_p, m_p, r_p, \text{estimate}_c) \text{ from } c_p \text{ or } c_p \text{ is suspected}] \]

if \( \text{received } (c_p, m_p, r_p, \text{estimate}_c) \text{ from } c_p \) then

\[ \text{estimate}_p \rightarrow \text{estimate}_c \]

\[ ts_p \rightarrow r_p \]

send \((p, m_p, r_p, \text{ack})\) to \( c_p \)

else

send \((p, m_p, r_p, \text{ack})\) to \( c_p \)

Phase 3: \{ if coordinator can collect majority acks, it sends everyone a decision message \}

if \( p = c_p \) then

wait until \([\frac{n+1}{2}] \text{ processes } q : \text{ received } (q, m_p, r_p, \text{ack}) \text{ or } (q, m_p, r_p, \text{ack}) \]

R-broadcast \((p, m_p, r_p, \text{estimate}_p, \text{decide})\)

when R-deliver \((q, m_p, r_p, \text{estimate}_c, \text{decide})\) \{ if a process receives a decision message, it decides accordingly \}

if \( \text{state}_p = \text{undecided} \) then

\[ \text{decide}(\text{estimate}_c, \text{val}) \]

\[ \text{elected\_coordinator}_p[m_p] \leftarrow \text{estimate}_c, \text{coord} \]

\[ \text{state}_p \leftarrow \text{decided} \]

Figure 1: Efficient consensus protocol – code of invocation \( m_p \) for process \( p \)
As long as there are no failures, each invocation of the protocol will terminate in 3 communication steps. Moreover, when there is a failure, the worst that can happen is that the coordinator of a round fails and that coordinator is also elected to be the initial coordinator for the next invocation. In such a case, both invocation require the four step recovery phase, but after that, additional invocations of the consensus protocol will terminate in three communication steps.

3 Final Comments

Rachid Guerraoui has pointed out a couple of advantages that the asynchronous leasing method has over our method. First, Asynchronous leasing allows running multiple instances of the consensus protocol in parallel. On the other hand, in our approach, each invocation must wait for the previous one to terminate. This is not so bad for atomic delivery, in which it may not make sense to decide on message $i$ before it is known which message is number $i - 1$, but might be significant in other applications. The second advantage of asynchronous leasing relates to the stability guarantee of $\diamond S$ failure detectors. Such failure detectors only guarantee that eventually some correct process will never be suspected, but do not guarantee anything about the others. Thus, in a system with three processes, $p_1$, $p_2$, and $p_3$, it is possible that $p_3$ is never suspected, but $p_1$ and $p_2$ are continuously suspected alternatingly. Specifically, in one invocation $p_1$ will be elected coordinator, but in the next one it will be suspected, and $p_2$ will be elected. Then, in the next run $p_2$ will be suspected and $p_1$ elected. The result is that all runs will be slow. Of course, this behavior is not very likely, especially in LANs, but might occur in a WAN.

References


