ARTICLE IN PRESS

Theoretical Computer Science xx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Narrowing power vs efficiency in synchronous set agreement: Relationship, algorithms and lower bound☆

Achour Mostéfaoui, Michel Raynal*, Corentin Travers

IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

ARTICLE INFO

Article history:
Received 22 July 2008
Received in revised form 9 March 2009
Accepted 1 September 2009
Communicated by G. Ausiello

Keywords:
Consensus
Efficiency
Lower bound
Round-based algorithm
Set agreement
Synchronous system
t-resilience

ABSTRACT

The k-set agreement problem is a generalization of the uniform 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 at most k different values are decided. It has been shown that any algorithm that solves the k-set agreement problem in synchronous systems that can suffer up to t crash failures requires $\lfloor \frac{t}{k} \rfloor + 1$ rounds in the worst case. It has also been shown that it is possible to design early deciding algorithms where no process decides and halts after $\min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ rounds, where f is the number of actual crashes in a run ($0 \leq f \leq t$).

This paper explores a new direction to solve the k-set agreement problem in a synchronous system. It considers that the system is enriched with base objects (denoted as $[m, \ell]$_SA objects) that allow solving the ℓ-set agreement problem in a set of m processes ($m < n$). The paper makes several contributions. It first proposes a synchronous k-set agreement algorithm that benefits from such underlying base objects. This algorithm requires $O(\frac{t\ell}{mk})$ rounds, more precisely, $\lfloor \frac{t}{\Delta} \rfloor + 1$ rounds, where $\Delta = m\lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)$. The paper then shows that this bound, that involves all the parameters that characterize both the problem (k) and its environment (t, m and ℓ), is a lower bound. The proof of this lower bound sheds additional light on the deep connection between synchronous efficiency and asynchronous computability. Finally, the paper extends its investigation to the early deciding case. It presents a k-set agreement algorithm that directs the processes to decide and stop by round $\min(\lfloor \frac{f}{\Delta} \rfloor + 2, \lfloor \frac{t}{\Delta} \rfloor + 1)$. These bounds generalize the bounds previously established for solving the k-set agreement problem in pure synchronous systems.

1. Introduction

Context of the work. The k-set agreement problem generalizes the uniform consensus problem (that corresponds to the case $k = 1$). This problem has been introduced by S. Chaudhuri to investigate how the number of choices (k) allowed for the processes is related to the maximum number (t) of processes that can crash during a run [8]. The problem can be defined as follows. Each of the n processors (processes) defining the system starts with a value (called a "proposed" value). Each process that does not crash has to decide on 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]

☆ A preliminary version of this paper has appeared in the proceedings of the Int'l Conference on Distributed Computing and Networking (ICDCN 2008). An extended version of this paper has been invited for publication in TCS as one of the best papers presented at the conference after being submitted to the standard TCS refereeing procedure.

* Corresponding author. Tel.: +33 299 84 71 88
E-mail addresses: achour@irisa.fr (A. Mostéfaoui), raynal@irisa.fr (M. Raynal), ctravers@irisa.fr (C. Travers).

[1] This paper considers the crash failure model. The reader interested in the k-set agreement problem in more severe send/receive/general omission failure models can consult the introductory survey [29].

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.09.002

When we consider asynchronous systems, the problem can trivially be solved when $k > t$. Differently, it has been shown that there is no solution in these systems as soon as $k \leq t$ [6,19,30]. (The asynchronous consensus impossibility, case $k = 1$, was demonstrated before, using a different technique [12].[2]) Several approaches have been proposed to circumvent the impossibility to solve the k-set agreement problem in asynchronous systems (e.g., probabilistic protocols [27], unreliable failure detectors with limited scope accuracy [17,26], or conditions associated with input vectors [24]).

The situation is different in synchronous systems where the k-set agreement problem can always be solved, whatever the respective values of t and k. This has an inherent cost, namely, the smallest number of rounds (time complexity measured in communication steps) that have to be executed in the worst case scenario is lower bounded by $\lfloor \frac{t}{k} \rfloor + 1$ [9]. (That bound generalizes the $t + 1$ lower bound associated with the consensus problem [1,3,11,22].)

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 [10,21,28], i.e., protocols that can cope with up to t process crashes, but decide in less than $t + 1$ rounds in favorable circumstances (e.g., when there are few failures). 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) [7,20,32].

In a very interesting way, it has also been shown that the early deciding lower bound for the k-set agreement problem is $\min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ [14]. This lower bound, not only generalizes the corresponding uniform consensus lower bound, but also shows an "inescapable tradeoff" among the number t of faults tolerated, the number f of actual faults, the degree k of coordination we want to achieve, and the best running time achievable. 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).

Related work (1). To our knowledge, two approaches have been proposed and investigated to circumvent the $\min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ lower bound associated with the synchronous k-set agreement problem.

The first is the *fast failure detector* approach that has been proposed and developed in [2] to expedite decision in synchronous consensus. That approach assumes a special hardware that allows a process to detect the crash of any process at most d time units after the crash occurred, where $d < D$, D being the maximum message delay provided by the synchronous system. Both d and D are a priori known by the processes. A fast failure detector-based consensus algorithm that terminates in $D + fd$ is proposed in [2], where it is also shown that $D + fd$ is a lower bound for any algorithm based on a fast failure detector.[3] To our knowledge, this approach has been considered only for the consensus problem.

A second approach that has been proposed to circumvent the $\min(f + 2, t + 1)$ lower bound is the use of conditions [25]. That approach considers that the values proposed by the processes define an input vector with one entry per process. Basically, a *condition* C_t^d (t and d are two parameters that allow defining instances of the condition) is a set of input vectors I such that $\forall I \in C_t^d$, there is a value that appears in I more than $t - d$ times. A deterministic way to define which value has to appear enough times in a vector I (e.g., the maximal value of the vector [23]) allows defining a hierarchy of conditions such that $C_t^0 \subset \cdots \subset C_t^x \subset \cdots \subset C_t^t$ (where C_t^t is the condition including all the input vectors).

[25] presents two main results. Let I be the input vector of the considered run, and C_t^d be a condition. The first result is a synchronous consensus algorithm that allows the processes to decide in (1) one round when $I \in C_t^d$ and $f = 0$, (2) two rounds when $I \in C_t^d$ and $f \leq t - d$, (3) $\min(d + 1, f + 2, t + 1)$ rounds when $I \in C_t^d$ and $f > t - d$, and (4) $\min(f + 2, t + 1)$ when $I \notin C_t^d$. The second result is a proof showing that $\min(d + 1, f + 2, t + 1)$ rounds are necessary in the worst case when $I \in C_t^d$ (and $I \notin C_t^{d-1}$).

An extension of this condition-based approach (combined with the use of appropriate failure detectors) to solve the k-set agreement problem in asynchronous systems has been considered in [24]. It is shown that $k > d$ is a necessary and sufficient requirement for obtaining an asynchronous k-set agreement algorithm based on a condition C_t^d.

Problem addressed in the paper. The paper is about the efficiency (measured in terms of the number of rounds required to decide) of synchronous set agreement algorithms. As it has just been shown, fast failure detectors and conditions are two ways to circumvent the synchronous lower bound. The paper investigates a third approach. That approach is based on base objects that allow narrowing the set of proposed values. Their aim is to play a part similar to fast failure detectors or conditions, i.e., allow expediting decision.

Let us consider as a simple example a test&set object. This object has consensus number 2 [16], which means that it allows solving consensus in an asynchronous system made up of two processes (where one of them can crash), but not in a system made up of $n > 2$ processes (where up to $n - 1$ can crash).[4] Is it possible to use such base objects to speed up synchronous set agreement in a system made up of n processes where up to t may crash? More generally, let $[m, \ell]$_SA denote an object

[2] The impossibility to solve consensus in asynchronous systems is usually named "FLP result" according to the names of its authors [12].

[3] Without a fast failure detector, the cost would be $D \times \min(f + 2, t + 1)$.

[4] The consensus number of a concurrent object type is the maximum number of processes that can solve consensus (despite any number of process crashes) using only atomic registers and objects of that type. The consensus number of test&set objects, queues, and stacks is 2 [16].

Please cite this article in press as: A. Mostéfaoui, et al., Narrowing power vs efficiency in synchronous set agreement: Relationship, algorithms and lower bound, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.002

that allows solving ℓ-set agreement in a synchronous system of m processes.[5] As fast failure detectors or conditions, these objects are assumed given for free. So, the previous question becomes:

- Is it possible to benefit from $[m, \ell]$_SA objects to build a t-resilient synchronous $[n, k]$_SA object (i.e., a k-set agreement object that has to cope with up to t process crashes)?
- If such a construction is possible, is its cost smaller than $\lfloor \frac{t}{k} \rfloor + 1$, or smaller than $\min(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1)$ if we are interested in an early deciding $[n, k]$_SA object?

If m, ℓ, n and k are such that there is an integer a with $n \leq am$ and $a\ell \leq k$, it is possible to solve the k-set agreement problem without exchanging any value (i.e., in 0 round!) whatever the value of t. This is trivially obtained by partitioning the n processes into a subsets of at most m processes, and using in each subset a $[m, \ell]$_SA object in order that each process be provided with a decided value. So, the interesting cases are when the values m, ℓ, n and k do not allow a trivial partitioning such as the previous one.

Another way to present the previous question is the following: how many crashes can we tolerate when we want to build a synchronous [10, 3]_SA object from [2, 1]_SA objects, if one wants to decide in at most one round? In at most two rounds? In at most three rounds?

So, the point investigated in the paper is a mathematical one, namely the computational power of $[m, \ell]$_SA objects when one wants to solve the k-set agreement problem in a set of n processes.

From a more practical point of view, we can see the system as made up of clusters of m processes, such that an operation involving only processes of a given cluster can be performed very efficiently, i.e., in a time that is smaller than the maximal message transfer delay involving processes belonging to different clusters. One can see each $[m, \ell]$_SA object as a hardwired object accessible by a set of m processes only. That is the sense in which the sentence "the $[m, \ell]$_SA objects are given for free" should be understood.

Related work (2). In [18], Herlihy and Rajsbaum are interested in the same question as ours in an asynchronous context: in which circumstances can $[m, \ell]$_SA objects help implement an $[n, k]$_SA object? As we will see, the lower bound established in the paper is a reduction to their lower bound (via a simulation due to Gafni [13]). This is an unusual case where a synchronous lower bound is proved from an asynchronous lower bound.

Results. The paper presents the following results.

- It first presents a synchronous message-passing algorithm that builds an $[n, k]$_SA object from $[m, \ell]$_SA objects. This algorithm works for any values of n, k, m, and ℓ (assuming, of course, $n > k$ and $m > \ell$).
- The paper then shows that the number of rounds (R_t) of the previous algorithm varies as $\Theta(\frac{t\ell}{mk})$. This means that (1) R_t decreases when the coordination degree k increases (i.e., when less synchronization is required), or when the number of processes m involved in each underlying object increases, and (2) R_t increases when the underlying object is less and less powerful (i.e., when ℓ increases) or when the number of process crashes that the algorithm has to tolerate increases. More precisely, we have:

$$R_t = \left\lfloor \frac{t}{m \lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)} \right\rfloor + 1.$$

When we consider the previous example of building, in a synchronous system, a [10, 3]_SA object from [2, 1]_SA objects, we can conclude that $R_t = 1$ requires $t < 6$, while $R_t = 2$ allows $t = 9$. Moreover, as there are only $n = 10$ processes, there is no value of t that can entail an execution in which $R_t = 3$ are required (for it to occur, we should have $12 \leq t < 18$ and $n > t$).

To have a better view of R_t, it is interesting to look at special cases.

- Case 1. Build a consensus object in a synchronous system from [1, 1]_SA base objects or $[m, m]$_SA objects (i.e., from base objects that have no power). It is easy to see that $R_t = t + 1$ (that is the well-known lower bound for synchronous t-resilient consensus).
- Case 2. Build an $[n, k]$_SA object in a synchronous system from [1, 1]_SA base objects or $[m, m]$_SA objects (base objects without power). It is easy to see that $R_t = \lfloor \frac{t}{k} \rfloor + 1$, (that is the lower bound for synchronous t-resilient k-set agreement).
- Case 3. Build a synchronous consensus from $[m, 1]$_SA base objects (i.e., consensus objects). In that case $R_t = \lfloor \frac{t}{m} \rfloor + 1$.
- Case 4. Build a synchronous $[n, \ell]$_SA object from $[m, \ell]$_SA base objects. In that case, $R_t = \lfloor \frac{t}{m} \rfloor + 1$.
- Case 5. Build a synchronous $[n, k]$_SA object from $[m, 1]$_SA base objects (i.e., consensus objects). We then have $R_t = \lfloor \frac{t}{mk} \rfloor + 1$.

[5] Objects such as $[m, \ell]$_SA objects have been used in [18] (under the name (m, ℓ)-consensus objects) to solve the k-set agreement problem in *asynchronous* systems prone to process crash failures.

Please cite this article in press as: A. Mostéfaoui, et al., Narrowing power vs efficiency in synchronous set agreement: Relationship, algorithms and lower bound, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.002

These particular instances show clearly how the coordination degree and the size of the base objects (measured by the value m) affect the maximal number of rounds executed by the algorithm and consequently allow expediting the decision.

- The paper then shows that the value R_t is optimal when one wants to build, in a synchronous system, an $[n, k]$_SA object from $[m, \ell]$_SA base objects. This optimality result generalizes previous lower bounds proved for special cases such as consensus [1,11,21], and set agreement [9].

  The optimality proof relies on two theorems, one from Gafni [13], the other from Herlihy and Rajsbaum [18]. Gafni's theorem establishes a deep connection between solvability in asynchronous system and lower bounds (efficiency) in synchronous systems. The Herlihy and Rajsbaum theorem is based on the impossibility to solve some set agreement problems in asynchronous systems.
- Finally, the paper extends the algorithm to the early decision case. More specifically, the maximal number of rounds of the early deciding version of the algorithm is the following:

$$R_f = \min\left(\left\lfloor \frac{f}{\Delta} \right\rfloor + 2, \left\lfloor \frac{t}{\Delta} \right\rfloor + 1\right) \quad \text{where } \Delta = m\left\lfloor \frac{k}{\ell} \right\rfloor + (k \bmod \ell).$$

  It is easy to see that this early decision bound generalizes the lower bounds that are known for the special consensus and set agreement cases.

This paper is an endeavor to capture the essence of the synchronous set agreement and provide the reader with a better understanding of it. To that end, it considers design simplicity as a first-class citizen when both designing algorithms and proving lower bound results.

As already noticed, the lower bound proof relies on previous theorems. We do think that Gafni's theorem [13] (that states that an asynchronous system with at most t' crashes can implement the first $\lfloor \frac{t}{t'} \rfloor$ rounds of a synchronous system with up to t failures) is a fundamental theorem of fault-tolerant distributed computing. The lower bound proof of this paper shows an application of this powerful theorem.

Roadmap. The paper is made up of 6 sections. Section 2 introduces the system model and definitions. Section 3 presents the algorithm that builds an $[n, k]$_SA object from $[m, \ell]$_SA objects in R_t synchronous rounds. Section 4 proves that R_t is a lower bound on the number of rounds for any synchronous algorithm that builds an $[n, k]$_SA object from $[m, \ell]$_SA objects. Section 5 considers the early decision case. Finally, Section 6 concludes the paper.

2. Computation model and the set agreement problem

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 [8] is defined by the following three properties (as we can see 1-set agreement is the uniform consensus problem):

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

Process model. The system model consists of a finite set of n processes, namely, $\Pi = \{p_1, \ldots, p_n\}$. A process is a sequence of steps (execution of a base atomic operation). A process is *faulty* during an execution if it stops executing steps (after it has crashed a process executes no step). As already indicated, t is an upper bound on the number of faulty processes, while f denotes the number of processes that crash during a particular run, $0 \leq f \leq t < n$. (Without loss of generality we consider that the execution of a step by a process takes no time.)

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 $t < k$ [8].

Communication/coordination model. The processes communicate by sending and receiving messages through channels. Every pair of processes p_i and p_j is connected by a channel. The sending of a message and the reception of a message are atomic operations. The underlying communication system is assumed to be failure-free: there is no creation, alteration, loss or duplication of messages.

In addition to messages, the processes can coordinate by accessing $[m, \ell]$_SA objects. Such an object is a one-shot object that can be accessed by at most m processes. Its power is to solve the ℓ-set agreement problem among m processes. Let us observe that, for $1 \leq m \leq n$, an $[m, n]$_SA object is a trivial object that has no coordination power.

Round-based synchrony. The system is *synchronous*. This means that each of its runs 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 read, and whose progress is given for free: it is managed by an external entity. A round is made up of two main consecutive phases:

Please cite this article in press as: A. Mostéfaoui, et al., Narrowing power vs efficiency in synchronous set agreement: Relationship, algorithms and lower bound, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.002

- A send phase in which each process sends zero or one message to each other process.[6] If a process crashes during the send phase of a round, an arbitrary subset of the processes to which it sent messages will receive these messages.
- A receive phase in which each process receives messages. 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 very same round r if p_i does not crash during the round r (if p_j crashes during the round r, the message can be or not received by p_j).[7]

Before or after a phase, a process can execute local computations (e.g., process the messages it received during the current round). It can also invoke an underlying $[m, \ell]$_SA base object.

3. A synchronous $[n, k]$_SA algorithm

This section presents a simple algorithm that, when at most t processes may crash, builds an $[n, k]$_SA object if the system provides the n processes with round-based synchrony and $[m, \ell]$_SA base objects.

Notation. In all the rest of the paper we are using the following notations:

- $\alpha = \lfloor \frac{k}{\ell} \rfloor$ and $\beta = k \bmod \ell$ (i.e., $k = \alpha\ell + \beta$),
- $\Delta = \alpha m + \beta = m \lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)$,[8]
- $R_t = \lfloor \frac{t}{\Delta} \rfloor + 1 = \left\lfloor \frac{t}{m \lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)} \right\rfloor + 1$.

3.1. The algorithm

The algorithm is pretty simple. It is described in Fig. 1. A process p_i invokes the operation $\text{propose}_{k,n}^{\ell,m}(v_i)$ where v_i is the value it proposes. That value is initially stored in the local variable est_i (line 01), that afterwards will contain the current estimate of p_i's decision value (line 10). The process terminates when it executes the return(est_i) statement.

Each process executes R_t rounds (line 02). During any round r, only Δ processes are allowed to send their current estimates. These processes are called the *senders* of round r. When $r = 1$, they are the processes $p_1, \ldots, p_\Delta$, during the second round the processes $p_{\Delta+1}, \ldots, p_{2\Delta}$, and so on (lines 04–05).

The Δ senders of a round r are partitioned into $\lceil \frac{\Delta}{m} \rceil$ subsets of m processes (the last subset containing possibly less than m processes), and each subset uses an $[m, \ell]$_SA object to narrow the set of its current estimates (lines 06–07). After this "narrowing", each sender process sends its new current estimate to all the processes. A process p_i accesses an $[m, \ell]$_SA object by invoking the operation propose(est_i). The $\lceil \frac{\Delta}{m} \rceil$ $[m, \ell]$_SA objects used during a round r are in the array $SA[r, 0..\lceil \frac{\Delta}{m} \rceil - 1]$.[9] Finally, when during a round, a process p_i receives estimates, it updates est_i accordingly (line 10).

It is important to see that, if at least one sender process does not crash during a round, at most $k = \alpha\ell + \beta$ estimates are sent during that round, which means that k-set agreement is guaranteed as soon as there is a round during which an active process does not crash.

3.2. Proof of the algorithm

Lemma 1. *Let $nc[r]$ be the number of processes that crash during the round r. There is a round r such that $r \leq R_t$ and $nc[r] < \Delta$.*

Proof. Let $t = \alpha'\Delta + \beta'$ with $\alpha' = \lfloor \frac{t}{\Delta} \rfloor$ and $\beta' = t \bmod \Delta$. The proof is by contradiction. let us assume that, $\forall r \leq R_t$, we have $nc[r] \geq \Delta$. We then have:

$$\sum_{r=1}^{R_t} nc[r] \geq \Delta \times R_t = \Delta\left(\left\lfloor \frac{t}{\Delta} \right\rfloor + 1\right) = \Delta\left(\alpha' + \left\lfloor \frac{\beta'}{\Delta} \right\rfloor + 1\right) = \Delta \times \alpha' + \Delta > t.$$

Consequently, there are more than t processes that crash: a contradiction. $\square_{Lemma\ 1}$

Lemma 2. *At any round r, at most k different estimate values are sent by the processes.*

[6] It is easy to see that this model has the same power as the model where, at each round, each process has to send the same message to all the processes [3,22,31].

[7] Let us observe that this synchronous model is very general: if, during a round r, a process first sends a message to all and then crashes, an arbitrary subset of the processes receive the message.

[8] A function $J(u) = \ell \lfloor \frac{u}{m} \rfloor + \min(\ell, u \bmod m) - 1$, is used in [18]. While Δ and the formula $J(u)$ share some "shape", it is easy to see that they are different and cannot be compared.

[9] Actually, only $R_t \lfloor \frac{\Delta}{m} \rfloor$ base $[m, \ell]$_SA objects are needed. This follows from the following observation: during each round r, if $\beta \neq 0$, the "last" β sender processes do not need to use such an $[m, \ell]$_SA object because $\beta \leq \ell$. (Let us recall that $0 \leq \beta < \ell$ and Δ is defined as $\alpha\, m + \beta$.)

Please cite this article in press as: A. Mostéfaoui, et al., Narrowing power vs efficiency in synchronous set agreement: Relationship, algorithms and lower bound, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.002

```
Operation propose^{ℓ,m}_{k,n}(v_i)
(01) est_i ← v_i;
(02) for r = 1, 2, ... , R_t do % r: round number %
(03) begin synchronous round
(04)     first_sender ← (r − 1)Δ + 1; last_sender ← rΔ;
(05)     if first_sender ≤ i ≤ last_sender then % p_i is "sender" at round r %
(06)       let y be such that first_sender + ym ≤ i < last_sender + (y + 1)m;
           % y is index of the [m, ℓ]_SA object used by p_i %
(07)       est_i ← SA[r, y].propose(est_i);
(08)       for each j ∈ {1, ... , n} do send (est_i) to p_j end do
(09)     end if;
(10)     est_i ← any est value received if any, unchanged otherwise
(11) end synchronous round;
(12) return(est_i).
```

Fig. 1. $[n, k]$_SA object from $[m, \ell]$_SA objects in a synchronous system (code for p_i).

Proof. Let us recall that $k = \alpha\,\ell + \beta$ (Euclidean division of k by ℓ) and $\Delta = \alpha\,m + \beta = m\,\lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)$.

Due to the lines 04–05, at most Δ processes are senders at each round r. These Δ sender processes are partitioned into $\alpha = \lfloor \frac{\Delta}{m} \rfloor$ sets of exactly m processes plus a set of β processes. As each underlying $[m, \ell]$_SA object used during the round r outputs at most ℓ estimates values from among the (at most) m values it is proposed, it follows that at most $\alpha\ell + \beta = k$ estimate values can be output by these objects, which proves the lemma. □*Lemma 2*

Lemma 3. [Agreement] *At most k different values are decided by the processes.*

Proof. At any round the number of senders is at most Δ (lines 04–05). Moreover, due to Lemma 1, there is at least one round $r \leq R_t$ during which a correct process is a sender. If follows from Lemma 2, line 08 and line 10, that, at the end of such a round r, the estimates of the processes contain at most k distinct values. □*Lemma 3*

Theorem 1. *The algorithm described in Fig. 1 is a synchronous t-resilient k-set agreement algorithm.*

Proof. The termination property follows directly from the synchrony of the model: a process that does not crash executes R_t rounds. The validity property follows directly from the initialization of the estimate values est_i, the correctness of the underlying $[m, \ell]$_SA objects (line 07), and the fact that the algorithm exchanges only est_i values. Finally, the agreement property is Lemma 3. □*Theorem 1*

4. Lower bound on the number of rounds

This section proves that the previous algorithm is optimal with respect to the number of rounds. The proof of this lower bound is based on (1) a deep connection relating synchronous efficiency and asynchronous computability in presence of failures [13], and (2) an impossibility result in asynchronous set agreement [18].

4.1. Notation and previous results

This section uses the following notations.

- $\mathcal{S}_{n,t}[\emptyset]$ denotes the classical round-based synchronous system model made up of n processes, where up to t processes may crash [3,22,31].
- $\mathcal{S}_{n,t}[m, \ell]$ is the $\mathcal{S}_{n,t}[\emptyset]$ system model enriched with $[m, \ell]$_SA objects. This is the model defined in Section 2 (n processes, at most t process crashes, coordination possible through $[m, \ell]$_SA objects).
- $\mathcal{AS}_{n,t}[\emptyset]$ denotes the classical shared memory asynchronous system model, as described in standard textbooks [3,22]. The system is made of n processes, at most t of them may crash. Processes communicate by reading and writing atomic shared registers.
- $\mathcal{AS}_{n,t}[m, \ell]$ denotes the asynchronous system model $\mathcal{AS}_{n,t}[\emptyset]$ enriched with $[m, \ell]$_SA objects. (From a computability point of view, $\mathcal{AS}_{n,t}[\emptyset]$ is weaker than $\mathcal{AS}_{n,t}[m, \ell]$.)

The following theorems are central in proving that R_t is a lower bound.

Gafni's theorem. This first theorem is on the simulation, on an asynchronous system, of a round-based algorithm designed for the $\mathcal{S}_{n,t}[\emptyset]$ model. Let A be a round-based t-resilient n-process synchronous distributed algorithm. The meaning of the words "it is possible to simulate" used in the theorem can be informally defined as follows. Let $S(A)$ denote an asynchronous k-resilient n-process distributed algorithm that takes A as input ($S(A)$ stands for simulation of A). The theorem states that there is an algorithm S such that $S(A)$ produces a round-based run whose outputs during the first $\lfloor \frac{t}{k} \rfloor$ rounds could have

Please cite this article in press as: A. Mostéfaoui, et al., Narrowing power vs efficiency in synchronous set agreement: Relationship, algorithms and lower bound, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.002

init $m_i \leftarrow \mathsf{compute}_i(0, v_i)$; % message to be sent in the first round %

for $r = 1, 2, \ldots, R$ **do** % r: round number %
  begin synchronous round
    for each $j \in \{1, \ldots, n\}$ **do** *send* (m_i) *to* p_j **end do**
    receive the round r messages; **let** $rec_i = \{\langle p_j, m_j\rangle : m_j \text{ is received from } p_j\}$;
    $\langle d_i, m_i\rangle \leftarrow \mathsf{compute}_i(r, rec_i)$
  end synchronous round;
end for
$\mathsf{return}(d_i)$.

Fig. 2. Generic algorithm solving a decision task in $\mathcal{S}_{n,t}[\emptyset]$.

been produced by the first $\lfloor \frac{t}{k} \rfloor$ rounds of A run in a synchronous system.[10] Said in another way, at an appropriate abstraction level, the outputs of the first $\lfloor \frac{t}{k} \rfloor$ rounds of $S(A)$ in $\mathcal{AS}_{n,k}[\emptyset]$ could have been produced by the first $\lfloor \frac{t}{k} \rfloor$ rounds of A in $\mathcal{S}_{n,t}[\emptyset]$.

Theorem 2 (*Gafni [13]*). *Let $n > t \geq k > 0$. It is possible to simulate in $\mathcal{AS}_{n,k}[\emptyset]$ the first $\lfloor \frac{t}{k} \rfloor$ rounds of any algorithm designed for $\mathcal{S}_{n,t}[\emptyset]$ system model.*

Let us now explain the meaning of the word "simulate" in a more operational way. Let us first observe that, at each round r of a run of an algorithm designed for $\mathcal{S}_{n,t}[\emptyset]$, each process p_i (that has not crashed) receives a set of messages (rec_i), performs local computation, and prepares a message (m_i) that will be sent during the send phase of the round $r + 1$. Thus, a deterministic synchronous algorithm is fully described at each process p_i by a function $\mathsf{compute}_i()$ that, at each round r, updates the local state of p_i from its previous state and the set of messages it has received in the current round. Moreover, $\mathsf{compute}_i()$ returns a pair $\langle d_i, m_i\rangle$ where d_i is a decision value or $\perp$ (if no decision has yet been determined), and m_i is the message to be sent by p_i in round $r + 1$. A generic synchronous algorithm for a decision task is described in Fig. 2.

Let us observe that, for a deterministic synchronous algorithm A, executed in $\mathcal{S}_{n,t}[\emptyset]$, the state of p_i after R rounds is fully determined by its initial state and its history, i.e., the sequence of the sets of messages $(rec_i^1, \ldots, rec_i^R)$ it has received during each round (from 1 to R). Hence, simulating R rounds of A means emulating the send and receive phases, for each process and each round r, $1 \leq r \leq R$. Operationally, the round r send and receive phases of the generic algorithm are replaced by an invocation of $\mathsf{simulate}()$, that takes as a parameter a message m_i and returns a set rec_i^r made of pairs $\langle p_j, m_j\rangle$. The simulation is correct if there exists a R rounds run of A in $\mathcal{S}_{n,t}[\emptyset]$ in which, for each process p_i, (1) the messages sets $rec_i^1, \ldots, rec_i^R$ are received by p_i in the corresponding rounds r, $1 \leq r \leq R$, and (2) if $\exists r$, $1 \leq r \leq R$, such that $p_i \notin rec_i^r$, then p_i has failed by the end of the round r. The round-based synchronous communication pattern and the fact that at most t processes fail in any run of $\mathcal{S}_{n,t}[\emptyset]$ defines allowable sequences of sets $rec_i^1, \ldots, rec_i^R$ as follows[11]:

- $\forall p_i \in \Pi, r, 1 \leq r \leq R : |rec_i^r| \geq n - t$,
- $\forall p_i, p_j, r, 1 \leq r \leq R : rec_j^{r+1} \subseteq rec_i^r$.

In addition, in order to benefit from the simulation of A in the base model $\mathcal{AS}_{n,k}[\emptyset]$, at least one process that does not fail must not fail in the emulated synchronous run. More precisely:

- $|\{p_i : p_i \in rec_i^R\}| \geq k + 1$.

The simulation algorithm described in [13] for the asynchronous system $\mathcal{AS}_{n,k}[\emptyset]$ ensures the three properties above subject to the constraint $R \leq \lfloor \frac{t}{k} \rfloor$.

The next corollary is a simple extension of Gafni's theorem suited to our needs.

Corollary 1. *Let $n > t \geq k > 0$, and let A an algorithm designed for $\mathcal{S}_{n,t}[m, \ell]$ system model that solves the x-set-agreement problem in at most $R \leq \lfloor \frac{t}{k} \rfloor$ rounds. There exists an algorithm A' that solves the x-set-agreement problem in $\mathcal{AS}_{n,k}[m, \ell]$.*

Proof. Without loss of generality, let us assume that A follows the generic pattern as described in Fig. 2. In $\mathcal{S}_{n,t}[m, \ell]$, during each round, a process can access $[m, \ell]\_SA$ objects when it updates its local state. This means that each execution of $\mathsf{compute}_i()$ may invoke the $\mathsf{propose}()$ operation on base $[m, \ell]\_SA$ objects.

The system model $\mathcal{AS}_{n,k}[m, \ell]$ also provides processes with base $[m, \ell]\_SA$ objects. Hence, R-rounds runs of algorithm A can be simulated in $\mathcal{AS}_{n,k}[m, \ell]$ as follows. The send and receive phases are emulated with the simulation algorithm whose

[10] The "simulation of a distributed algorithm, designed for a system model, in a different system model" is an important notion encountered in distributed computing. Among these simulations, one of the most famous is the so-called "synchronizer concept" that is an asynchronous algorithm that allows any failure-free synchronous algorithm to be run on the top of a failure-free asynchronous system [4]. When failures are considered, simulations become more involved, or can even be impossible. The interested reader may consult the specialized literature (e.g., [3,5,22]) where simulation-related impossibility results and simulations in presence of failures are presented.

[11] $rec_i^r \subseteq rec_j^{r'}$ is a shorthand for $\{p_k : \exists\langle p_k, m_k\rangle \in rec_j^r\} \subseteq \{p_{k'} : \exists\langle p_{k'}, m_{k'}\rangle \in rec_j^{r'}\}$. Similarly, $p_k \in rec_i$ means $\exists\langle p, m\rangle \in rec_i$ such that $p = p_k$.

Please cite this article in press as: A. Mostéfaoui, et al., Narrowing power vs efficiency in synchronous set agreement: Relationship, algorithms and lower bound, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.002

existence and correctness are guaranteed by Theorem 2, while the propose() operation on the $[m, \ell]\_SA$ objects are directly supplied by $\mathcal{AS}_{n,k}[m, \ell]$.

As $R \leq \lfloor \frac{t}{k} \rfloor$, Gafni's simulation guarantees that at least one correct process p_i in the base model $\mathcal{AS}_{n,k}[m, \ell]$ has not failed by the end of round R in the emulated synchronous run. Since in every synchronous t-resilient run of A, every process that does not fail computes a decision by the end of round R, p_i obtains a valid decision value d_i by the end of the emulation of the first R rounds. By the specification of the k-set-agreement problem, this decision is also a valid decision for any other process p_j. Hence, p_i writes its decision d_i in shared memory to help other processes to decide before returning that value. Since p_i is correct, every correct process that fails in the simulated synchronous run eventually observes the value written by p_i and consequently eventually decides. □*Corollary 1*

The Herlihy–Rajsbaum theorem. Finally, the theorem that follows characterizes the "computability power" of a system $\mathcal{AS}_{n,t}[\emptyset]$ augmented with $[m, \ell]\_SA$ objects. Expressed with our terminology, it answers the following question: "Which are the values of $K < n$ for which there is no implementation of a $[n, K]\_SA$ object in $\mathcal{AS}_{n,t}[m, \ell]$?"

Theorem 3 (*Herlihy–Rajsbaum [18]*). *Let $J_{m,\ell}$ be the function $u \to \ell \lfloor \frac{u}{m} \rfloor + \min(\ell, u \bmod m) - 1$. There is no algorithm that solves the K-set agreement problem, with $K = J_{m,\ell}(t+1)$, in $\mathcal{AS}_{n,t}[m, \ell]$.*

4.2. The lower bound

Theorem 4. *Let $1 \leq \ell \leq m < n$ and $1 \leq k \leq t < n$. Any algorithm that solves the k-set agreement problem in $\mathcal{S}_{n,t}[m, \ell]$ has at least one run in which at least one process does not decide before the round $R_t = \left\lfloor \frac{t}{m\lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)} \right\rfloor + 1$.*

Proof. The proof is by contradiction. Let us assume that there is an algorithm A that solves the k-set agreement problem in at most $R < R_t$ rounds in $\mathcal{S}_{n,t}[m, \ell]$ (this means that any process decides by at most R rounds, or crashes before). We consider two cases.

- $k < \ell$. We have then $R < R_t = \lfloor \frac{t}{k} \rfloor + 1$.
  1. As $k < \ell$, the ℓ-set agreement can be solved in $\mathcal{AS}_{n,k}[\emptyset]$. It follows that, as far as set agreement is concerned, $\mathcal{AS}_{n,k}[\emptyset]$ and $\mathcal{AS}_{n,k}[m, \ell]$ have the same computational power.
  2. As A solves the k-set-agreement problem in at most $R \leq \lfloor \frac{t}{k} \rfloor$ rounds in system model $\mathcal{S}_{n,t}[m, \ell]$, it follows from the corollary of Gafni's theorem that the k-set agreement problem can be solved in $\mathcal{AS}_{n,k}[m, \ell]$.
  3. Combining the two previous items, we obtain an algorithm that solves the k-set agreement problem in $\mathcal{AS}_{n,k}[\emptyset]$. This contradicts the impossibility to solve the k-set agreement problem in $\mathcal{AS}_{n,k}[\emptyset]$ [6,19,30], which proves the theorem for the case $k < \ell$.
- $k \geq \ell$. Let us recall the definition $\Delta = m\lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell) = \alpha\, m + \beta$. We then have $R < R_t = \lfloor \frac{t}{\Delta} \rfloor + 1$.
  1. A solves the k-set-agreement problem within at most $R \leq \lfloor \frac{t}{\Delta} \rfloor$ rounds in system model $\mathcal{S}_{n,t}[m, \ell]$. By Corollary 1, there exists an algorithm that solves the k-set-agreement problem in system model $\mathcal{AS}_{n,\Delta}[m, \ell]$.
  2. Considering the argument used in the Herlihy–Rajsbaum theorem, we have the following:

$$\begin{aligned} J_{m,\ell}(\Delta+1) &= \ell \left\lfloor \frac{\Delta+1}{m} \right\rfloor + \min\big(\ell, (\Delta+1) \bmod m\big) - 1, \\ &= \ell \left\lfloor \frac{\alpha\, m + \beta + 1}{m} \right\rfloor + \min\big(\ell, (\alpha\, m + \beta + 1) \bmod m\big) - 1, \\ &= \ell \left(\alpha + \left\lfloor \frac{\beta+1}{m} \right\rfloor\right) + \min\big(\ell, (\beta+1) \bmod m\big) - 1. \end{aligned}$$

  Let us observe that $\ell \leq m$. Moreover, as $\beta = k \bmod \ell$, we also have $\beta < \ell$. To summarize: $\beta < \ell \leq m$. There are two cases to consider.

  (a) $m = \beta + 1$. Observe that this implies that $\ell = m$ and $\ell - 1 = \beta$.

$$\begin{aligned} J_{m,\ell}(\Delta+1) &= \ell(\alpha+1) + \min\big(\ell, m \bmod m\big) - 1, \\ &= \ell\,\alpha + \ell - 1 = \ell\,\alpha + \beta = k. \end{aligned}$$

  (b) $m > \beta + 1$:

$$\begin{aligned} J_{m,\ell}(\Delta+1) &= \ell\,\alpha + \min\big(\ell, (\beta+1) \bmod m\big) - 1, \\ &= \ell\,\alpha + \beta + 1 - 1 = k. \end{aligned}$$

  In both cases, $J_{m,\ell}(\Delta+1) = k$. It follows from the Herlihy–Rajsbaum theorem that there is no algorithm that solves the $J_{m,\ell}(\Delta+1)$-set agreement problem (i.e., the k-set agreement problem) in $\mathcal{AS}_{n,\Delta}[m, \ell]$.
  3. The two previous items contradict each other, thereby proving the theorem for the case $k < \ell$. □*Theorem 4*

Operation ED_propose$^{\ell,m}_{k,n}(v_i)$
(01) $est_i \leftarrow v_i$;
(02) **for** $r = 1, 2, \ldots, R_t$ **do** % r: round number %
(03) **begin synchronous round**
(A1) **if** (COMMIT received during round $(r-1)$) **then**
(A2) **for each** $j \in \{1, \ldots, n\} \setminus \{i\}$ **do** *send* (COMMIT) *to* p_j **end do**;
(A3) halt()
(A4) **end if**;
(04) $first\_sender \leftarrow (r-1)\Delta + 1$; $last\_sender \leftarrow r\Delta$;
(05) **if** $first\_sender \leq i \leq last\_sender$ **then** % p_i is "sender" at round r %
(06) let y be such that $first\_sender + ym \leq i < last\_sender + (y+1)m$;
% y is index of the $[m, \ell]$_SA object used by p_i %
(07) $est_i \leftarrow SA[r, y]$.propose$(est_i)$;
(08) **for each** $j \in \{1, \ldots, n\}$ **do** *send* (est_i) *to* p_j **end do**
(09) **end if**;
(B1) **if** (p_i was a sender at round $r-1$) **then**
(B2) **for each** $j \in \{1, \ldots, n\} \setminus \{i\}$ **do** *send* (COMMIT) *to* p_j **end do**;
(B3) decide(est_i); halt()
(B4) **end if**;
(10) $est_i \leftarrow$ any est value received during r if any, unchanged otherwise;
(C1) **if** (COMMIT received during r) **then** decide(est_i) **end if**
(11) **end synchronous round**;
(12') **if** (not yet decided) **then** decide(est_i) **end if**; halt()

Fig. 3. Early deciding $[n, k]$_SA object from $[m, \ell]$_SA objects in a synchronous system (code for p_i).

5. Early decision

This section extends the algorithm described in Fig. 1 in order to obtain an early deciding algorithm. As announced in the Introduction, the resulting algorithm allows the processes to decide by round $R_f = \min\left(\lfloor \frac{f}{\Delta} \rfloor + 2, \lfloor \frac{t}{\Delta} \rfloor + 1\right)$, where $\Delta = m\lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)$.

In the algorithm described in Fig. 1, decision and halting of a process p_i are a single atomic action, namely, a process atomically decides and halts when it executes the statement return(est_i) on line 12. Differently, in the algorithm developed in this section, "p_i decides" and "p_i stops" are two distinct events. A process first decides by executing the statement decide(est_i), and then stops when it executes the statement halt().

5.1. The early deciding algorithm

This algorithm is described in Fig. 3. It is obtained by enriching the base algorithm in a relatively simple way: a few new statements are added to obtain early decision. These are the new lines prefixed with the letter A, B or C. None of the other lines is modified, except line 12 (relabeled 12') that has now to take into account the fact that decision and halting are distinct base operations.

Base principle. The design principles of the enriched algorithm are as follows. A process p_i that is a sender during a round r' and participates in the next round $r'+1$ (so, it has not crashed by the end of r'), sends to all the processes a control message (denoted COMMIT) during the round $r'+1$ (additional lines B1–B4). In that way, p_i informs all the processes that the estimate value it sent during the previous round r' was received by all the processes (this follows from the communication synchrony property). Consequently, p_i decides it and halts (line B4).

Moreover, as at most k different values are sent during a round (Lemma 2), and at least one process (namely, p_i) sent a value to all during r', it follows from the fact that p_i participates to the round $r'+1$ that the estimates of all the processes contain at most k different values at the end of r'. Consequently, a process that receives a COMMIT message during a round $r'+1$ can decide the value of its estimate at the end of the round r' (additional line C1).

Always ensuring early decision and halting. Considering the base algorithm enriched with the lines prefixed by B and C it is easy to see that, if at least one process in $p_1, \ldots, p_\Delta$ does not crash, the processes decide in two rounds. If all the processes $p_1, \ldots, p_\Delta$ crash and at least one process in $p_{\Delta+1}, \ldots, p_{2\Delta}$ does not crash, the decision is obtained in at most 3 rounds etc.

Unfortunately, the previous addition of statements is not sufficient to ensure early decision in all failure scenarios. As an example, let us consider the following scenario in which all the processes $p_1, \ldots, p_\Delta$ have initially crashed, except one of them, say p_i. That process p_i executes the first round (sending its estimates to all the processes at line 08, and consequently est_i is the only estimate value present in the system at the end of the first round), and proceeds to the second round during which it sends COMMIT to all the processes in $p_{\Delta+1}, \ldots, p_{2\Delta}$ and then crashes before sending COMMIT to the other processes (line B2). Moreover, no other process crashes. It follows that the processes $p_{\Delta+1}, \ldots, p_{2\Delta}$ early decide during the second round (line C1), while the other processes do not. In order to ensure that they decide as soon as possible, a simple solution

consists in adding the lines prefixed by A: a process that has received a COMMIT message during a round r (it has then decided during that round at line C1), forwards this COMMIT message during the next round $r+1$ and then halts (lines A1–A4).

The final algorithm described in Fig. 3 is such that:

- Decision and halting during the same round: If a process p_i decides and halts during the same round r, this occurs at line B3. In that case, it follows from line C1 that p_i has sent its estimate to all during the round $r-1$.
- Decision halting at different rounds: If a process p_i decides during a round r and halts during a round r', we have the following: it decides at line D1 during r and halts at line A3 during $r' = r+1$.

5.2. Proof and early decision

Notation.

- Let $SENDERS[r]$ be the set of the processes p_i such that p_i is a sender during the round r (i.e., $(r-1)\Delta + 1 \leq i \leq r\Delta$).
- Let $EST[0]$ be the set of proposed values, and $EST[r]$ be the set of the values contained in the est_i local variables of the processes that decide during r or proceed to $r+1$.

Lemma 4 (*Agreement*). *At most k different values are decided by the processes.*

Proof. If no process decides at line B3 or C1, the proof is the same as in Lemma 3. So, let us consider the case where at least one process decides at line B3 or C1. Let us observe that if a process decides at line C1 during a round r, there is necessarily a process that sent a COMMIT message at line B3 during a round $r' \leq r$ (maybe this message arrives after having been forwarded from round to round). This means that there is a round $r'' < r'$ during which a process p_x has sent its estimate value to all (line 08), and at least one COMMIT message sent by p_x (line C2) during $r''+1 \leq r' \leq r$ has been received. We conclude from the synchronous model that the estimate value sent by p_x during r'' has been received by all the processes. This constitutes observation $O1$.

Lemma 2 states that t any round at most k different estimates values are sent by the processes. It is easy to see that this remains true when considering the algorithm of Fig. 3. This constitutes observation $O2$.

It follows from $O1$ and $O2$ that there are at most k different values present in the system at the end of r, i.e., $|EST[r]| \leq k$. As $EST[r+1] \subseteq EST[r]$, it follows from the claim that there are at most k values in the variables est_x when a process decides. As a value that is decided is a value that is in an est_i variable of a process p_i that participates in a round $r' \geq r$, it follows that no more than k different values are decided. $\square_{Lemma\ 4}$

Lemma 5 (*Early Decision*). *No process decides after the round* $R_f = \min\left(\lfloor \frac{f}{\Delta} \rfloor + 2, R_t\right)$, *where f denotes the number of processes that crash during the run.*

Proof. As for the base algorithm, the proof that no process executes more than R_t rounds follows from the code of the algorithm. Consequently, the proof consists in showing that no process decides after the round $\lfloor \frac{f}{\Delta} \rfloor + 2$. Let r_f be the first round that has a correct sender, i.e., $k \in SENDERS[r_f]$. As Δ is the number of processes that can send values in a round, it follows that $r_f \leq \lfloor \frac{f}{\Delta} \rfloor + 1$. We consider two cases.

- p_k receives a COMMIT message at a round $r \leq r_f - 1$. In that case, p_k decides during r (at line C1) and forwards the COMMIT message at the beginning of the round $r+1 \leq r_f \leq \lfloor \frac{f}{\Delta} \rfloor + 1$. It follows that all the processes receive this COMMIT message during the round $r+1$. Consequently, all the non-crashed processes that have not yet decided, decide at line C1 during the round $r+1 \leq r_f \leq \lfloor \frac{f}{\Delta} \rfloor + 1$.
- p_k does not receive a COMMIT message at a round $r \leq r_f - 1$. As p_k is correct and $k \in SENDERS[r_f]$, it sends its estimate value est_i to all the processes during the round r_f (line 08) and decides at line B3 during the round $r_f + 1$. Moreover, during the round $r_f + 1$, it sends a COMMIT message to all the processes (line B2) and halts (line B3). As p_k is correct, all the non-crashed processes receive this message during the round $r_f + 1$, and decide (if not yet done) during that round (line C1). The fact that $r_f + 1 \leq \lfloor \frac{f}{\Delta} \rfloor + 2$ completes the proof of that case. $\square_{Lemma\ 5}$

Theorem 5. *The algorithm described in Fig. 3 is a synchronous t-resilient k-set agreement algorithm. Moreover, no process decides after the round* $R_f = \min\left(\lfloor \frac{f}{\Delta} \rfloor + 2, R_t\right)$, *and a process halts at most one round after it has decided.*

Proof. The proof of the validity property is as in Theorem 1. The agreement property has been proved in Lemma 4. The early deciding property has been proved in Lemma 5. Finally, it follows from lines B2–B3 on one side, and C1 plus A1–A4 on the other side, that a process halts at most one round after it has decided. If a process decides at line 12', it decides and halts during the same round (namely, the round R_t). $\square_{Theorem\ 5}$

5.3. Remark

It is important to remark that, in the algorithm described in Fig. 3, a message carries only an estimate value or the constant value COMMIT. Said differently, no message carries information such as the number or the identities of crashed processes as currently known by the message sender. Using such an additional information could allow designing an algorithm in which a process both decides and halts by round $R_f = \min\left(\lfloor \frac{f}{\Delta} \rfloor + 2, \lfloor \frac{t}{\Delta} \rfloor + 1\right)$.

Please cite this article in press as: A. Mostéfaoui, et al., Narrowing power vs efficiency in synchronous set agreement: Relationship, algorithms and lower bound, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.002

6. Conclusion

The paper has investigated a new approach to circumvent the $\lfloor \frac{t}{k} \rfloor + 1$ lower bound associated with the k-set agreement problem in synchronous systems that can suffer up to t crash failures.

Assuming that the system is composed of clusters of m processes such that the ℓ-set agreement can be efficiently solved within each cluster (i.e., with a negligible cost with respect to the inter-cluster communication), the paper has shown that it is possible to solve the synchronous k-set agreement problem in $R_t = \left\lfloor \frac{t}{m\lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)} \right\rfloor + 1$ rounds (a round being counted as an inter-cluster communication). When considering early decision, it has been shown that a very simple addition of two statements to the base algorithm produces an early deciding algorithm in which no process decides and halts after $R_f = \min\left(\lfloor \frac{f}{\Delta} \rfloor + 2, \lfloor \frac{t}{\Delta} \rfloor + 1\right)$ rounds (where $\Delta = m\lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)$), f being the actual number of crashes in a run ($0 \leq f \leq t$).

The paper has also shown that the bound $R_t = \left\lfloor \frac{t}{m\lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)} \right\rfloor + 1$ is a lower bound. This shows an inherent tradeoff relating the "narrowing" power of the base objects that are used and the cost of any k-set synchronous agreement algorithm. In that sense, the paper generalizes the previous $R_t = \lfloor \frac{t}{k} \rfloor + 1$ lower bound that "implicitly" considers base object without narrowing power. In a very interesting way, this optimality proof relies on two important theorems of distributed computing. One (due to Gafni) is on the number of rounds of a synchronous algorithm that can be simulated in an asynchronous system prone to failures. The second (due to Herlihy and Rajsbaum) states an impossibility on asynchronously solving the k-set agreement problem from some base objects. In that sense, the paper shows another link connecting possibility/impossibility results in asynchronous systems and efficiency in synchronous systems.

This paper leaves open some problems (suggested by a referee). One concerns the lower bound for the early deciding case. We think that R_f is the corresponding lower bound but have no proof of it. Moreover, it is not clear if the technique used in the paper can be extended to proof it or if another technique has to be used. An answer to this question could be based on extending the topological approach that has been introduced in [15] to prove the classical $\min\left(\lfloor \frac{f}{k} \rfloor + 2, \lfloor \frac{t}{k} \rfloor + 1\right)$ lower bound.

Another open problem concerns the resilience of the underlying base $[m, \ell]$-SA objects. The paper has implicitly assumed that these objects are $(m-1)$-resilient (also called wait-free): an object invocation by a correct process always terminates (i.e., despite the crash of any subset of the $m-1$ other processes that access this object). A generalization would consist in considering more general $[m, \ell]$-SA objects, namely t'-resilient $[m, \ell]$-SA objects with $1 \leq t' \leq m-1$.

Acknowledgments

The authors want to thank Yoram Moses for constructive comments on a draft of this paper that helped improve both its presentation and its content. They also want to thank the referees for their constructive comments that helped improve the presentation and the content of the paper.

Appendix

n	Total number of processes
t	Upper bound on the number of faulty processes
f	Actual number of faulty processes
m	Number of processes accessing the same $[m, \ell]$-SA object
ℓ	Narrowing power of an $[m, \ell]$-SA object
k	Maximal number of values that can be decided

$$\alpha = \lfloor \frac{k}{\ell} \rfloor$$
$$\beta = k \bmod \ell$$
$$k = \alpha\ell + \beta$$
$$\Delta = \alpha\, m + \beta = m\lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)$$
$$R_t = \lfloor \frac{t}{\Delta} \rfloor + 1 = \left\lfloor \frac{t}{m\lfloor \frac{k}{\ell} \rfloor + (k \bmod \ell)} \right\rfloor + 1$$
$$R_f = \min\left(\lfloor \frac{f}{\Delta} \rfloor + 2, \lfloor \frac{t}{\Delta} \rfloor + 1\right)$$

References

[1] M.K. Aguilera, S. Toueg, A simple bivalency proof that t-resilient consensus requires $t+1$ rounds, Information Processing Letters 71 (1999) 155–178.
[2] M.K. Aguilera, G. Le Lann, S. Toueg, On the impact of fast failure detectors on real-time fault-tolerant systems, in: Proc. 16th Symposium on Distributed Computing, DISC'02, in: LNCS, vol. 2508, Springer-Verlag, 2002, pp. 354–369.
[3] H. Attiya, J. Welch, Distributed Computing, Fundamentals, Simulation and Advanced Topics, second edition, in: Wiley Series on Parallel and Distributed Computing, 2004, p. 414.
[4] B. Awerbuch, Complexity of network synchronization, Journal of the ACM 32 (4) (1985) 804–823.
[5] R. Bazzi, G. Neiger, Simplifying fault-tolerance: Providing the abstraction of crash failures, Journal of the ACM 48 (3) (2001) 499–554.
[6] E. Borowsky, E. Gafni, Generalized FLP impossibility results for t-resilient asynchronous computations, in: Proc. 25th ACM Symposium on Theory of Distributed Computing, STOC'93, ACM Press, 1993, pp. 91–100.
[7] B. Charron-Bost, A. Schiper, Uniform consensus is harder than consensus, Journal of Algorithms 51 (1) (2004) 15–37.
[8] S. Chaudhuri, More *choices* allow more *faults:* Set consensus problems in totally asynchronous systems, Information and Computation 105 (1993) 132–158.
[9] S. Chaudhuri, M. Herlihy, N. Lynch, M. Tuttle, Tight bounds for k-set agreement, Journal of the ACM 47 (5) (2000) 912–943.

Please cite this article in press as: A. Mostéfaoui, et al., Narrowing power vs efficiency in synchronous set agreement: Relationship, algorithms and lower bound, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.002

[10] D. Dolev, R. Reischuk, R. Strong, Early stopping in byzantine agreement, Journal of the ACM 37 (4) (1990) 720–741.
[11] M.J. Fischer, N.A. Lynch, A lower bound on the time to assure interactive consistency, Information Processing Letters 14 (4) (1982) 183–186.
[12] M.J. Fischer, N.A. Lynch, M.S. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM 32 (2) (1985) 374–382.
[13] E. Gafni, Round-by-round fault detectors: Unifying synchrony and asynchrony, in: Proc. 17th ACM Symposium on Principles of Distributed Computing, PODC'98, ACM Press, 1998, pp. 143–152.
[14] E. Gafni, R. Guerraoui, B. Pochon, From a static impossibility to an adaptive lower bound: The complexity of early deciding set agreement, in: Proc. 37th ACM Symposium on Theory of Computing, STOC 2005, ACM Press, 2005, pp. 714–722.
Q2 [15] R. Guerraoui, M. Herlihy, B. Pochon, A topological trzatment of early-deciding set-agreement, Theoretical Computer Science, 2009 (in press).
[16] M.P. Herlihy, Wait-free synchronization, ACM Transactions on Programming Languages and Systems 13 (1) (1991) 124–149.
[17] M.P. Herlihy, L.D. Penso, Tight bounds for *k*-set agreement with limited scope accuracy failure detectors, Distributed Computing 18 (2) (2005) 157–166.
[18] M.P. Herlihy, S. Rajsbaum, Algebraic spans, Mathematical Structures in Computer Science 10 (4) (2000) 549–573.
[19] M.P. Herlihy, N. Shavit, The topological structure of asynchronous computability, Journal of the ACM 46 (6) (1999) 858–923.
[20] I. Keidar, S. Rajsbaum, A simple proof of the uniform consensus synchronous lower bound, Information Processing Letters 85 (2003) 47–52.
[21] L. Lamport, M. Fischer, Byzantine generals and transaction commit protocols, Unpublished manuscript, 16 pages, April 1982.
[22] N.A. Lynch, Distributed Algorithms, Morgan Kaufmann Pub., San Francisco, CA, 1996, p. 872.
[23] A. Mostéfaoui, S. Rajsbaum, M. Raynal, Conditions on input vectors for consensus solvability in asynchronous distributed systems, Journal of the ACM 50 (6) (2003) 922–954.
[24] A. Mostéfaoui, S. Rajsbaum, M. Raynal, C. Travers, The combined power of conditions and failure detectors to solve asynchronous set agreement, SIAM Journal of Computing 38 (4) (2008) 1574–1601.
[25] A. Mostéfaoui, S. Rajsbaum, M. Raynal, Synchronous condition-based consensus, Distributed Computing 18 (5) (2006) 325–343.
[26] A. Mostéfaoui, M. Raynal, *k*-set agreement with limited accuracy failure detectors, in: Proc. 19th ACM Symposium on Principles of Distributed Computing, PODC'00, ACM Press, 2000, pp. 143–152.
[27] A. Mostéfaoui, M. Raynal, Randomized set agreement, in: Proc. 13th ACM Symposium on Parallel Algorithms and Architectures, SPAA'01, ACM Press, 2001, pp. 291–297.
[28] M. Raynal, Consensus in synchronous systems: A concise guided tour, in: Proc. 9th IEEE Pacific Rim Int'l Symposium on Dependable Computing, PRDC'02, IEEE Computer Press, 2002, pp. 221–228.
[29] M. Raynal, C. Travers, Synchronous set agreement: A concise guided tour (with open problems), in: Proc. 12th Int'l IEEE Pacific Rim Dependable Computing Symposium, PRDC'2006, IEEE Computer Press, 2006, pp. 267–274.
[30] M. Saks, F. Zaharoglou, Wait-free *k*-set agreement is impossible: The topology of public knowledge, SIAM Journal on Computing 29 (5) (2000) 1449–1483.
[31] N. Santoro, Design and Analysis of Distributed Algorithms, in: Wiley Series on Parallel and Distributed Computing, 2007, p. 589.
[32] X. Wang, Y.M. Teo, J. Cao, A bivalency proof of the lower bound for uniform consensus, Information Processing Letters 96 (2005) 167–174.

Please cite this article in press as: A. Mostéfaoui, et al., Narrowing power vs efficiency in synchronous set agreement: Relationship, algorithms and lower bound, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.002