Decision Optimal Early-Stopping k-set Agreement in Synchronous Systems Prone to Send Omission Failures

Philippe RAÏPIN PARVÉDY Michel RAYNAL Corentin TRAVERS

IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France
{praipinp|raynal|ctravers}@irisa.fr

Abstract

The k-set agreement problem is a generalization of the consensus problem: each process proposes a value, and each non-faulty process has to decide a value such that a decided value is a proposed value, and no more than k different values are decided. This paper focuses on the k-set agreement problem in the context of synchronous systems where up to $t < n$ processes can experience crash or send omission failures (n being the total number of processes). The paper presents a k-set agreement protocol for this failure model (the first to our knowledge) which has two main outstanding features. (1) It provides the following early deciding and stopping property: no process decides or halts after the round $\min(\lfloor f/k \rfloor + 2, \lfloor t/k \rfloor + 1)$ where f is the number of actual crashes ($0 \leq f \leq t$). (2) It is decision-optimal. This new optimality criterion, suited to the omission failure model, concerns the number of processes that decide, namely, the protocol forces all the processes that do not crash to decide (regardless of whether they commit omission faults or not). It is noteworthy that each of these properties (early deciding/stopping vs decision-optimality) is not obtained at the detriment of the other. Last but not least, the protocol enjoys another first-class property, namely, simplicity.

Keywords*: Agreement problem, Crash failure, Early stopping, k-set agreement, Message-passing system, Round-based computation, Send omission failure, Synchronous system, Uniform consensus.*

1 Introduction

Context of the paper The k-set agreement problem generalizes the uniform consensus problem (that corresponds to the case $k = 1$). It has been introduced by Soma Chaudhuri to investigate how the number of choices (k) allowed to the processes is related to the maximum number (t) of processes that can crash [5]. The problem can be defined as follows. Each of the n processes (processors) defining the system starts with a value (called a "proposed" value). Each process that does not crash has to decide a value (termination), in such a way that a decided value is a proposed value (validity) and no more than k different values are decided (agreement)[1].

k-set agreement can be trivially solved in asynchronous systems when $k > t$. Differently, it has been shown that there is no solution in these systems as soon as $k \leq t$ [3, 15, 28]. (The asynchronous consensus impossibility, case $k = 1$, was demonstrated before, using different techniques [9][2].) Several approaches have been proposed to circumvent the impossibility to solve the k-set agreement problem in asynchronous systems (e.g., probabilistic protocols [20], or unreliable failure detectors with limited scope accuracy [14, 19]).

The situation is different in synchronous systems where the k-set agreement problem can always be solved, whatever the value of t with respect to k. It has also been shown that, in the worst case, the lower bound on the number of rounds (time complexity measured in communication steps) is $\lfloor t/k \rfloor + 1$ [6]. (This bound generalizes the $t + 1$ lower bound associated with the consensus problem [1, 2, 8, 18].)

Although failures do occur, they are rare in practice. For the uniform consensus problem ($k = 1$), this observation has motivated the design of early deciding synchronous protocols [4, 7, 17, 27], i.e., protocols that can cope with up to t process crashes, but decide in less than $t + 1$ rounds in favorable circumstances (when there are fewer failures).

[1] A process that decides and thereafter crashes is not allowed to decide one more value, in addition to the k allowed values. This is why k-set agreement generalizes *uniform consensus* where no two processes (be them faulty or not) can decide different values. Non-uniform consensus allows a faulty process to decide a value different from the value decided by the correct processes. The non-uniform version of the k-set agreement problem has not been investigated in the literature.

[2] The impossibility to solve consensus in asynchronous systems is usually named "FLP result" according to the names of its authors [9].

More precisely, these protocols allow the processes to decide in $\min(f+2, t+1)$ rounds, where f is the number of processes that crash during a run, $0 \leq f \leq t$, which has been shown to be optimal (the worst scenario being when there is exactly one crash per round) [4, 16][3].

In a very interesting way, it has very recently been shown that the early deciding lower bound for the k-set agreement problem in the synchronous crash failure model is $\lfloor f/k \rfloor + 2$ for $0 \leq \lfloor f/k \rfloor \leq \lfloor t/k \rfloor - 2$, and $\lfloor f/k \rfloor + 1$ otherwise [10]. This lower bound, not only generalizes the corresponding uniform consensus lower bound, but also shows an "inescapable tradeoff" among the number t of crashes tolerated, the number f of actual crashes, the degree k of coordination we want to achieve, and the best running time achievable [6]. As far as the time/coordination degree tradeoff is concerned, it is important to notice that, when compared to consensus, k-set agreement divides the running time by k (e.g., allowing two values to be decided halves the running time).

The consensus problem has been investigated in synchronous systems whose failure models are more severe than the crash failure model. More precisely, consensus can be solved in the Byzantine failure model (where a process can fail by exhibiting an arbitrary behavior) provided that $t < n/3$ [11, 22][4]. The general omission failure model [23] lies in between the crash failure model and the Byzantine failure model: a process can crash or fail by omitting to send or receive a message. An optimal early-stopping uniform consensus protocol for the general omission failure model is described in [24]. This protocol assumes $t < n/2$ which is a necessary requirement [21].

There are two types of early-deciding consensus protocols that have been designed for the omission failure model. In all cases a non-faulty process decides by round $\min(f+2, t+1)$, but some protocols (e.g., [23]) allow a faulty process that does not crash to execute more rounds (up to $t+1$), while in others protocols (e.g., [24]) no process executes more than $\min(f+2, t+1)$ rounds.

The consensus termination property concerns only the correct processes: they all have to decide. This requirement is tied to the problem, independently of a particular failure model. Due to the very nature of the corresponding faults, there is no way to force a faulty process to decide in the crash failure model, or in the Byzantine failure model. This could be different in the omission failure model, as this model prevents a faulty process that does not crash to behave arbitrarily. We could envisage that such processes be obliged to decide in some circumstances. But, none of the protocols that we are aware of for this failure model, forces a non-crashed faulty process to decide in some particular well-identified circumstances. Stated in another way, none of these protocols characterizes runs where they force non-crashed faulty processes to decide. Usually, as soon as they have identified a non-crashed faulty process, these protocols force it not to decide[5].

Content of the paper While not-early deciding k-set agreement protocols for the synchronous crash failure model (i.e., protocols that always terminate in $\lfloor t/k \rfloor + 1$ rounds) are now well understood [2, 6, 18], to our knowledge, so far only two early deciding k-set agreement protocols have been proposed [10, 25]. The protocol described in [10] assumes $t < n-k$, which means that (contrarily to what we could "normally" hope) the coordination degree k increases when the maximum number t of processes that can crash decreases. The protocol described in [25], which imposes no constraint on t, is based on a mechanism that allows the processes to take into account the actual pattern of crash failures and not only their number, thereby allowing the processes to decide in much less than $\lfloor f/k \rfloor + 2$ rounds in a lot of cases (the worst case being only when the crashes are evenly distributed in the rounds with k crashes per round).

This paper is on the k-set agreement problem in synchronous systems prone to process crashes and send omission failures (i.e., a process is faulty if it crashes or omits to send one or more messages). A protocol is proposed and proved correct that enjoys the following noteworthy properties[6]:

- *Resilience optimality*: the protocol requires only $t < n$ (at least one non-faulty process).
- *Early stopping*: no process executes more than $\min(\lfloor f/k \rfloor + 2, \lfloor t/k \rfloor + 1)$ rounds.
  So, differently from some consensus protocols, a faulty process that does not crash, does not penalize the termination time of the protocol as it does not execute more rounds than a correct process. When $\lfloor \frac{f}{k} \rfloor + 2 \leq \lfloor \frac{t}{k} \rfloor$, the protocol extends consequently the $\lfloor \frac{f}{k} \rfloor + 2$ lower bound for a correct process to decide (1) to the send omission failure model, and (2) to the processes that commit only send omission failures.
- *Decision optimality*: this new criterion in on the faulty processes that do not crash and are required to decide.

[3] More precisely, the lower bound is $f+2$ when $f \leq t-2$, and $f+1$ when $f = t-1$ or $f = t$.

[4] It is interesting to notice that, when it has been introduced for the first time (in 1980 by Pease, Shostak and Lamport [22]), the consensus problem was considered in synchronous systems prone to Byzantine process failures.

[5] More precisely, these faulty processes are unilaterally forced to "decide" a default value $\perp$, whose meaning is "no decision" [27].

[6] To our knowledge, this protocol is also the first k-set agreement protocol designed so far for the send omission failure model.

The protocol is optimal with respect to this criterion as it forces all the processes that do not crash to decide (be them faulty or not). We think that this is an important property (which, as noticed previously, is not met by the uniform consensus protocols designed so far for the send -or send/receive- omission failure models).

The design of a protocol that satisfies, simultaneously and despite process crashes and send omission faults, the agreement property of the k-set problem, the early stopping property and the decision optimality property, is not entirely obvious, as these properties are partly antagonistic. This is due to the fact that agreement requires that no more than k distinct values be decided (be the deciding processes correct or not), while early stopping requires the processes to stop as soon as possible after deciding. Consequently the protocol has not to prevent processes from deciding at different rounds, and so, after it has decided, a process can appear to the other processes as committing send omission failures, while it is actually correct. Finally, the decision optimality property prevents from eliminating from the protocol a faulty process as soon as it has been discovered as faulty, as it has to decide a value if it does not crash later. A major difficulty in the design of the protocol consists in obtaining simultaneously all these properties (agreement, early stopping for all processes, and decision optimality), and not each one at the price of not satisfying one of the two others.

When instantiated with $k = 1$, the protocol provides a new early stopping uniform consensus protocol for the send omission failure model, where all the processes that do not crash decide. To our knowledge, this is the first uniform consensus protocol that enjoys this property.

Last but not least, the proposed protocol enjoys another first class property, namely, design simplicity. Its design relies on (1) the existence of at least one correct process, and (2) the fact that each process that does not crash -be it faulty or not- continues executing rounds until it decides, and (3) the fact that a process that is informed it is faulty stops sending messages: it becomes passive in the sense that it forgets all its past, and from then on it only receives messages (at least from the correct processes), and these messages will help it decide correctly (this means that, when it discovers it is faulty, a faulty process executes a kind of "reset", and from then on is "fed" only with values from the processes it considers correct).

As already noticed, the main difficulty becomes from the fact that not all the processes decide during the same round, creating some uncertainty on which processes are correct. Nevertheless, the protocol succeeds in having the correct processes play a "pivot" role through which values converge and from which they are disseminated, thereby allowing agreement, early stopping and decision optimality to be met.

Roadmap The paper consists of 5 sections. Section 2 presents the computation model and gives a definition of the k-set agreement problem. Section 3 presents the protocol. Section 4 provides a formal statement of its properties (lemmas and theorems). The proofs of these properties are given in an appendix (interestingly, due to the specificity of send omission failures, the proof techniques used for some lemmas and theorems differ deeply from the ones used to prove the "corresponding properties" in the crash failure model). Finally Section 5 discusses the protocol and concludes the paper.

2 Computation Model and k-Set Agreement

2.1 Round-Based Synchronous System

The system model consists of a finite set of processes, namely, $\Pi = \{p_1, \ldots, p_n\}$, that communicate and synchronize by sending and receiving messages through channels. Every pair of processes is connected by a channel. The underlying communication system is assumed to be failure-free: there is no creation, alteration, loss or duplication of messages.

The system is *synchronous*. This means that each of its executions consists of a sequence of *rounds*. Those are identified by the successive integers $1, 2$, etc. For the processes, the current round number appears as a global variable r that they can only read, and whose progress is managed by the underlying system. A round is made up of three consecutive phases [2, 18, 27]. (1) A send phase in which each process sends messages. (2) A receive phase in which each process receives messages. (3) A computation phase during which each process processes the messages it received during that round and executes local computation. The fundamental property of the synchronous model lies in the fact that a message sent by a process p_i to a process p_j at round r, is received by p_j at the same round r.

2.2 Process Failure Model

A process is *faulty* during an execution if its behavior deviates from that prescribed by its algorithm, otherwise it is *correct*. A *failure model* defines how a faulty process can deviate from its algorithm [13]. We consider here the following types of faults.

- Crash failure. A faulty process stops its execution prematurely. After it has crashed, a process does nothing. Let us observe that if a process crashes in the middle of a sending phase, only a subset of the messages it was supposed to send might actually be sent.
- Send omission failure model. A faulty process crashes or omits sending messages it was supposed to send

[23]. Let us observe that a faulty process can omit to send messages to some processes during some round, and not during other rounds[7]. It can also crash after having committed send omission faults.

It is easy to see that these failure models are of increasing "severity" in the sense that any protocol that solves a problem in the send omission failure model, also solves it in the (less severe) crash failure model [13]. This follows from the fact that if a process crashes, it trivially commits omission failures in a permanent way after it has crashed.

As already indicated, n, t and f denote the total number of processes, the maximum number of processes that can be faulty, and the actual number of processes that commit failures during a run, respectively ($0 \leq f \leq t < n$); n and t are initially known by each process.

2.3 The k-Set Agreement Problem

The problem has been informally stated in the Introduction: every process p_i *proposes* a value v_i and each correct process has to *decide* on a value in relation to the set of proposed values. More precisely, the k-set agreement problem is defined by the following three properties:

- Termination: Every correct process decides.
- Validity: If a process decides v, then v was proposed by some process.
- Agreement: No more than k different values are decided.

As we can see 1-set agreement is the uniform consensus problem. In the following, we implicitly assume $k \leq t$. This is because k-set agreement can trivially be solved in synchronous or asynchronous systems when $t < k$ [5]. A one communication step protocol is as follows: (1) k processes are arbitrarily selected prior the execution; (2) each of these k processes sends its value to all processes; (3) a process decides the first value it receives.

3 An Optimal k-Set Agreement Protocol

This section presents a k-set agreement protocol that enjoys both early-stopping and decision optimality. Before presenting its design, let us first notice that transformations have been proposed that translate protocols designed for the crash failure model in corresponding protocols for the send omission failure model [12, 21]. These general transformations are irrelevant to our purpose for two reasons. The first reason lies in the fact that, as they are general, they have a cost (e.g., in the transformation described in [21], the send omission failure model requires two rounds to simulate each round of the crash failure model). The second issue is related to the decision optimality property. These transformation protocols track faulty processes and force the processes that have committed a send omission failure to unilaterally simulate a crash failure by returning a predefined default value (e.g., a value that cannot be proposed by a process, usually denoted $\perp$ and whose meaning is "no decision"). It follows that, inherently, these general transformations can guarantee neither time optimality with respect to the early stopping property, nor decision optimality with respect to the number of processes that decide. Actually, on one side attaining decision optimality, and on the other side forcing a process that has committed only send omission failures to crash are antagonistic.

This discussion shows that a k-set agreement protocol that enjoys both early stopping and decision optimality, cannot be obtained by simply "extending" a crash-tolerant protocol (this appears clearly in Section 4 where appropriate claims and properties are stated and proved).

3.1 Underlying Principles

The protocol uses the well-known "flooding strategy" to disseminate estimate values. Basically, it strives to partition the processes in two groups: the ones that appear as being correct and the ones that are faulty. The processes in the first group remain active (they send and receive messages), while the ones in the second group become passive (they only receive messages). This partitioning aims at preventing inconsistencies due to the fact that a faulty process can send messages to some processes while committing send omission with respect to others. After it knows it is faulty, a process p_i forgets its past as far as the computation of the decided value is concerned, resets the corresponding local variables, and, as from then on it receives messages only from processes it perceives as being correct, it will decide the same value as one of them.

3.2 Local Variables

The code for a process p_i is described in Figure 1. A process p_i starts participating in the k-set agreement protocol when it invokes k-SET_AGREEMENT(v_i) where v_i is the value it proposes. It terminates either when it crashes or when it executes the return (est_i) statement (line 11, 13 or 16) where est_i is the value it decides. As already announced, the protocol is round-based: r is the common shared variable that the processes (can only) read and that defines the progress of the whole computation.

In addition to est_i, each process p_i manages two local

[7] A send omission failure actually models a failure of the output buffer of a process. A buffer overflow is a typical example of such a failure.

set variables: $sender_i$ and can_dec_i. The meaning of these variables is the following.

- est_i is p_i's local estimate of the decision value. Initialized to v_i (line 1), it is then updated according to the estimate values received by p_i (line 8).
- When p_i starts a new round r, $sender_i$ contains the processes from which p_i is waiting for messages during that round. Its initial value is Π (the whole set of processes, line 1). It is then updated at line 6 to contain the processes that have not decided in a previous round and that p_i considers as correct. (These processes should normally either decide during the current round r or proceed to the round $r+1$ and send messages during $r+1$).
- can_dec_i is a set of processes p_j such that, to p_i's knowledge, p_j knows one of the k smallest values currently present in the system. Initialized to $\emptyset$ (line 1), this variable is then updated according to the messages received by p_i (line 8).

3.3 Process Behavior: Part 1

The behavior of a process p_i during a round r (lines 4-14) can be divided in two parts according to the management of its local variables, $sender_i$ on one side, and the pair (est_i, can_dec_i) on the other side. The first part (lines 4-6) is devoted to the management of the $sender_i$ variable.

If it is not considered as faulty ($i \in sender_i$), p_i sends to all the processes a message containing the current value of the three local variables ($sender_i$, est_i and can_dec_i) defining its local state (line 4). Then, p_i (be it faulty or not) waits for messages from each process p_j that it considers as being correct (i.e., from each p_j that it considers as a potential sender, namely such that $j \in sender_i$). Finally, according to the messages it has received, p_i computes the new value of $sender_i$ by intersecting its current $sender_i$ set, the subset of potential senders p_j from which it has received a message (rec_from set), and the $sender_j$ sets it has received from these processes p_j (line 6).

It follows from these statements that if a process p_y commits a send omission fault with respect to a correct process p_j during a round r, all the (non-crashed) processes p_i will suspect p_y during the round $r+1$, i.e., we will have $k \notin sender_i$ at the end of $r+1$. Differently, if p_y commits a send omission fault with respect to a process p_j during a round r, and p_j (that behaved correctly until r) commits a send omission fault during $r+1$, the fact that p_y be suspected depends on the actual pattern failure.

Let $sender_i[r]$ be the value of $sender_i$ after it has been updated at line 6 during the round r, $0 \leq r \leq \lfloor t/k \rfloor + 1$ (with $sender_i[0] = \Pi$). Line 6 provides the following monotonicity property: $sender_i[r] \subseteq sender_i[r-1]$, and gives $sender_i[r]$ the following meaning: it is the set of processes that have not decided during $r' < r$ and that p_i considers as being correct (according to the information it has received up to r).

3.4 Process Behavior: Part 2

The second part (lines 7-14) concerns the management of the est_i and can_dec_i variables. It is the crucial part of the protocol to ensure early stopping and decision optimality.

The normal case. Considering a process p_i, the "normal" case is when, after p_i has updated $sender_i$ at line 6 of r, we have $sender_i[r] \neq \emptyset$.

In that "normal" case, the process p_i first updates est_i and can_dec_i (line 8) by taking into account the messages it has received from the processes it currently considers as being correct (as defined by $sender_i[r]$). As the protocol requires that a process p_i decides the smallest value it has received from a process it considers correct, est_i is updated to the smallest est_j value that p_i has received from the processes in $sender_i[r]$.

Let us notice that $i \notin sender_i[r]$ means that p_i has been informed it is faulty. From now on, it does no longer consider its previous est_i value. Basically, when it discovers it is faulty, a process p_i proceeds to a "passive" mode where it does no longer send messages (line 4), and "resets" est_i only with the est_j values sent by the processes p_j it considers correct: a faulty process forgets its past. In that way, a decided value will always have been witnessed by correct processes.

Let us now focus on the local predicate evaluated at line 10, namely $(n - |sender_i[r]| < kr)$. This predicate is for early decision and stopping. As shown in the proof, this predicate means that p_i knows one of the k smallest values currently present in the system. To get a part of its underlying intuition, let us consider the simpler case where there are only crash failures. Let r be the first round during which the predicate is satisfied at p_i. We can then conclude that, among all the processes that were alive at the beginning of r, p_i has not received messages from at most $k-1$ of them [25, 27]. It consequently knows one of the k smallest values present in the system at round r. While this predicate is still correct for the send omission failure model, its correction proof is much more involved in this model than in the crash failure model as soon as we want to show that it does not prevent decision optimality. This difficulty comes from the fact that crash failures are both stable and global (if a process crashes at r, all the non-crashed processes suspect it at the latest at $r+1$), while send omission faults are nei-

```
Function k-SET_AGREEMENT(v_i)
(1)  est_i ← v_i; sender_i ← Π; can_dec_i ← ∅; % r = 0 %
(2)  for r = 1, ... , ⌊t/k⌋ + 1 do
(3)  begin_round
(4)    if (i ∈ sender_i) then foreach j ∈ Π do send (sender_i, est_i, can_dec_i) to p_j enddo endif;
(5)    let rec_from = {j : (sender_j, est_j, can_dec_j) is received from p_j during round r ∧ j ∈ sender_i};
(6)    sender_i ← sender_i ∩ rec_from ∩ ( ⋂_{j∈rec_from\{i}} sender_j );
(7)    if (sender_i ≠ ∅)
(8)      then can_dec_i ← ⋃_{j∈sender_i} can_dec_j; est_i ← min({est_j}_{j∈sender_i});
(9)           if (i ∈ sender_i ∧ i ∉ can_dec_i)
(10)            then if ((n − |sender_i| < k r) ∨ (can_dec_i ≠ ∅)) then can_dec_i ← can_dec_i ∪ {i} endif
(11)            else if (sender_i ⊆ can_dec_i) then return (est_i) endif
(12)          endif
(13)     else return (est_i) % est_i is the smallest value received at r − 1 %
(14)   endif;
(15) end_round;
(16) return (est_i)
```

Figure 1. k-set protocol for send omission, $1 \leq k$, $t < n$ (code for p_i)

ther stable (a process can commit a send omission during a round and not during the next round) nor global (during a round, p_y can commit send omission with respect to p_i and not with respect to p_j).

So, when $(n - |sender_i[r]| < kr)$, p_i indicates that it knows one of the k smallest values present in the system by adding its id to can_dec_i (line 10). If this predicate is false while $can\_dec_i \neq \emptyset$ (e.g., $j \in can\_dec_i$), p_i learns indirectly from p_j one of the k smallest values present in the system. It then adds its id to can_dec_i (notice that line 10 is the only line where a process adds its own id to this set).

The behavior of p_i then depends on the fact it considers itself as correct ($i \in sender_i$) and the value of its set can_dec_i. We consider three cases.

- $i \in sender_i \wedge i \notin can\_dec_i$ is satisfied (lines 9-10). $i \in sender_i[r]$ means that p_i perceives itself as being correct, while $i \notin can\_dec_i$ means that it does not know if est_i is one of the k smallest values in the system. Consequently, it checks if it is the case, either directly because $n - |sender_i[r]| < kr$ is satisfied, or indirectly because another process informed it ($can\_dec_i \neq \emptyset$).

- $i \in sender_i \wedge i \in can\_dec_i$ (lines 9 and 11). In that case, p_i perceives itself as being correct, and, at least from the previous round, it knows that est_i is one of the k smallest values. It decides est_i and stops if $sender_i \subseteq can\_dec_i$.

  As a correct process p_c that has not yet decided cannot be suspected, $sender_i \neq \emptyset \wedge sender_i \subseteq can\_dec_i$ means that p_i knows that the estimate value of each correct process that has not yet decided is among the k smallest values currently present in the system. As the estimate values of these correct processes are know by all the processes participating in the current round, this means that p_i can safely decide.

- $i \notin sender_i$ (lines 9 and 11). In that case, p_i knows it is faulty. It is in the "passive "mode, waiting for messages from the processes it perceives as being correct.

  Similarly to the previous case, as a correct process p_c (that has not yet decided) cannot be suspected, $sender_i \subseteq can\_dec_i$ means that p_i knows the estimate value of these correct processes and that these values are among the k smallest ones. It can consequently decides.

This terminates the discussion concerning the case of a process p_i such that $sender_i[r] \neq \emptyset$ after it has updated $sender_i$ during r.

A particular case. This case is when, after a process p_i has updated $sender_i$ at line 6 of r, we have $sender_i[r] = \emptyset$.

This case occurs in the particular failure pattern where (1) the correct processes decide during $r - 1$, (2) a process p_j behaved correctly until $r - 2$ and, during $r - 1$, commits send omission failure only with respect to the correct processes, and (3) p_j commits send omission failure with respect to the other faulty processes during r. This case can happen because the correct processes stop at $r - 1$ after having decided, and p_j is never informed it is faulty. When this occurs, the non-crashed faulty process p_i is such that $sender_i[r] = \emptyset$. It then decides the current value of est_i it has computed during the previous round.

4 Proof of the Protocol

This section shows that the k-set agreement protocol satisfies the validity, agreement and termination properties

defining the k-set agreement problem, plus the early deciding and stopping property, and the decision optimality property.

Notation The proof uses the following notations.

- $x_i[r]$ denotes the value of p_i's local variable x at the end of round r.
- $Participating[r]$ is the set of processes that execute round r. More precisely :
  $Participating[r] = \{p_i : p_i \text{ decides during round } r \text{ or proceeds to } r+1\}$.
- $EST[r] = \{est_i[r] : p_i \in Participating[r]\}$. By definition $EST[0]$ = the proposed values. ($EST[r]$ contains the values that are present in the system at the end of round r.)
- $Silent[r] = \{p_i : \forall p_j \in Participating[r] : i \notin sender_j[r]\}$.

  A process p_i that has crashed or has decided before the end of round $r-1$ is in $Silent[r]$. On the contrary, a process $p_i \in Silent[r]$ has not necessarily crashed or decided before the end of round r. For example, a process which does not crash but does not communicate with any process during round $r-1$ is in $Silent[r]$. It is important to remark that if $p_j \in Silent[r]$, then no process p_i (including p_j itself) takes into account est_j and can_dec_j sent by p_j (if any) to update its local variables est_i and can_dec_i at line 8 of the round r.

Basic properties The proof of the following inclusions are left to the reader:
- $Participating[r+1] \subseteq Participating[r]$,
- $Silent[r] \subseteq Silent[r+1]$,
- $\forall i \in Participating[r] : Silent[r] \subseteq \Pi - sender_i[r]$,
- $\forall i \in Participating[r+1]$:
  p_i is correct $\Rightarrow \Pi - sender_i[r] \subseteq Silent[r+1]$.

Lemmas and theorems Due to page limtation, the proof of the lemmas and theorems stated below cannot be presented here. The interested reader will find then in [26].

The first lemma establishes a monotonicity property on the values of the abstract variables $EST[r]$.

Lemma 1 $\forall r \geq 0 : EST[r+1] \subseteq EST[r]$.

The second lemma is fundamental for agreement and early decision and stopping. It relates an operational view (the number of messages received by a process p_i during a round r) with a semantic information (the fact that $est_i[r]$ is one of the k smallest values currently present in the system). Interestingly, this lemma uses a "witness" correct process.

Lemma 2 *Let us consider a round r, $1 \leq r \leq \lfloor \frac{t}{k} \rfloor + 1$, such that (1) there is a correct process $p_c \in Participating[r]$, and (2) $|EST[r]| > k$ (let v_m denote the greatest value among the k smallest values of $EST[r]$). Let $p_i \in Participating[r]$. We have $n - k\ r < |sender_i[r]| \Rightarrow est_i[r] \leq v_m$.*

The previous lemma established the validity of the locally evaluable predicate associated with the early stopping property. The next lemma, also associated with agreement and early stopping, defines the meaning of $can\_dec_i[r]$.

Lemma 3 *Let $p_i \in Participating[r]$ ($1 \leq r \leq \lfloor \frac{t}{k} \rfloor + 1$). $can\_dec_i[r] \neq \emptyset \Rightarrow est_i[r]$ is one of the k smallest values in $EST[r]$.*

Lemma 4 *Let p_i be a process that decides during round $r \leq \lfloor \frac{t}{k} \rfloor$. No process decides after the round $r+1$.*

Theorem 1 [Agreement] *No more than k different values are decided.*

Theorem 2 [Decision optimality] *A process that does not crash decides.*

As a correct process does not crash, we have : (consequence of the previous theorem):

Corollary 1 [Termination] *Every correct process decides.*

The next corollary follows from the proof of the previous theorem.

Corollary 2 [Validity] *A decided value is a proposed value.*

Theorem 3 [Early Stopping] *No process halts after the round* $\min(\lfloor f/k \rfloor + 2, \lfloor t/k \rfloor + 1)$.

5 Conclusion

This paper has introduced the notion of *decision optimality* for agreement problems in omission failure models. This optimality criterion means that all the processes that do not crash decide. The paper has presented a k-set agreement protocol that enjoys both this optimality property and is early stopping. A process that does not crash (be it correct or committing send omission failures) decides and halts by the round $\min(\lfloor f/k \rfloor + 2, \lfloor t/k \rfloor + 1)$ where f is the number of actual crashes and $t < n$ the maximum number of processes that may crash. Interestingly, when $\lfloor f/k \rfloor + 2 \leq \lfloor t/k \rfloor$, the very existence of this protocol extends the previous $\lfloor f/k \rfloor + 2$ lower bound [10] to the send omission failure model.

A k-set agreement protocol suited to the synchronous crash failure model is presented in [25]. This protocol is

not only early deciding but provides also "very early" decision in a lot of scenarios. More explicitly, if during the very first rounds, there are either few crashes or a lot of crashes, the protocol terminates very quickly. As an example, the protocol stops after only three rounds when xk ($\forall\ x > 1$) processes have crashed before the protocol starts, and less than k processes crash thereafter. The $\lfloor f/k \rfloor + 2$ lower bound is attained only in the worst case scenario where the crashes are evenly distributed with k crashes per round. To attain this goal, this protocol uses a differential approach that allows a process to take into account the failure pattern and not only the number of failures that occur. Stated with the variables used in the present paper this differential approach would be expressed with the following predicate $|sender_i[r-1] - sender_i[r]| < k\ r$ (instead of $n - k\ r < |sender_i|$). An open problem is the existence (and its design if there is such a protocol) of a k-set agreement protocol satisfying such a "very early decision" criterion despite omission failures.

References

[1] Aguilera M.K. and Toueg S., A Simple Bivalency Proof that t-Resilient Consensus Requires $t+1$ Rounds. *Information Processing Letters*, 71:155-178, 1999.

[2] Attiya H. and Welch J., Distributed Computing, Fundamentals, Simulation and Advanced Topics (Second edition). *Wiley Series on Par. and Dist. Comp.*, 414 pages, 2004.

[3] Borowsky E. and Gafni E., Generalized FLP Impossibility Results for t-Resilient Asynchronous Computations. *Proc. 25th ACM Symposium on Theory of Computation (STOC'93)*, California (USA), pp. 91-100, 1993.

[4] Charron-Bost B. and Schiper A., Uniform Consensus is Harder than Consensus. *J. of Algorithms*, 51(1):15-37, 2004.

[5] Chaudhuri S., More *Choices* Allow More *Faults:* Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation,* 105:132-158, 1993.

[6] Chaudhuri S., Herlihy M., Lynch N. and Tuttle M., Tight Bounds for k-Set Agreement. *Journal of the ACM*, 47(5):912-943, 2000.

[7] Dolev D., Reischuk R. and Strong R., Early Stopping in Byzantine Agreement. *J. of the ACM*, 37(4):720-741, 1990.

[8] Fischer M.J., Lynch N.A., A Lower Bound on the Time to Assure Interactive Consistency. *Information Processing Letters*, 14(4):183-186, 1982.

[9] Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.

[10] Gafni E., Guerraoui R. and Pochon B., From a Static Impossibility to an Adaptive Lower Bound: The Complexity of Early Deciding Set Agreement. *Proc. 37th ACM Symp. on Theory of Computing (STOC'05)*, 2005.

[11] Garay J.A. and Moses Y., Fully Polynomial Byzantine Agreement for $n > 3t$ Processes in $t+1$ Rounds. *SIAM Journal on Computing*, 27(1):247-290, 1998.

[12] Hadzilacos V., Issues of Fault Tolerance in Concurrent Computations. *PhD Thesis, Tech Report 11-84*, Harvard University, Cambridge (MA), 1985.

[13] Hadzilacos V. and Toueg S., Reliable Broadcast and Related Problems. In *Distributed Systems*, ACM Press, New-York, pp. 97-145, 1993.

[14] Herlihy M.P. and Penso L. D., Tight Bounds for k-Set Agreement with Limited Scope Accuracy Failure Detectors. *Proc. 17th Int. Symposium on Distributed Computing (DISC'03)*, Springer Verlag LNCS #2848, pp. 279-291, 2003.

[15] Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923, 1999.

[16] Keidar I. and Rajsbaum S., A Simple Proof of the Uniform Consensus Synchronous Lower Bound. *Information Processing Letters*, 85:47-52, 2003.

[17] Lamport L. and Fischer M., Byzantine Generals and Transaction Commit Protocols. *Unp. manuscript*, 1982.

[18] Lynch N.A., Distributed Algorithms. *Morgan Kaufmann Pub.*, San Fransisco (CA), 872 pages, 1996.

[19] Mostéfaoui A. and Raynal M., k-Set Agreement with Limited Accuracy Failure Detectors. *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 143-152, Portland (OR), 2000.

[20] Mostéfaoui A. and Raynal M., Randomized Set Agreement. *Proc. 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA'01)*, ACM Press, pp. 291-297, Hersonissos (Crete), 2001.

[21] Neiger G. and Toueg S., Automatically Increasing the Fault-Tolerance of Distributed Algorithms. *Journal of Algorithms*, 11:374-419, 1990.

[22] Pease L., Shostak R. and Lamport L., Reaching Agreement in Presence of Faults. *J. of the ACM*, 27(2):228-234, 1980.

[23] Perry K.J. and Toueg S., Distributed Agreement in the Presence of Processor and Communication Faults. *IEEE Transactions on Software Eng.*, SE-12(3):477-482, 1986.

[24] Raïpin Parvédy Ph. and Raynal M., Optimal Early Stopping Uniform Consensus in Synchronous Systems with Process Omission Failures. *Proc. 16th ACM Symposium on Parallel Algorithms and Architectures (SPAA'04)*, Barcelona (Spain), ACM Press, pp. 302-310, 2004.

[25] Raïpin Parvédy Ph., Raynal M. and Travers C., Early-stopping k-set agreement in synchronous systems prone to any number of process crashes. *Proc. 8th Int. Conference on Parallel Computing Technologies (PaCT'05)*, Krasnoyarsk (Russia), Springer Verlag LNCS #3606, pp. 49-58, 2005.

[26] Raïpin Parvédy Ph., Raynal M. and Travers C., Decision Optimal Early-Stopping k-set Agreement in Synchronous Systems Prone to Send Omission Failures. *Tech Report #1689*, IRISA, Université de Rennes (France), 17 pages 2005. http://www.irisa.fr/bibli/publi/pi/2005/1689/1689.html

[27] Raynal M., Consensus in Synchronous Systems: a Concise Guided Tour. *Proc. 9th IEEE Pacific Rim Int. Symposium on Dependable Computing (PRDC'02)*, Tsukuba (Japan), IEEE Computer Press, pp. 221-228, 2002.

[28] Saks M. and Zaharoglou F., Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.