Consensus Based on Strong Failure Detectors: A Time and Message-Efficient Protocol

Fabíola GREVE[†] Michel HURFIN[†] Raimundo MACÊDO[‡] Michel RAYNAL[†]

[†] IRISA - Campus de Beaulieu, 35042 Rennes Cedex, France,
‡ LaSiD-CPD-UFBA, Campus de Ondina, CEP 40170-110 Bahia, Brazil
{fgreve|hurfin|raynal}@irisa.fr macedo@ufba.br

Abstract. The class of *strong* failure detectors (denoted $\mathcal{S}$) includes all failure detectors that suspect all crashed processes and that do not suspect some (a priori unknown) process that never crashes. So, a failure detector that belongs to $\mathcal{S}$ is intrinsically unreliable as it can arbitrarily suspect correct processes. Several $\mathcal{S}$-based consensus protocols have been designed. Some of them systematically require n computation rounds (n being the number of processes), each round involving n^2 or n messages. Others allow early decision (i.e., the number of rounds depends on the maximal number of crashes when there are no erroneous suspicions) but require each round to involve n^2 messages.
This paper presents an early deciding $\mathcal{S}$-based consensus protocol each round of which involves $3(n-1)$ messages. So, the proposed protocol is particularly time and message-efficient.

Keywords: Asynchronous Distributed System, Consensus, Crash Failure, Perpetual Accuracy Property, Unreliable Failure Detector.

1 Introduction

Several crucial practical problems (such as *atomic broadcast* and *atomic commit*) encountered in the design of reliable applications built on top of unreliable asynchronous distributed systems, actually belong to a same family: the family of *agreement problems*. This family can be characterized by a single problem, namely the *Consensus* problem, that is their "*greatest common subproblem*". That is why the consensus problem is considered as a fundamental problem. This is practically and theoretically very important. From a practical point of view, this means that any solution to consensus can be used as a building block on top of which solutions to particular agreement problems can be designed. From a theoretical point of view, this means that an agreement problem cannot be solved in systems where consensus cannot be solved.

Informally, the consensus problem can be defined in the following way. Each process proposes a value and all correct processes have to decide the same value, which has to be one of the proposed values. Solving the consensus problem in asynchronous distributed systems where processes may crash is far from being a trivial task. It has been shown by Fischer, Lynch and Paterson [3] that there is no deterministic solution to the consensus problem in those systems as soon as processes (even only one) may crash. This impossibility result comes from the fact that, due to the uncertainty created by asynchrony and failures, it is impossible to precisely know the system state. So, to be able to solve agreement problems in asynchronous distributed systems, those systems have to be "augmented" with additional assumptions that make consensus solvable in such improved systems. A major and determining advance in this direction has been done by Chandra and Toueg who have proposed [1] (and investigated with Hadzilacos [2]) the *Unreliable Failure Detector* concept.

A failure detector can informally be seen as a set of *oracles*, one per process. The failure detector module (oracle) associated with a process provides it with a list of processes it guesses to have crashed. A failure detector can make mistakes by not suspecting a crashed process, or by erroneously suspecting a correct process. In their seminal paper [1], Chandra and Toueg have defined two types of property to characterize classes of failure detectors. A class is defined by a *Completeness* property and an *Accuracy* property. A completeness property is on the actual detection of crashes. The completeness property we are interested in basically states that "every crashed process is eventually suspected by every correct process". An accuracy property limits the mistakes a failure detector can make.

In this paper, we are interested in solving the consensus problem in asynchronous distributed systems equipped with a failure detector of the class $\mathcal{S}$. A failure detector of this class suspects all crashed processes (completeness) and guarantees that there is a correct process that is never suspected, but this process is not a priori known (perpetual weak accuracy). Several $\mathcal{S}$-based consensus protocols have been proposed. They all assume $f \leq n-1$ (where f is the maximal number of processes that may crash), and consequently are optimal with respect to the number of crash failures they can tolerate. They all proceed in asynchronous "rounds".

The $\mathcal{S}$-based consensus protocol proposed in [1] requires exactly n rounds, each round involving n^2 messages (each message being composed of n values). The $\mathcal{S}$-based protocols presented in [7, 8] also require n rounds, but each round involves only n messages carrying a single value. It is important to emphasize that these three protocols require n rounds whatever the value of f, the number of actual crashes and the occurrences of erroneous suspicions are.

To our knowledge, very few *early deciding* $\mathcal{S}$-based consensus protocols have been proposed, more precisely, we are only aware of the generic protocol presented in [6][1]. When instantiated with a failure detector $\in \mathcal{S}$, this generic protocol

[1] The generic dimension of the protocol introduced in [6] lies in the class of the failure detector it relies on. This generic protocol can be instantiated with any failure detector of $\mathcal{S}$ (provided $f \leq n-1$) or $\diamond\mathcal{S}$ (provided $f < n/2$). A failure detector that belongs to $\diamond\mathcal{S}$: (1) eventually suspects permanently all crashes processes, and (2) guarantees that there is a time after which there is a correct process that is never suspected.

provides a $\mathcal{S}$-based consensus protocol that terminates in at most $(f+1)$ rounds when there are no erroneous suspicions. So, when the failure detector is tuned to very seldom make mistakes, this protocol provides early decision. Each round of this protocol involves n^2 messages (each message being made of a proposed value plus a round number) and one or two communication steps.

This paper presents an *early deciding* $\mathcal{S}$-based consensus protocol. When there are no erroneous suspicions, the proposed protocol requires $(f+1)$ rounds, in the worst case. When there are neither crashes nor erroneous suspicions, it requires a single round, a round being made up of two communication steps. Each round involves $3(n-1)$ messages, and each message carries at most three values: a round number, a proposed value and a timestamp (i.e., another round number). So, the protocol is both time and message-efficient. Moreover, a generalization of the protocol exhibits an interesting tradeoff between the number of rounds and the number of messages per round.

The paper is made up of five sections. Section 2 introduces the asynchronous system model, the class $\mathcal{S}$ of failure detectors, and the consensus problem. Then, Section 3 presents the $\mathcal{S}$-based consensus protocol. Section 4 discusses its cost. Finally, Section 5 concludes the paper.

2 Asynchronous Distributed Systems, Failure Detectors and the Consensus Problem

The system model is patterned after the one described in [1,3]. A formal introduction to failure detectors is provided in [1].

2.1 Asynchronous Distributed System with Process Crash Failures

We consider a system consisting of a finite set Π of $n > 1$ processes, namely, $\Pi = \{p_1, p_2, \ldots, p_n\}$. A process can fail by *crashing*, (i.e., by prematurely halting). It behaves correctly (i.e., according to its specification) until it (possibly) crashes. By definition, a *correct* process is a process that does not crash. Let f denote the maximum number of processes that can crash ($f \leq n-1$).

Processes communicate and synchronize by sending and receiving messages through channels. Every pair of processes is connected by a channel. Channels are not required to be FIFO, they may also duplicate messages. They are only assumed to be reliable in the following sense: they do not create, alter or lose messages. This means that a message sent by a process p_i to a process p_j is assumed to be eventually received by p_j, if p_j is correct[2].

The multiplicity of processes and the message-passing communication make the system *distributed*. There is no assumption about the relative speed of pro-

[2] The "no message loss" assumption is required to ensure the Termination property of the protocol. The "no creation and no alteration" assumptions are required to ensure its Validity and Agreement properties.

cesses or the message transfer delays. This absence of timing assumptions makes the distributed system *asynchronous*.

2.2 The Class $\mathcal{S}$ of Unreliable Failure Detectors

Informally, a failure detector consists of a set of modules, each attached to a process: the module attached to p_i maintains a set (named $suspected_i$) of processes it currently suspects to have crashed. Any failure detector module is inherently unreliable: it can make mistakes by not suspecting a crashed process or by erroneously suspecting a correct one. Moreover, suspicions are not necessarily stable: a process p_j can be added to and removed from a set $suspected_i$ according to whether p_i's failure detector module currently suspects p_j or not. As in [1], we say "process p_i suspects process p_j" at some time t, if at time t we have $j \in suspected_i$.

As indicated in the introduction, a failure detector class is defined by two abstract properties, namely a *Completeness* property and an *Accuracy* property. In this paper we are interested in the following properties [1]:

- Strong Completeness: Eventually, every crashed process is permanently suspected by every correct process.
- Perpetual Weak Accuracy: Some correct process is never suspected.

The failure detectors that satisfy these properties define the class $\mathcal{S}$ (*Strong* failure detectors). It is important to note that a failure detector $\in \mathcal{S}$ can make an arbitrary number of mistakes: at any time all (but one) correct processes can be erroneously suspected. Moreover, a process can alternatively suspect and not suspect some correct processes.

2.3 The Consensus Problem

In the consensus problem, every correct process p_i *proposes* a value v_i and all correct processes have to *decide* on some value v, in relation with the set of proposed values. More precisely, the *Consensus* problem is defined by the three following properties [1, 3]:

- Termination: Every correct process eventually decides on some value.
- Validity: If a process decides v, then v was proposed by some process.
- Agreement: No two correct processes decide differently.

The agreement property applies only to correct processes. So, it is possible that a process decides on a distinct value just before crashing. *Uniform Consensus* prevents such a possibility. It has the same Termination and Validity properties plus the following agreement property:

- Uniform Agreement: No two processes (correct or not) decide differently.

In the following we are interested in the *Uniform Consensus* problem.

3 The $\mathcal{S}$-Based Consensus Protocol

3.1 The Protocol

3.2 Underlying Principles

As other failure detector-based consensus protocols, the proposed protocol uses the *rotating coordinator* paradigm and processes proceed in asynchronous rounds [1]. There are at most n rounds. Each round r ($1 \leq r \leq n$) is managed by a predetermined coordinator, namely, p_r. Moreover, during r, the coordinator of the next round (namely, p_{r+1}) acts also a particular role. Each process p_i manages three local variables: the current round number (r_i), its current estimate of the decision value (est_i), and a timestamp (ts_i) that indicates the round number during which it adopted its current estimate est_i.

As in [6–8], during a round, the current coordinator tries to impose its current estimate as the decision value. To attain this goal, each round r is made of two steps (see Figure 1).

- During the first step (lines 4-6) the current coordinator p_r broadcasts a message carrying its current estimate, namely, the message PHASE1(r, est_r). When a process p_i receives such a PHASE1(r, v) message, it adopts v as its new estimate and consequently updates ts_i to the current round number (line 6). If p_i suspects p_r, its "state variables" est_i and ts_i keep their previous values.
- During the second phase (lines 7-13), each process sends its "current state" to the current round coordinator (p_r) and the next round coordinator (p_{r+1}). This "state" is carried by a PHASE2 message (line 8). The triple (r, est_i, ts_i) indicates that during r, (1) the estimate of the decision value considered by p_i is est_i and (2) this value has been adopted during the round ts_i.
  Then, p_r and p_{r+1} follow the same behavior: each waits until it has received PHASE2 messages from all the processes it does not suspect (let us note that due to the completeness property of the underlying failure detector, all crashed processes are eventually suspected).
  If all the PHASE2 messages the process p_r (resp. p_{r+1}) has received have a timestamp equal to the current round number (r), p_r (resp. p_{r+1}) decides on its current estimate (lines 11-12). This means that the current estimate of p_r has been imposed as decision value.
  Whether the process p_{r+1} decides during r or proceeds to $r+1$, it must have a correct estimate (in order not to violate the consensus agreement property). This is ensured by requiring it to update its local estimate (est_{r+1}) to the estimate it has received with the highest timestamp in a PHASE2(r, est, ts) message (line 10).

3.3 Structure

The protocol is fully described in Figure 1. A process p_i starts a consensus execution by invoking $\mathsf{Consensus}(v_i)$, where v_i is the value it proposes. The protocol terminates for p_i when it executes the statement *return* which provides it with the decided value (at line 12 or 15).

It is possible that distinct processes do not decide during the same round. To prevent a process from blocking forever (i.e., waiting for a value from a process that has already decided), a process that decides, uses a reliable broadcast [5] to disseminate its decision value. To this end, the Consensus function is made of two tasks, namely, $T1$ and $T2$. $T1$ implements the previous discussion. Line 12 and $T2$ implement the reliable broadcast.

```
Function Consensus($v_i$)

Task $T1$:
(1)  $r_i \leftarrow 0$; $est_i \leftarrow v_i$; $ts_i \leftarrow 0$;
(2)  while $r_i < n$ do
(3)      $r_i \leftarrow r_i + 1$;
         % $p_{r_i}$ is the coordinator of the current round %
         % $p_{r_i+1}$ is the coordinator of the next round %

         ———————— Phase 1 of round $r$: $p_{r_i}$ proposes to all ————————————
(4)      if $(i = r_i)$ then $\forall j$: send PHASE1$(r_i, est_i)$ to $p_j$ endif;
(5)      wait until (PHASE1$(r_i, v)$ has been received from $p_{r_i} \vee r_i \in suspected_i$);
(6)      if (PHASE1$(r_i, v)$ received from $p_{r_i}$) then $est_i \leftarrow v$; $ts_i \leftarrow r_i$ endif;

         ———————— Phase 2 of round $r$: each process replies to $p_{r_i}$ and $p_{r_i+1}$ ——
(7)      let $X = \{r_i, r_i + 1\}$ if $(r_i < n)$, $X = \{r_i\}$ otherwise;
(8)      $\forall j \in X$: send PHASE2$(r_i, est_i, ts_i)$ to $p_j$;
(9)      if $(i \in X)$ then wait until (PHASE2$(r_i, est, ts)$ messages have been
                                          received from all non suspected processes);
(10)                 if $i = r_i + 1$ then $est_i \leftarrow est$ rec. with highest $ts$ endif;
(11)                 if (all PHASE2 messages are such that $ts = r_i$) then
(12)                     $\forall j \neq i$ : send DECISION$(est_i)$ to $p_j$; return$(est_i)$ endif
(13)     endif
(14) endwhile

Task $T2$:
(15) upon the reception of DECISION$(est)$ from $p_k$:
                     $\forall j \neq i, k$ : send DECISION$(est)$to $p_j$; return$(est)$
```

Fig. 1. The $\mathcal{S}$-Based Consensus Protocol

3.4 Proof

The protocol satisfies the Termination, Validity, and Uniform Agreement properties defining the Consensus problem (these properties have been stated in Section 2.3). The reader interested in the proof will consult [4].

4 Cost of the Protocol

Time complexity The number of rounds of the protocol is $\leq n$. Differently from the $\mathcal{S}$-based consensus protocols described in [1, 7, 8] that always require n rounds, the actual number of rounds of the proposed protocol depends on failures occurrences and erroneous suspicions occurrences. So, to analyze the time complexity of the protocol, we consider the length of the sequence of messages (number of communication steps) exchanged during a round. Moreover, as we do not master the quality of service offered by the underlying failure detector, but as in practice failure detectors can be tuned to very seldom make mistakes, we do this analysis considering the underlying failure detector behaves reliably. In such a context, the time complexity of a round is characterized by a pair of integers [6]. Considering the most favorable scenario that allows to decide during the current round, the first integer measures its number of communication steps (without counting the cost of the reliable broadcast implemented by the task $T2$). The second integer considers the case where a decision cannot be obtained during the current round and measures the minimal number of communication steps required to progress to the next round. Let us consider these scenarios.

- The first scenario is when the current round coordinator is correct and is not suspected. In that case, 2 communication steps are required to decide. During the first step, the current coordinator broadcasts a PHASE1 message (line 4). During the second step, each process sends a PHASE2 message (line 8). So, in the most favorable scenario that allows to decide during the current round, the round is made up of two communication steps.
- The second scenario is when the current round coordinator has crashed and is suspected by all processes. In that case, as processes correctly suspect the coordinator (line 5), the first communication step is actually skipped. Processes only send PHASE2 message (line 8) and proceed to the next round. So, in the most favorable scenario to proceed to the next round, the round is made up of a single communication step.

So, when the underlying failure detector behaves reliably, according to the previous discussion, the time complexity of a round is characterized by the pair $(2, 1)$ of communication steps.

Message complexity of a round During each round, the round coordinator broadcasts a PHASE1 message and each process sends two PHASE2 messages. Hence, the message complexity of a round is bounded by $3(n-1)$.

Message type and size There are three types of message: PHASE1, PHASE2 and DECISION. A DECISION message carries only a proposed value. A PHASE1 message carries a proposed value plus a round number. A PHASE2 message carries a proposed value plus two round numbers. As the number of rounds is bounded by n, the size of the round number is bounded by $log_2(n)$.

Let $|v|$ be the bit size of a proposed value. According to the previous discussion, $3n(n-1)(|v| + log_2 n)$ is an upper bound of the bit complexity of the protocol.

5 Conclusion

The paper has studied the consensus problem in the setting of asynchronous distributed system equipped with a failure detector of the class $\mathcal{S}$. This class includes all the failure detectors that suspect all crashed processes and that do not suspect some (a priori unknown) correct process. The proposed protocol proceeds in asynchronous rounds and allows early decision. If there are neither failures nor false suspicions the decision is obtained in a single round. A round is made up of two communication steps and involves $3(n-1)$ messages.

The proposed protocol compares very favorably to the previous $\mathcal{S}$-based consensus protocols, as those require n rounds or n^2 messages per round.

References

1. Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, March 1996.
2. Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685–722, July 1996.
3. Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, April 1985.
4. Greve F., Hurfin M., Macêdo R. and Raynal M., Consensus Based on Strong Failure Detectors: Time and Message-Efficient Protocols. *Tech Report #1290*, IRISA, Université de Rennes, France, January 2000. http://www.irisa.fr/EXTERNE/bibli/pi/1290/1290.html.
5. Hadzilacos V. and Toueg S., Reliable Broadcast and Related Problems. In *Distributed Systems*, ACM Press (S. Mullender Ed.), New-York, pp. 97-145, 1993.
6. Mostéfaoui A. and Raynal M., Solving Consensus Using Chandra-Toueg's Unreliable Failure Detectors: a General Quorum-Based Approach. *Proc. 13th Int. Symposium on Distributed Computing (DISC'99) (formerly, WDAG)*, Springer-Verlag LNCS 1693, pp. 49-63, (P. Jayanti Ed.), Bratislava (Slovaquia), September 1999.
7. Mostéfaoui A. and Raynal M., Consensus Based on Failure Detectors with a Perpetual Weak Accuracy Property. *Proc. Int. Parallel and Distributed Processing Symposium (IPDPS'2k), (14th IPPS/11th SPDP)*, Cancun (Mexico), May 2000.
8. Yang J., Neiger G. and Gafni E., Structured Derivations of Consensus Algorithms for Failure Detectors. *Proc. 17th ACM Symposium on Principles of Distributed Computing*, Puerto Vallarta (Mexico), pp.297-308, 1998.