Message and time efficient consensus protocols for synchronous distributed systems

Xianbing Wang\textsuperscript{a}, Yong Meng Teo\textsuperscript{a,\footnote{Corresponding author. Fax: +65 6779 4580. E-mail address: teoym@comp.nus.edu.sg (Y.M. Teo).}}, Jiannong Cao\textsuperscript{b}

\textsuperscript{a}College of Computer Science and Technology, Wuhan University, 430072, China
\textsuperscript{b}Department of Computing, Hong Kong Polytechnic University, Hong Kong

Received 23 August 2006; received in revised form 12 July 2007; accepted 31 August 2007
Available online 18 September 2007

Abstract

For a synchronous distributed system of \(n\) processes with up to \(t\) potential and \(f\) actual crash failures, where \((t < n - 1, f < t)\), the time lower bound for a protocol to achieve consensus is \(\min(t + 1, f + 2)\) rounds. Currently, most researches in this field focus on the time efficiency of consensus protocols. This paper proposes consensus protocols for synchronous distributed systems that achieve both message and time efficiency. Based on an early stopping consensus protocol for synchronous distributed system with crash failures, we propose a rotating coordinator scheme that significantly reduces message complexity. However, this protocol is not time efficient because it requires \(\min(t + 1, f + 3)\) rounds to reach consensus. Thus, to achieve both time and message efficiency, we propose another protocol in which \((t + 1)\) coordinators are used to send messages in each round. Furthermore, we show that the proposed consensus protocol with crash failures can be revised to be more message-efficient with orderly crash failures. When a process is able to send more than one message to another in a round, we propose an optimal message efficient early stopping consensus protocol for synchronous distributed systems with orderly crash failures.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Consensus; Synchronous distributed systems; Message efficient; Orderly crash failure; Early stopping

1. Introduction

Consensus is one of the fundamental problems in distributed computing theory and practice. In a consensus problem, assume a distributed system consists of a set of \(n\) processes, \(\Pi = \{p_1, \ldots, p_n\}\), where each process \(p_i\) initially proposes a value \(v_i\), and all non-faulty processes have to decide on one common value \(v\) which is proposed by some processes. A process is faulty during an execution if its behavior deviates from that prescribed by its algorithm, otherwise it is correct. There are various failure models such as Byzantine fault, omission fault and crash fault\[18\]. In this paper, we focus on the crash failure model, in which a process is faulty when it crashes during an execution. More precisely, the consensus problem is defined by the following three properties:

(1) \textbf{Termination:} Every correct process eventually decides on a value.

(2) \textbf{Validity:} If a process decides on \(v\), then \(v\) has been proposed by some process.

(3) \textbf{Agreement:} No two correct processes decide differently.

The agreement property applies only to correct processes. Thus, it is possible that a process decides on a distinct value just before becoming faulty\[6\]. The uniform consensus prevents this by replacing the agreement property with the following:

(3') \textbf{Uniform Agreement:} No two processes (correct or not) decide differently.

Consensus has been extensively studied over the last two decades both in synchronous\[1–3,7,8,13\] and asynchronous\[4,10\] distributed systems. This paper focuses on consensus protocols for synchronous distributed systems. In a synchronous distributed system, each pair of processes is
connected by a channel. Both message delay and relative process speed are bounded, and these bounds are known. The underlying communication system is assumed to be failure-free: there is no creation, alteration, loss or duplication of message. In synchronous consensus protocols, every execution of the protocol consists of a sequence of rounds. Every process will start and finish the same round simultaneously.

Most consensus protocols for synchronous distributed systems focus on achieving time efficient lower bound based on the number of rounds required [1–3,8,13]. There are few studies on the overhead or the message efficiency of time efficient synchronous consensus protocols [3,19]. The rotating coordinator paradigm has been proposed to reduce message complexity of consensus protocols [3]. In each round, only the rotating coordinator sends messages and the other processes wait to receive these messages.

To investigate the message and time efficiency of synchronous consensus protocols, we focus on two key design considerations: the number of messages a process can send to another process in each round [20], and the failure model. Most existing synchronous consensus protocols are designed for the following model, the S-Round model [1–3,8,13], in which each process in round \( r \) executes the following steps sequentially:

1. Send round \( r \) messages to the other processes, with at most one message to a destination process;
2. Wait for round \( r \) messages from the other processes;
3. Execute local computations.

The S-Round model can be extended by relaxing the first step of each round as follows:

1. Send round \( r \) messages to the other processes, in which more than one message can be sent to a destination process.

This new round-based synchronous model is called the M-Round model. Every process in both round-based synchronous models executes local computations based on the messages received at the end of the current round.

Most existing synchronous consensus protocols designed for the S-Round model tolerate crash failures [1–3,8,13]. When a process crashes in a round, it sends a subset of the messages to be sent in that round, and does not execute any subsequent rounds [13]. If a protocol allows processes to reach consensus in which at most \( t \) (\( t < n - 1 \)) processes can crash, the protocol is said to tolerate \( t \) faults or to be a \( t \)-resilient consensus protocol. It has been proved [1,2,13] that in the S-Round model, the lower bound on the number of rounds is \( t + 1 \) for any consensus protocol tolerating up to \( t \) crash failures. Merritt’s result [14] is actually stronger than the one described above because it established the \( t + 1 \) rounds lower bound for the restricted failure model of “orderly crash failures” where faulty processes must respect the order specified by the protocol in sending messages to its neighbors. It is obvious that a lower bound for uniform consensus or consensus with orderly crash failures also holds for those with crash failures. But the reverse may not be true.

Fig. 1 shows the orderly crash failure model. Process \( p_j \) is specified to send messages \( m_{j_1}, \ldots, m_{j_i}, \ldots, m_{j_k} \), in a round to processes \( p_{j_1}, \ldots, p_{j_i}, \ldots, p_{j_k} \), respectively, where \( k \) is the number of messages prescribed to be sent by \( p_j \) and \( j_1, \ldots, j_i, \ldots, j_k \in \{1, \ldots, n\} \). Assume \( p_j \) fails to send message \( m_{j_i} \) and stops by doing nothing. Then every non-crashed process, \( p_{j_i} \), must have successfully received \( m_{j_1} \), where \( 1 \leq i \leq i - 1 \), and processes \( p_{j_1}, \ldots, p_{j_{i-1}} \) will not receive messages from \( p_j \) in the same round. Thus, when \( p_{j_{i-1}} \) receives the message, it can conclude that every non-crashed process \( p_{j_i} \) must have successfully received \( m_{j_{i-1}} \), where \( 1 \leq i \leq i - 2 \).

This illustration is suitable for orderly crash failure models in both the M-Round and S-Round models. Because a process can send multiple messages to another process in a round in the M-Round model but only one in the S-Round model, \( j_1, \ldots, j_i, \ldots, j_k \) are different in the S-Round model, but in the M-Round model \( j_i \) may be equal to \( j_{i-1} \), such that \( p_j \) may send several messages to the same process in a round.

Based on the failure model, the protocols can be divided into crash failures and orderly crash failures. Thus, we have four different system models for the consensus problem as shown in Table 1.

There is only one existing early-deciding uniform consensus protocol called OptEDAUc [5], which can be seen as an example of the M-Round model with crash failures. Because in this scenario, all messages sent from a process to another in one round can be combined into one message, the rotating coordinator based message efficient consensus protocol for S-Round with crash failures also works in M-Round with crash failures. Thus, without combining with the orderly crash failure model, it is not meaningful to discuss consensus problem in the M-Round model with crash failures. The main contributions of this paper are:

- We propose a number of rotating coordinator-based message efficient consensus protocols for the S-Round model with crash failures. The rotating coordinator based early stopping consensus protocol, called the S-Protocol, reduces message complexity significantly, but it requires \( \min(t + 1, f + 3) \)

![Fig. 1. Example of an orderly crash failure model.](image_url)
2.1. S-Round model with crash failures

To achieve both time and message efficiency, we propose a protocol in which \((t + 1)\) coordinators send messages in each round. However, we cannot prove the proposed protocol achieves the message complexity lower bound in this case.

- We extend the S-Protocol to a rotating coordinator-based time and message efficient consensus protocol for the S-Round model with orderly crash failures. This protocol called the MO-Protocol is more message efficient than the S-Protocol.

- We extend the S-Protocol to a rotating coordinator-based message efficient consensus protocol for the M-Round model with orderly crash failures. This protocol called MO-Protocol is both time and message efficient. In each round of the MO-Protocol, the current rotating coordinator sends two messages to other coordinator processes. However, in the SO-Protocol, all second messages are delayed to be sent in next round. Because in the orderly crash failure model, receivers will receive messages sent from the current rotating coordinator in prescribed order, the MO-Protocol saves one round to achieve consensus when compared with the SO-Protocol.

Round duration can differ in different models. It is clear that the round duration in orderly crash failure model is much longer than the crash failure model. In this paper, we consider the number of rounds only in the time complexity analysis. To improve the number of round lower bound in the M-Round model with orderly crash failures, we propose the M-Round model and the MO-Protocol. The message complexity lower bound of the consensus problem has been proved by Tzeng and Siu [19]. They proposed a reliable broadcast protocol very similar to the MO-Protocol but without the notion of rounds. They did not consider the time complexity lower bound in their paper. But the proof ensures that the MO-Protocol also achieve the message complexity lower bound. Thus, the MO-Protocol is the most message efficient and time efficient when considering only the number of rounds.

The rest of the paper is organized based on the four different system models presented. Section 2 reviews the time lower bound of consensus protocols. For crashes failures, Section 3 describes our message efficient consensus protocols for S-Round. For orderly crash failures, Sections 4 and 5 present message efficient consensus protocols for S-Round and M-Round, respectively. Section 6 compares and summarized our main results. Finally, Section 7 concludes this paper.

2. Review of time lower bounds

2.1. S-Round model with crash failures

If a protocol achieves consensus and stops before round \(t + 1\) when there are actually \(f\) faults, we call this an early stopping protocol. Lamport and Fisher [12] pointed out that if only \(f < t\) processes crash, no algorithm can avoid sending round \(f + 2\) messages in the S-Round model. Dolev et al. discuss early stopping in Byzantine agreement for the S-Round model with crash failures [8]. First, they consider the orderly crash failure model to prove that there is no early-stopping protocol for simultaneous Byzantine agreement in which all processes must decide in the same round, which means the lower bound of simultaneous Byzantine agreement is \((t + 1)\) rounds even when only one process crashes actually occur. Then they prove that \(\text{min}(t + 1, f + 2)\) rounds is the lower bound for early-stopping eventual Byzantine agreement. However, proving the lower bound of \(\text{min}(t + 1, f + 2)\) rounds for the orderly crash failure model remains an open problem.

The class of protocols in which all processes decide before round \(t + 1\) with actual \(f\) faults are called early-deciding protocols. Charron-Bost and Schiper [5] prove that the lower bound for early deciding consensus protocols in the S-Round model with crash failures is \((f + 1)\) rounds, and the lower bound also holds for orderly crash failures because it is deduced from Merritt’s lower bound. They also prove that early deciding uniform consensus requires at least \((f + 2)\) rounds if \(f\) is less than \(t - 1\) (this lower bound has been proved using three other methods: layering [11,15], oracle argument [22], and bivalency argument [21]), and only requires \((f + 1)\) rounds if \(f = t - 1\) or \(f = t\). However, Charron-Bost and Schiper [5] comment that their proof of the lower bound for early-deciding uniform consensus works only for crash failures, and whether this result still holds for the restricted class of orderly crash failures remains an open problem.

Charron-Bost and Schiper [5] also show that early deciding uniform consensus is reducible to early stopping consensus. Consider a consensus algorithm \(A\) in which each correct process eventually reaches a stopping state; \(A\) can be transformed into an algorithm \(B = T(A)\) which is identical to \(A\), except that each process postpones its decision until it stops (the decision value in \(B\) is thus the same as in \(A\)). They prove that the \(B\) algorithm solves the uniform consensus problem. Thus, the lower bound for early deciding uniform consensus also holds for early stopping consensus. But the two lower bounds are different when \(f = t - 1\) as shown above. Charron-Bost and Schiper presented an early deciding uniform consensus algorithm, the Tree, algorithm, in which every process can decide by the end of round \(f + 1\) when \(f = t - 1\).

2.2. M-Round model with crash failures

In [20], we propose the M-Round model. The early-deciding uniform consensus protocol, OptEDAUC, proposed in [5] can be seen as an example protocol of the M-Round model. OptEDAUC is designed to achieve the lower bound for early deciding uniform consensus problem for the S-Round model with crash failures. It consists of the EDAUC and Tree, algorithms which are also proposed in [5]. In OptEDAUC, each process executes EDAUC and Tree, in parallel, during the first \(t - 1\) rounds; a process decides according to EDAUC, that is if EDAUC allows it to decide; otherwise, during round \(t\) or \(t + 1\), a process decides according to Tree, unless it receives a \((D, v)\) message from EDAUC, which means it should decide \(v\) and an EDAUC decision prevails over a Tree, decision. Clearly, in each round, a process sends an EDAUC message and a Tree, message to another process. Thus, OptEDAUC is a consensus
consensus and early deciding uniform consensus are orderly crash failures the lower bounds of both early stopping protocol in the M-Round model with crash failures are the same as in the S-Round model with crash failures [20] because all messages sent from one process to another in one round can be combined into a single message.

2.3. S-Round model with orderly crash failures

To solve the open problem discussed in [8], we use the bivalency argument to prove that the lower bound for early-stopping consensus in the S-Round model with orderly crash failures is \( \min(t+1, f+2) \) rounds [20]. Using the bivalency argument, we show that the lower bound for early deciding uniform consensus in the S-Round model with orderly crash failures is \( f+2 \) rounds when \( 0 \leq f \leq t-2 \) [20]. This addresses the open problem in [5]. Both proofs are extended from the bivalency argument proof for early deciding uniform consensus for synchronous distributed systems with crash failures [21].

2.4. M-Round model with orderly crash failures

In [20], we propose an early-deciding uniform consensus protocol for the M-Round model with orderly crash failures, which can always achieve uniform consensus by the end of round \( f+1 \). Next, we prove that in the M-Round model with orderly crash failures the lower bounds of both early stopping consensus and early deciding uniform consensus are \( f+1 \) rounds.

The time lower bounds for both round-based models with various failure models are shown in Table 2.

### Table 2

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Crash failures</th>
<th>Orderly crash failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-Round</td>
<td>M-Round</td>
<td>Orderly crash failures</td>
</tr>
<tr>
<td>Early-stopping consensus</td>
<td>( \min(t+1, f+2) ) [8]</td>
<td>( \min(t+1, f+2) ) [20]</td>
</tr>
<tr>
<td>Early-deciding uniform consensus</td>
<td>( f+2, 0 \leq f \leq t-2 )</td>
<td>( f+1 )</td>
</tr>
<tr>
<td></td>
<td>( f+1, \text{otherwise} )</td>
<td>( f+1, \text{otherwise} )</td>
</tr>
</tbody>
</table>

3. Message efficient protocols for S-Round with crash failures

In this section, we show how the early stopping and deciding factor affects the time efficiency of synchronous consensus for S-Round with crash failures, and how rotating coordinator paradigm affects the message efficiency of synchronous consensus for S-Round with crash failures.

3.1. Flood-set consensus protocol

Fig. 2 presents a variant of the well-known Flood-set consensus protocol [2,13]. Each process \( p_i \) invokes the function \( \text{Consensus}(v_i) \) where \( v_i \) is the value it proposes. It stops with the invocation of the return statement that provides the decided value. \( \text{Consensus()} \) is made up of \( t+1 \) rounds whose aim is to fill in an array \( V[1..n] \) in such a way that \( V[j] \) contains the value proposed by \( p_j \) (Fig. 2, lines 7–10). The flood-set strategy used to attain this goal is particularly simple: in round \( r \), a process sends all the new information it received during the previous round, round \( r-1 \) (Fig. 2, line 6). The management of the other local variables of a process \( p_i \) is self-explanatory (\( \bot \) denotes a default value that cannot be proposed). It always needs \( t+1 \) rounds to achieve consensus. The protocol actually achieves uniform consensus because all processes decide by the end of round \( t+1 \) and they must have maintained the same values in their arrays at that time.

**Message cost:** In the best case, with no crash occurring in the execution, each process sends a message to all processes in the first two rounds, thus sending a total of \( 2n^2 \) messages. In the worst case, there is one and only one crash occurring in each of the first \( t \) rounds, and each crash causes every alive process (at any time, we call a process alive if the process does not crash at that time) to send a message to all processes in the next round, thus \( (t+1)n^2 \) messages are sent in total. If there are actually \( f \) crashes in an execution, in the worst case, \( (f+1)n^2 \) messages are sent.

3.2. Early stopping

Fig. 3 presents the Protocol_RA Y02 protocol for synchronous systems proposed in [17]. The protocol is an early-stopping variant of the above well-known flood-set consensus protocol. When considering early-stopping or early-deciding, each process must send a message to all processes in each round, even if its content is an empty set (Fig. 3, line 6).
Protocol RA02 can achieve uniform consensus. The proof is straightforward and can be found in [17]. The message complexity of Protocol RA02 is \( \min(t+1, f+2)n^2 \), because in each round each process has to send a message to all processes. However, in the Flood-set protocol, a process sends a message to all processes only when it gets new votes. Thus, Protocol RA02 is less message efficient than the Flood-set protocol.

In Protocol RA02 [17], a flag is set to direct a process to decide at the end of the next round if it has learned all information that can be known (Fig. 3, lines 12, 13). But it does not specify sending the decision flag or decision to other processes. A full version of Protocol RA02 with the flag transmitted by a variable called \( w_i \) can be found in [16].

### 3.3. Message-efficient rotating coordinator protocol

Another way to reduce message complexity is to use the rotating coordinator paradigm [3, 9], where in each round only the coordinator sends messages to other processes. Fig. 4 presents a \( t \)-resilient consensus protocol for synchronous distributed systems. \( \text{Consensus}(v) \) is made up of \( t+1 \) rounds. Each round \( r (1 \leq r \leq t+1) \) is managed by a predetermined coordinator \( p_r \). In each round \( r, p_r \) sends \( v_r \) to every process whose ID is bigger than \( r \) (Fig. 4, line 5). When a process \( p_j \) receives a value in this round, it will set \( v_j \) to that value (Fig. 4, lines 6, 7).

**Theorem 1.** The protocol described in Fig. 4 solves the Uniform Consensus problem in synchronous systems where up to \( t \) processes can crash.

**Proof.** Termination follows from the bounded number of rounds. Validity follows from the fact that in each round \( r \) just \( v_r \) is sent and each process makes decision on the value maintained at the end of round \( t+1 \).

Because at most \( t \) processes can crash, there is at least one round of the \( t+1 \) rounds in which the coordinator in that round does not crash. Assume \( r \) is the first round in which the coordinator \( p_r \) does not crash, all alive processes whose ID is bigger than \( r \) will set their value to \( v_r \) at the end of round \( r \) (Fig. 4, lines 5–7). Because all processes whose ID is smaller than \( r \) have crashed in this case, then after round \( r \), all alive processes maintain the same value. This ensures that all alive processes maintain the same value at the end of round \( t+1 \). Because a process can only decide at the end of round \( t+1 \) (Fig. 4, line 10), no two processes can decide differently, the Uniform Agreement property is assured. □

**Cost:** The time complexity is trivially \( t+1 \) rounds. During each round \( r \), the round coordinator sends \( n-r \) messages if the coordinator does not crash. Hence, the message complexity of the protocol is bounded by \( \sum_{i=1}^{t+1}(n-i) = (t+1)(n-t/2-1), O(t \times n) \). In non-rotating coordinator-based consensus protocols, every process sends a message to all other processes in the first round, then the message cost in the first round is at least \( (n-1) \times n \), which is more than \( \sum_{i=1}^{t+1}(n-i) \). Thus, compared to those protocols, the rotating coordinator protocol uses the minimum number of messages.

The shortcoming of the protocol is that it is not an early-stopping one.

### 3.4. Early stopping rotating coordinator protocol

Because the above rotating coordinator protocol always needs \( t+1 \) rounds to achieve consensus, we design an early stopping rotating coordinator consensus protocol. Fig. 5 presents the rotating coordinator based S-Protocol for S-Round with crash failures. Each round \( r (1 \leq r \leq t+1) \) is managed by a predetermined coordinator, \( p_r \). In each round \( r, p_r \) will send \( v_r \) to all processes whose ID is bigger than \( r \) (Fig. 5,
line 6). When a process \( p_j \) receives a value in this round, it will set \( v_j \) to that value (Fig. 5, line 8). In each round \( r \) of this protocol, if the previous coordinator \( p_{r-1} \) does not crash, it can deduce that all alive processes maintain the same value as \( v_{r-1} \). Then it can make an early decision as \( v_{r-1} \) and send the decision to every other process (Fig. 5, line 10). This principle is realized using a Boolean variable \( done \) in the protocol, which initially is set to \( false \) (Fig. 5, line 2). During each round, the coordinator will set \( done \) to \( true \) after it sends messages (Fig. 5, line 6). Then during the next round, it can decide and send that decision to all other processes. When a process \( p_i \) receives a decision message \( (D, v) \), it sets its value \( v_i \) to \( v \) and set \( done \) to \( true \) (Fig. 5, line 7), and it can decide and send its decision to all other processes during the next round also.

Unlike the orderly crash failure model, in S-Protocol with crash failure, when a rotating coordinator crashes in a round, no process can establish whether another process has received the message from the coordinator in the current round. So in this model, a process must send a message to all other processes for early stopping after it decides.

### 3.4.1. Correctness proof of S-Protocol

**Theorem 2.** S-Protocol solves the Uniform Consensus problem in synchronous systems where up to \( t \) processes can crash.

**Proof.** Termination follows from the bounded number of rounds. Because in the protocol only a value proposed by a process is sent and set as the new value of another process (Fig. 5, lines 6–8), and finally a process decides on its own value (Fig. 5, lines 10, 13), the Validity property is assumed.

Because at most \( t \) processes can crash, there is at least one round of the first \( t + 1 \) rounds in which the rotating coordinator in that round does not crash. Assume \( r \) is the first round in which the coordinator \( p_r \) does not crash, all alive processes whose ID is larger than \( r \) will set their value to \( v_r \) (Fig. 5, lines 6, 8). Because all processes whose ID is smaller than \( r \) have crashed in this case, then after round \( r \), all alive processes maintain the same value. According to the algorithm (Fig. 5, lines 2, 5, 6), any decision can be made after round \( r \), thus no two processes can decide differently. The Uniform Agreement property is assured.  

**Theorem 3.** Every process that decides will do so at a round \( r \leq \min(t + 1, f + 3) \).

**Proof.** Firstly, every process that decides will decide within \( t + 1 \) rounds which is bounded by the number of rounds (Fig. 5, lines 2–4). It is obvious that the theorem is true when \( f \geq t - 2 \).

Now consider \( f < t - 2 \). There is at least one round of the first \( f + 2 \) rounds in which the previous coordinator does not crash in current round, assume \( r \) is the first such round. Thus, in both round \( r \) and round \( r - 1 \), process \( p_{r-1} \) does not crash. During round \( r - 1 \), \( p_{r-1} \) set \( done \) to \( true \) after sending messages to others (Fig. 5, line 6). At round \( r \), \( p_{r-1} \) will decide and send the decision to all other processes (Fig. 5, line 10), then all alive processes will receive the message and set \( done \) to \( true \) at the end of that round (Fig. 5, line 7). At round \( r + 1 \), all those processes will decide and send their decision to all other processes (Fig. 5, line 10). It is clear there is no round \( r + 2 \), because all alive processes decide and stop in round \( r + 1 \). Here, \( r \leq f + 2 \), thus every process that decides will do so within \( f + 3 \) rounds.

Thus every process that decides will do so at a round \( r \leq \min(t + 1, f + 3) \).

**3.4.2. Cost analysis of S-Protocol**

**Time cost:** The time complexity is \( \min(t + 1, f + 3) \) rounds.

**Message cost:** The message complexity consists of two parts. The first part is that every process will send a message to all other processes for early stopping. This part is bounded by \( n(n - 1) \). Another part is that in each round the coordinator will send a message to other processes. This part is bounded by \( \sum_{i=1}^{\min(f + 3, t + 1)} (n - i) \). Then the total message complexity of this protocol is bounded by \( \sum_{i=1}^{\min(f + 3, t + 1)} (n - i) + n(n - 1) \), \( O(n^2) \).

In case of failure-free, S-Protocol will execute three rounds, the number of messages is \( (n - 1) + (n - 2) + n(n - 1) \), because in the first round only \( p_1 \) sends messages to others, and in the second round \( p_1 \) sends decision messages to others and \( p_2 \) sends \( n - 2 \) messages, and in the third round all processes just send decision messages.

### 3.5. Time and message efficient early stopping coordinator based protocol

Because the S-Protocol solves the consensus problem in S-Round with crash failures within \( \min(t + 1, f + 3) \) rounds, this section presents the TS-Protocol, a time and message efficient early stopping consensus protocol. The protocol is called TS-Protocol. Initially, \( t + 1 \) processes are chosen as Coordinating Processes to achieve uniform consensus. Only Coordinators can send messages in the protocol, other processes just wait for receiving messages and decide after they find that every Coordinating Process can decide.

### 3.5.1. The TS-Protocol

In this protocol, every process \( p_i \) has a unique identity \( i \). Without losing generality, we choose the first \( t + 1 \) processes, whose identities range from 1 to \( t + 1 \), as CPs, \( \prod_{CP} = \{ CP_i \mid 1 \leq i \leq t + 1 \} \). Other processes are called non-CPs, \( \prod_{non-CP} = \prod - \prod_{CP} \).

The TS-Protocol is illustrated in Fig. 6. It consists of two functions, Consensus() executed by CPs (Fig. 6a) and Consensus2() executed by non-CPs (Fig. 6b). Function Consensus() is adopted from Protocol_RAY02. Each \( CP_i \) invokes the function Consensus\((v_j)\) where \( v_j \) is the value it proposes. It stops with the invocation of the statement return() that provides the decided value. Compared to Protocol_RAY02, at each round, \( CP_i \) not only sends a message including New() to all other processes but also includes a Boolean variable \( done \) in the message which indicates if \( CP_i \) can decide at the end of the round (Fig. 6a, line 7). \( R_i(r) \), initiated as \( R_i(0) = \prod_{CP} - \{ p_i \} \), is a
set of CPs from which CP<sub>i</sub> receives a message during round r. If it is the same as \( R_i(r - 1) \) then to the knowledge of CP<sub>i</sub> there are no CP crashes during round r (Fig. 6a, line 16). In this case, we will prove CP<sub>i</sub> gets the full information on the votes initially proposed by all processes, which means no other CP can maintain any new vote (for each CP<sub>j</sub> (j \( \neq i \)), CP<sub>i</sub> maintains \( v \)). To ensure the uniform agreement property, it can decide at the end of the next round (Fig. 6a, line 10) because at that time every other CP may keep the same value set as it keeps. If a CP finds that the done value of another CP is true in a round, it will set its done value to “true” also (Fig. 6a, line 15).

In Consensus2(), each non-CP maintains a copy vector of each CP maintained in the previous round (Fig. 6b, lines 3, 8, 10–12). If it receives a message from a CP and the CP’s done is true (Fig. 6b, line 9), this means if the CP does not crash at end of this round, it must decide. So if it finds all messages received in this round indicating all senders can decide at the end of the round, the non-CP can decide as well (Fig. 6b, lines 14–16). Otherwise, at the end of round \( r + 1 \), the non-CP will decide also (Fig. 6b, lines 18, 19).

3.5.2. Correctness proof

We need to prove that the proposed TS-Protocol satisfies the three properties of uniform consensus.

**Lemma 1.** At the end of round \( r \), if a Coordinating Process CP<sub>i</sub> sets done to true, then it maintains full information.
Proof. We need to prove that there does not exist a pair \((v, k)\), where \((v, k) \in CP_j\) \((j \neq i)\) and \(CP_j\) is alive but \((v, k) \notin CP_i\). Firstly, \((v, k)\) is sent by \(CP_k\) initially (Fig. 6a, lines 2, 7). When being sent or re-sent by other \(CPs\), \((v, k)\) does not change (Fig. 6a, line 14). Thus, \((v, k) \notin CP_i\) means \(V_i[k] = 1\). According to the protocol, there are two cases where \(CP_i\) sets done to true at the end of round \(r\)—Case 1: \(R_i(r) = R_i(r - 1)\) (Fig. 6a, line 16) and Case 2: \(CP_i\) finds another \(CP_j\) has set done to true in round \(r - 1\) (Fig. 6a, line 15).

Case 1: \(R_i(r) = R_i(r - 1)\).

Assume the contrary, there exists a pair \((v, k) \in CP_j\) \((j \neq i)\) and \(CP_j\) does not crash but \((v, k) \notin CP_i\). Because \(R_i(r) = R_i(r - 1)\) at the end of round \(r\), \(CP_i\) must receive a message from \(CP_j\) (Fig. 6a, lines 7, 8). According to the assumption, \((v, k)\) cannot be included in the message, which means \((v, k)\) cannot be a new vote obtained by \(CP_j\) during round \(r - 1\). If \(CP_j\) knows \((v, k)\) before round \(r - 1\) (Fig. 6a, lines 9, 12–14). Here, if \(k = j\), \(CP_j\) knows \((v, k)\) at the beginning (Fig. 6a, line 2). Nevertheless, \(CP_j\) must have sent \((v, k)\) to \(CP_i\) before round \(r\) (Fig. 6a, line 7). It is a contradiction to \((v, k) \notin CP_i\).

The last case is \(CP_j\) gets a new vote \((v, k)\) in round \(r\) (Fig. 6a, line 14). There must exist a \(CP_j\), which sends \((v, k)\) to all other processes in round \(r\), it is clear that \(CP_i \in R_i(r - 1)\). Because \((v, k) \notin CP_i\), \(CP_j\) must receive the message and \(CP_i\) fails to receive it, which means \(CP_i \notin R_i(r)\). It is a contradiction to \(R_i(r) = R_i(r - 1)\).

Therefore \(CP_i\) maintains full information and sets done to true by the end of round \(r\). But \(CP_i\) cannot know if other \(CPs\) maintain the same information. Hence, \(CP_i\) proceeds to round \(r + 1\) (to send the new pairs), decides and stops at end of round \(r + 1\).

Case 2: \(CP_i\) finds another \(CP_j\) has set done to true in round \(r - 1\).

According to the protocol, \(CP_i\) must contain every pair \(CP_j\) contains (Fig. 6a, lines 9, 12–14). If \(CP_j\) set done to true in round \(r - 1\) at Fig. 6a, line 16, \(R_i(r - 1) = R_i(r - 2)\), as discussed in Case 1, \(CP_j\) must maintain full information. Thus \(CP_i\) maintains full information. Another case is \(CP_j\) also set done to true in round \(r - 1\) at Fig. 6a, line 15. In this case, recursively we can get a \(CP\) which set its done to true as in Case 1. As discussed in Case 1, the \(CP\) gets full information. Thus, we can recursively prove \(CP_i\) maintains full information.

Lemma 2. If two \(CPs\) set done to true at the end of the same round, they maintain the same information.

Proof. Assume \(CP_i\) and \(CP_j\) \((j \neq i)\) set done to true at the end of the same round. By Lemma 1, \(CP_i\) maintains full information, which means for each \((v, k) \in CP_j\), we have \((v, k) \in CP_j\). On the other hand, by Lemma 1, \(CP_j\) maintains full information, which means for each \((v, k) \in CP_j\), we have \((v, k) \in CP_j\). Thus, \(CP_i\) and \(CP_j\) maintain the same information.

Lemma 3. If \(CP_i\) decides at the end of round \(r\) \((r < t + 1)\), all other alive \(CPs\) in round \(r\) must maintain the same information as \(CP_i\).

Proof. By Lemma 1, since \(CP_i\) set done to true at the end of round \(r - 1\), it maintains full information. According to the protocol, all other \(CPs\) that do not decide in round \(r\) must get all information from \(CP_i\) at the end of round \(r\) (Fig. 6a, lines 9, 12–14), they will maintain the same information as \(CP_i\) and set done to true at the end of round \(r\).

Theorem 4. The protocol solves the Uniform Consensus problem in synchronous systems where up to \(t\) processes can crash.

Proof. The Validity and Termination properties are obviously true.

Now we show that the uniform agreement property is also achieved. We will show that any two processes maintain the same information before they decide, and thus they decide on the same value. The theorem is proved in two steps: in step 1, we prove all \(CPs\) make the same decision, in step 2, we prove any non-\(CP\) makes the same decision as a \(CP\).

Step 1: Any two \(CPs\) make the same decision, which is proved by contradiction. Assume two \(CPs\), \(CP_i\) and \(CP_j\) \((i \neq j)\), decide on different values.

Firstly assume they decide in the same round \(r\).

• Both \(CP_i\) and \(CP_j\) set done to true at the end of round \(r - 1\). By Lemma 2, both \(CP_i\) and \(CP_j\) maintain the same information. Because they use the same method to decide in round \(r\) (Fig. 6a, line 10), they cannot make different decisions—a contradiction.

• At least one of them does not set done to true at the end of round \(r - 1\). Because the two processes decide in round \(r\), we have \(r = t + 1\) (Fig. 6a, lines 18, 19). Without losing generality, assume \(CP_i\) does not set done to true at the end of round \(r - 1\). This means in each round \(k\) of the first \(t\) rounds, \(CP_i\) has \(R_i(k) \neq R_i(k - 1)\), which means in each round, a \(CP\) has crashed and \(CP_i\) does not receive the message sent from the crashed process. There are totally \(t + 1\) \(CPs\), so in round \(r\) there is only one alive \(CP\) left. But here we have \(CP_i\) and \(CP_j\) \((i \neq j)\) in round \(r\)—a contradiction.

Secondly, assume they decide in different rounds, \(CP_i\) decides in round \(r_1\) and \(CP_j\) decides in round \(r_2\). Without losing generality, assume \(r_1 < r_2\). It is clear that \(r_1 < t + 1\), by Lemma 3, when \(CP_i\) decides in round \(r_1\), \(CP_i\) and all other alive \(CPs\) will maintain the same information as \(CP_i\). They cannot make a different decision—a contradiction.

Step 2: Any non-\(CP\) makes the same decision as a \(CP\) does.

There are three cases in which a non-\(CP\) makes decision.

• The first case is that the non-\(CP\) finds all alive \(CPs\) can decide (Fig. 6b, lines 14–16). This means all alive \(CPs\) set their done to true in the previous round, by Lemma 2 all of them maintain the same information. According to the protocol, the non-\(CP\) must keep a copy vector of each alive \(CP\) (Fig. 6b, lines 8, 10, 11). The non-\(CP\) decides based on a copy vector of an alive \(CP\) (Fig. 6b, line 15). Because both processes use the same method to decide (Fig. 6a, line 10 and Fig. 6b, line 15), the non-\(CP\) makes the same decision as the \(CP\).
The second case is that the non-CP finds it does not receive any messages in the round (Fig. 6b, lines 12, 13). This means all alive CPs crash in this round and there must be a CP, $CP_1$, which decided and stopped before the current round (Otherwise, contrary to at most $t$ processes can crash). Thus, in the previous round, the non-CP must have received at least one message whose sender set its done to true. In this case, when the non-CP decides, it decides based on a copy vector of such a CP obtained in the previous round (Fig. 6b, lines 12 and 15). Thus the non-CP makes the same decision as a CP does.

The third case is the non-CP decides at the end of round $t + 1$. According to the protocol, in this round, the non-CP must have received a message from a CP, assume $CP_1$, which did not set its done to true in round $t$ (Fig. 6b, lines 7–11). This means that in each of the first $t$ rounds, there must exist one and just one CP crash in each of previous $t$ rounds, and $CP_1$ does not receive the message sent from the crashed CP. Because there is only $t + 1$ CPs, in round $t + 1$, only $CP_1$ left as a correct CP, and $R(t + 1)$ of the non-CP only has $CP_1$ (Fig. 6b, line 7) also. Thus, the non-CP makes the same decision as $CP_1$ (Fig. 6b, lines 18, 19).

As proved in Step 1, all CPs make the same decision. As proved in Step 2, every non-CP makes the same decision as a CP. Thus all processes will make the same decision.

Theorem 5. Every process that decides will do so by the end of a round $r$, where $r \leq \min(t + 1, f + 2)$.

Proof. There are two cases:

Case 1: $t + 1 \leq f + 2$.

According to the $t + 1$ rounds bound of protocols (Fig. 6a, lines 4–6 and Fig. 6b, lines 4–6), all processes must decide by the end of round $t + 1$. The theorem is true in this case.

Case 2: $t + 1 > f + 2$.

Assume the contrary, there is a process which needs at least $f + 3$ rounds to decide. Firstly, assume it is a CP, $CP_1$. Thus in round $f + 1$, $CP_1$ cannot set its done to true, otherwise, $CP_1$ can decide in round $f + 2$ (Fig. 6a, line 10). Because actually $f$ processes crashed, in each of the first $f$ rounds, there must be a CP crashed and $CP_1$ fails to receive the message sent from it, otherwise $CP_1$ will set done to true before round $f + 1$. Thus, there is no crash in round $f + 1$, $CP_1$ will have $R_t(f) = R_t(f + 1)$ and set done to true in round $f + 2$ (Fig. 6a, line 10)—a contradiction. The above proof indicates all alive CPs can set done to true by the end of round $f + 1$. Secondly, assume it is a non-CP which needs at least $f + 3$ rounds to decide. As discussed above, all alive CPs set done to true by the end of round $f + 1$. According to the protocol, the non-CP will decide and stop at Fig. 6b, line 15 within round $f + 2$ also—a contradiction. The theorem is true in this case also.

3.5.3. Cost of the protocol

According to the TS-Protocol, during each round, every CP sends a message to all other processes, and non-CPs just receive messages. The message complexity in each of these rounds is: $(t + 1) \times (n - 1)$. By Theorem 5, the protocol needs at most $\min(t + 1, f + 2)$ rounds, thus, the message complexity of our proposed protocol is

$$\min(t + 1, f + 2)(t + 1)(n - 1), \quad O(f \times t \times n).$$

Comparing to Protocol _RAY02_, non-CPs just receive messages from CPs in TS-Protocol. Thus, during each round the number of messages in TS-Protocol decreases by $(n - t - 1)(n - 1)$ comparing to Protocol _RAY02_.

4. Message efficient protocols for S-Round with orderly crash failures

In this section, we present the SO-Protocol, a revised version of the S-Protocol with orderly crash failures, which only needs $\min(t + 1, f + 2)$ rounds to achieve consensus.

4.1. SO-Protocol description

Fig. 7 presents the SO-Protocol for the S-Round model with orderly crash failures, which can tolerate up to $t$ faults ($t < n - 1$). In the first round, only the prescribed coordinator sends messages. After the first round, in each round $r$ ($1 < r \leq t + 1$), both the current coordinator and the previous coordinator send messages. Other processes just wait for messages from the round coordinators.

```plaintext
1 Function Consensus(v)
2 r = 0;
3 while r < t + 1 do
4 \quad r = r + 1;
5 \quad if (r = 1) then
6 \quad \quad if (i = r) then
7 \quad \quad \quad for j = r + 1 to n step 1 send (v) to p_j;
8 \quad \quad \quad endif
9 \quad \quad let v be the value received from p_r during the rth round;
10 \quad \quad if (i > t + 1) then // processes in Π_{non-CP}
11 \quad \quad \quad if receive the message from p_r then return (v) endif
12 \quad \quad \quad endif
13 \quad \quad else // r > 1
14 \quad \quad \quad if (i = r - 1) then
15 \quad \quad \quad \quad for j = r + 1 to t step -1 send (v) to p_j;
16 \quad \quad \quad \quad endif
17 \quad \quad \quad if (i = r) then
18 \quad \quad \quad \quad for j = r + 1 to n step 1 send (v) to p_j;
19 \quad \quad \quad \quad endif
20 \quad \quad \quad if (i = r - 1) then return (v) at the end of the round endif
21 \quad \quad \quad let v be the value received from p_{r-1} during the rth round;
22 \quad \quad \quad let v’ be the value received from p_r during the rth round;
23 \quad \quad \quad if (i \geq r AND i \leq t + 1) then // processes in Π_{CP}
24 \quad \quad \quad \quad if receive one message from p_{r-1} then return (v) endif
25 \quad \quad \quad \quad if receive one message sent by p_r then v’ = v’’ endif
26 \quad \quad \quad \quad if (r = t + 1) return (v); endif
27 \quad \quad \quad \quad else // processes in Π_{non-CP}
28 \quad \quad \quad \quad if receive the message from p_r then return (v)’ endif
29 \quad \quad \quad \quad endif
30 \quad \quad \quad \quad endif
31 \quad \quad \quad \quad end while
```

Fig. 7. Protocol for S-Round with orderly crash failures.
Processes in the protocol are divided into two sets, $\prod_{CP}$ which consists of the IDs of the rotating coordinator processes, and $\prod_{\text{non-CP}} = \prod - \prod_{CP}$ which consists of the IDs of non-coordinator processes. Because the protocol aims to tolerate $t$ process crashes and just consists of $t+1$ rounds, the size of $\prod_{CP}$ is $t+1$. For simplicity, we choose the first $t+1$ processes from $\prod$ to form $\prod_{CP}$. Thus $\prod_{CP} = \{p_i | 1 \leq i \leq t+1\}$. A round $r$ consists of the following steps:

1. The rotating coordinator sends round $r$ messages to other processes, and the previous coordinator sends round $r$ messages to other processes in $\prod_{CP}$ ($r > 1$).
2. Every process waits for round $r$ messages in the round.
3. After a process has received messages, it executes local computations.

During round $r$, the rotating coordinator $p_r$ will first send $v_r$ in ascending order to all processes IDs larger than $r$ (Fig. 7, lines 6–8 when $r = 1$ and lines 17–19 when $r > 1$). After the first round the previous coordinator $p_{r-1}$ sends $v_{r-1}$ in descending order to processes whose IDs ranges from $t+1$ to $r$ (Fig. 7, lines 14–16). The previous coordinator will decide on its own value at the end of the round (Fig. 7, line 20). When a process $p_j$ ($j \neq r-1$) in $\prod_{CP}$ receives a value from $p_{r-1}$ in round $r$, it decides on the value and stops immediately (Fig. 7, line 24).

Otherwise, if it receives a value from $p_r$ in round $r$, it will set $v_j$ to the received value (Fig. 7, line 9 when $r = 1$ and line 25 when $r > 1$). When a process $p_j$ in $\prod_{\text{non-CP}}$ receives a value from $p_1$ in the same round, it will decide on the received value and stop immediately (Fig. 7, lines 9–12 when $r = 1$ and line 28 when $r > 1$).

4.2. Correctness proof of SO-Protocol

Theorem 6. The proposed protocol solves the Uniform Consensus problem in the S-Round model with orderly crash failures, in which up to $t$ processes can crash, and all alive processes decide by the end of $\min(t+1, f+2)$ rounds (the round lower bound), where $t < n-1$, $f \leq t$, and $f$ is the number of failures that actually occur.

Proof. It is obvious that the revised protocol satisfies the Validity property, because only values of rotating coordinators are sent and set by receiving processes, and they decide on one of such values.

It is also obvious that the revised protocol satisfies the Termination property, which is ensured by the bounded rounds (Fig. 7, line 3). Now let us prove the round lower bound $\min(t+1, f+2)$ rounds.

Firstly, we consider $f = t$. The protocol will stop by the end of round $t+1$ according to the algorithm though the current coordinator crashed at the beginning of each round.

Now, we show that the round lower bound is achieved by the end of round $f+2$ where $f < t$.

Firstly, we established the following rules from the algorithm:

Rule 1: In a round, if the rotating coordinator does not crash, all alive processes in $\prod_{\text{non-CP}}$, which have not decided, can make the same decision in that round (Fig. 7, lines 9–12 when $r = 1$ and line 28 when $r > 1$).

Rule 2: In a round, if the previous rotating coordinator does not crash, all alive processes in $\prod_{CP}$, which have not decided, can make the same decision in that round (Fig. 7, line 24).

Because at most $t$ processes can crash, and actually $f$ processes crash, there is at least one round of the first $f+2$ rounds in which the previous coordinator does not crash. Assume $r$ is the first such round, where $r \leq f+2$. By Rule 1, all alive processes in $\prod_{\text{non-CP}}$ will decide in round $r - 1$ (here we have $r > 1$, because there is no previous coordinator in the first round). By Rule 2, all alive processes in $\prod_{CP}$ will decide in round $r$. Thus, the round lower bound, $\min(t+1, f+2)$ rounds, is achieved.

To prove that the Uniform Agreement property is achieved, we first prove Lemmas 4 and 5.

Lemma 4. If two processes decide in the same round, they make the same decision.

Proof. According to the protocol, a process in $\prod_{\text{non-CP}}$ decides on the value of the current rotating coordinator in a round if it can do so (Fig. 7, lines 9–12 when $r = 1$ and line 28 when $r > 1$), and a process in $\prod_{CP}$ decides on the value of the previous rotating coordinator in a round if it can do so (Fig. 7, line 24). It is clear that two processes in $\prod_{\text{non-CP}}$ decide on the same value in the same round and two processes in $\prod_{CP}$ do so. Now, assuming a process in $\prod_{CP}$ and another process in $\prod_{\text{non-CP}}$ decide in the same round, according to the protocol, this case only occurs in round $t+1$. In this case, the non-CP must receive a value from the CP and set its value to the receiving value, they must decide on the same value. Thus all processes that decide in the same round must make the same decision. \(\square\)

Lemma 5. If all alive processes in $\prod_{CP}$ maintain the same value $v$ at the end of a round, all processes which decide after that round will make the same decision on $v$.

Proof. The Lemma is obviously true. During the following rounds a process decides on the value of the corresponding current coordinator and previous coordinator. Because the values of the alive processes in $\prod_{CP}$ are the same, all processes that decide after the round will make the same decision on $v$. \(\square\)

Now, we prove the uniform agreement property, assuming by contradiction that two processes $p_i$ and $p_j$ ($i \neq j$) make different decisions. By Lemma 4, they cannot decide in the same round. Without losing generality, assume that $p_i$ decides in round $r$ and $p_j$ decides in round $r'$, and $r < r'$. There are three possible cases:

(a) $p_i$ is the previous rotating coordinator $p_{r-1}$ ($i = r-1$). According to the protocol (Fig. 7, line 20), by Rule 1, all alive processes in $\prod_{\text{non-CP}}$ have decided in the previous round (Fig. 7, lines 9–12 when $r = 1$ and line 28 when $r > 1$) and by Rule 2, all alive processes in $\prod_{CP}$ will decide and
stop in round $r$ (Fig. 7, line 24). So $p_j$ cannot decide in round $r'\prime$—a contradiction.

(b) $p_i$ is the current rotating coordinator $p_r$ ($i = r$). According to the protocol, $p_i$ must receive a message from the previous coordinator (Fig. 7, line 24) or ($r = t + 1$). In the first sub-case, by Rule 1, all alive processes in $\prod_{\text{non}-CP}$ must have decided in the previous round. According to the protocol, all other alive processes in $\prod_{CP}$ maintain the same value (Fig. 7, line 9 when $r = 1$ and line 25 when $r > 1$). By Lemma 5, $p_j$ cannot make a different decision with $p_i$—a contradiction. In the second sub-case, $p_j$ cannot decide in a round $r' > t + 1$—a contradiction.

(c) $p_i$ is not the rotating coordinator $p_r$ or the previous rotating coordinator $p_{r-1}$. According to the protocol and the property of the orderly crash model, if $p_i$ is in $\prod_{\text{non}-CP}$, when $p_i$ decides on the value of $p_r$ in round $r$, all alive processes in $\prod_{CP}$ must have received the message sent from $p_r$ and set their values to the value of $p_r$. So their values are the same. By Lemma 5, all processes which decide after round $r$ will make the same decision on the value of $p_r$. Thus $p_j$ cannot make a different decision with $p_i$—a contradiction. Otherwise, if $p_i$ is in $\prod_{CP}$, this means the previous coordinator does not crash, thus all alive processes in $\prod_{CP}$ maintain the same value as the previous coordinator. By Lemma 5, $p_j$ cannot make a different decision with $p_i$—a contradiction.

So, any two processes make the same decision. Thus, Theorem 4 to 6 true because all three properties of the uniform consensus are satisfied. □

4.3. Cost analysis of SO-Protocol

**Time cost:** The time complexity is min$(t + 1, f + 2)$ rounds.

**Message cost:** In case of failure-free, SO-Protocol will execute two rounds, the number of messages is $(n-1)+(t+n-2)$, because in the first round only $p_1$ sends messages to others, and in the second round $p_1$ sends $t$ messages and $p_2$ sends $n-2$ messages. The message complexity is bounded by $(n - 1) + \sum_{i=2}^{\min(f+2, t+1)}(n + t + 1 - 2i)$, as $O(f \times n)$. In the worst case, after the first round the previous coordinator crashes at the beginning of each round. Then the number of total messages is $\sum_{i=1}^{f+1}(n - i)$.

5. Message efficient protocols for M-Round with orderly crash failures

In this section, we present the MO-Protocol, a revised version of the SO-Protocol to work in the M-Round model with orderly crash failures, which requires only $f + 1$ rounds to achieve consensus.

5.1. Protocol description

Fig. 8 presents our early stopping uniform consensus protocol, MO-Protocol, for the M-Round model with orderly crash failures, which can tolerate up to $t$ faults ($t < n - 1$). Each process $p_i$ is assigned a unique identity $i$ ($1 \leq i \leq n$). Each process $p_i$ invokes the function Consensus ($v_i$), where $v_i$ is the value it proposes. It stops with the invocation of the return() statement that provides the decided value.

Consensus() is made up of $t + 1$ rounds. Each round $r$ ($1 \leq r \leq t + 1$) is managed by a predetermined coordinator $p_r$. Only the coordinator can send messages in a round. Therefore, a round $r$ consists of the following steps:

1. The rotating coordinator for this round sends round $r$ messages to other processes.
2. Every process waits for round $r$ messages from the rotating coordinator in that round.
3. After a process has received messages, it executes local computations.

Thus, all processes in the protocol are divided into two sets, $\prod_{CP}$ which consists of the IDs of the rotating coordinator processes, and $\prod_{\text{non}-CP} = \prod - \prod_{CP}$ which consists of the IDs of non-coordinator processes. Because the protocol aims to tolerate $t$ process crashes and just consists of $t + 1$ rounds, the size of $\prod_{CP}$ is $t + 1$. For simplicity, we choose the first $t + 1$ processes from $\prod$ to form $\prod_{CP}$. Thus $\prod_{CP} = \{p_i|1 \leq i \leq t + 1\}$.

During round $r$, the rotating coordinator $p_r$ will first send $v_r$ in ascending order to all processes whose IDs are larger than $r$ (Fig. 8, line 6), and then send $v_r$ in descending order to processes whose IDs ranges from $t + 1$ to $r + 1$ (Fig. 8, line 7). The coordinator will decide on its own value at the end of the round (Fig. 8, line 9). When a process $p_j$ ($j \neq r$) in $\prod_{CP}$ receives a value from $p_r$ in round $r$, it will set $v_j$ to the received value (Fig. 8, line 12); if it receives the value twice, it decides on the value and stops immediately (Fig. 8, line 13). When a process $p_j$ in $\prod_{\text{non}-CP}$ receives a value from $p_r$ in the same round, it will decide on the received value and stop immediately (Fig. 8, line 15).

5.2. Correctness proof

The correctness proof of the MO-Protocol is similar to that of SO-Protocol.
Theorem 7. The proposed protocol solves the Uniform Consensus problem in the M-Round model, in which up to \( t \) processes can crash, and all alive processes decide by the end of \((f + 1)\) rounds, where \( t < n - 1 \), \( f \leq t \), and \( f \) is the number of failures that actually occur.

Proof. It is obvious that the proposed protocol satisfies the Validity property.

To show that the Termination property is achieved, we first prove Lemma 6.

Lemma 6. If the current rotating coordinator does not crash in a round, all alive processes, which have not decided, can make the same decision in that round.

Proof. Assume the current rotating coordinator \( p_r \) is the first coordinator that does not crash in its round. According to the protocol, all alive processes in \( \prod_{CP} \) that have not decided can receive a message from \( p_r \) in the round, and will decide on the value \( v_r \) maintained by \( p_r \) (Fig. 8, line 15). All alive processes in \( \prod_{CP} \), except \( p_r \), that have not decided can receive two messages from \( p_r \) in the round, and will decide on \( v_r \) (Fig. 8, line 13). \( p_r \) decides on its value, \( v_r \), at the end of the round.

Because at most \( t \) processes can crash and actually \( f \) processes crash, there is at least one of the first \( f + 1 \) rounds in which the corresponding coordinator does not crash. Assume \( r \) is the first round in which the coordinator \( p_r \) does not crash and \( r \) must be not more than \( f + 1 \). By Lemma 6, all alive processes will decide in the round. Thus, the Termination property is achieved.

To prove that the Uniform Agreement property is achieved, we first prove Lemmas 7 and 8.

Lemma 7. If two processes decide in the same round, they make the same decision.

Proof. According to the protocol, a process decides on the value of the current rotating coordinator in a round if it can do so. Thus all processes that decide in the same round must make the same decision.

Lemma 8. If all alive processes in \( \prod_{CP} \) maintain the same value \( v \) at the end of a round, all processes which decide after that round will make the same decision on \( v \).

Proof. The lemma is obviously true. During the following rounds a process decides on the value of the corresponding rotating coordinator. Because the values of the alive processes in \( \prod_{CP} \) are the same, all processes that decide after the round will make the same decision on \( v \).

We prove the uniform agreement property by contradiction. Assume two processes \( p_i \) and \( p_j \) \((i \neq j)\) make different decisions. By Lemma 7, they cannot decide in the same round. Without losing generality, assume that \( p_i \) decides in round \( r \) and \( p_j \) decides in round \( r' \), and \( r < r' \). There are two possible cases:

(a) \( p_i \) is the rotating coordinator \( p_r \) \((i = r)\), by Lemma 6, all alive processes will decide and stop in round \( r \). So \( p_j \) cannot decide in round \( r' \),—a contradiction.

(b) \( p_i \) is not the rotating coordinator \( p_r \). According to the protocol and the property of the new model, when \( p_i \) decides on the value of \( p_r \) in round \( r \), all alive processes in \( \prod_{CP} \) must have received at least one message from \( p_r \) and set their values to the value of \( p_r \). So their values are the same. By Lemma 8, all processes which decide after round \( r \) will make the same decision on the value of \( p_r \). Thus \( p_j \) cannot make a different decision with \( p_i \),—a contradiction.

So, any two processes make the same decision. Thus, Theorem 6 to 7 true because all three properties of the uniform consensus are satisfied.

5.3. Cost analysis

Time cost: The time complexity is \( f + 1 \) rounds.

Message cost: If \( p_1 \) does not crash in the first round, the protocol can stop in the first round, and the number of messages is \((n + t - 1)\). In the failure-free case, the number of messages is also \((n + t - 1)\).

The message complexity is bounded by \( \sum_{i=1}^{f+1} ((n - i) + (t + 1 - i)) \), as \( O(f \times n) \). In the worst case, in each round the coordinator fails to send the second message to any process in \( \prod_{CP} \). The total number of messages is \( \sum_{i=1}^{f+1} (n - i) \), as in the worst case of the SO-Protocol.

6. Comparison and summary of results

In this paper, we have proposed five coordinator based message efficient consensus protocols. Table 3 shows the comparison of these protocols in the SC model.

Firstly, it is clear that coordinator based protocols are message efficient compared with non-coordinator based protocols, because only coordinators send messages in coordinator based protocols. We can see the rotating coordinator protocol, S-Protocol and TS-Protocol are more message efficient than flood-set protocol and Protocol_RAY02.

Secondly, the TS-protocol is less message-efficient than other coordinator based protocols, because it uses \((t + 1)\) coordinators in each round, but the rotating coordinator protocol and
S-Protocol only have one rotating coordinator in each round. To achieve more time efficient, we have to increase the message complexity.

Table 4 shows the comparison of time and message complexity of consensus protocol which use rotating coordinator paradigm. It is clear that the MO-Protocol is the most time and message efficient. The MO-Protocol differs from the SO-Protocol in that in each round, the prescribed coordinator sends two messages to a process in $\prod_{i \in P} n$, but in the SO-Protocol, for each coordinator, all second messages sent to $\prod_{i \in P}$ are delayed to be sent in next round. Thus, in most cases, the SO-Protocol need one more round to achieve consensus, and requires $n - i$ more messages to achieve consensus than the MO-Protocol, where $i \leq f + 2$.

Lastly, when compared with the MO-Protocol and the SO-Protocol, the S-Protocol needs more messages to achieve consensus, the number of extra messages is bounded by $n(n - 1)$, because each process requires to send a message to all others after decided in the S-Protocol. The most message efficient rotating coordinator based consensus protocol is the MO-Protocol, and the worst message efficient one is the S-Protocol.

### 7. Conclusion

Consensus protocols so far exploit time efficiency using early stopping methods or improve message efficiency using the rotating coordinator paradigm. In this paper, we propose and evaluate various consensus protocols for synchronous distributed systems that achieve both time and message efficiencies. In synchronous distributed systems with crash failures, we introduce the rotating coordinator based message efficient consensus protocols called the S-Protocol and the TS-Protocol that still achieve the time lower bound of $min(t + 1, f + 2)$. Synchronous distributed systems are further divided into the S-Round model and the M-Round model where multiple messages can be sent to another process in a round. By restricting the crash failure model to orderly crash failures, we propose two new classes of rotating coordinator based message efficient consensus protocols namely SO-Protocol and MO-Protocol. Our analysis shows that the MO-Protocol is the most message efficient and time efficient when considering only the number of rounds. Future work includes investigating message and time efficient consensus protocols in asynchronous distributed systems with various failure models.

### Acknowledgment

This work is partially supported by the National University of Singapore, under Academic Research Fund R-252-000-180-112.

### References

Xianbing Wang received the B.Sc. in computer science from Wuhan University, China in 1992, the M.Eng. from Wuhan University of Hydraulic and Electric Engineering, China in 1997, and the Ph.D. from Wuhan University, China in 2001, both in computer application. He is currently a lecturer in the College of Computer Science and Technology at Wuhan University, China. From 2001 to 2003, he was a research associate at the Hong Kong Polytechnic University. From 2003 to 2006, he held a postdoctoral research fellow position at the National University of Singapore. His current research interests include parallel and distributed computing, fault tolerance, mobile computing, and mobile sensor networks.

Yong Meng Teo received his B.Tech. first class honours degree in computer science from the University of Bradford UK in 1983. He obtained his M.Sc. and Ph.D. degrees in computer science from the University of Manchester in UK, in 1987 and 1989, respectively. He is currently an associate professor in the Department of Computer Science at the National University of Singapore, and an associate senior scientist in the Asia-Pacific Science & Technology Centre at Sun Microsystems Inc. He was a fellow of the Singapore-Massachusetts Institute of Technology Alliance from 2002 to 2006. His research interests include parallel and distributed computing, grid computing, fault tolerance, parallel simulation and performance analysis.

Jiannong Cao received the B.Sc. degree in computer science from Nanjing University, Nanjing, China in 1982, and the M.Sc. and the Ph.D degrees in computer science from Washington State University, Pullman, WA, USA, in 1986 and 1990, respectively. He is currently a professor in the Department of Computing at Hong Kong Polytechnic University, Hung Hom, Hong Kong. He is also the director of the Internet and Mobile Computing Lab in the department. Before joining Hong Kong Polytechnic University, he was on the faculty of computer science at James Cook University and University of Adelaide in Australia, and City University of Hong Kong. His research interests include parallel and distributed computing, networking, mobile and wireless computing, fault tolerance, and distributed software architecture. He has published over 200 technical papers in the above areas. Dr. Cao is a senior member of China Computer Federation, a senior member of the IEEE Computer Society, the IEEE Communication Society, IEEE, and ACM. He is also a member of the IEEE Technical Committee on Distributed Processing, IEEE Technical Committee on Parallel Processing, IEEE Technical Committee on Fault Tolerant Computing. He has served as a member of editorial boards of several international journals, a reviewer for international journals/conference proceedings, and also as a chair and member of organizing/program committees for many international conferences.