Complexité du consensus anonyme en l'absence de concurrence. †

Claire Capdevielle[1], Colette Johnen[1], Petr Kuznetsov[2] et Alessia Milani[1]

[1] *Univ. Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France*
[2] *Télécom ParisTech, Paris, France*

Le consensus est l'une des abstractions fondamentales du distribué. En permettant à un ensemble de processus de se mettre d'accord sur l'une des valeurs qu'ils proposent, le consensus peut être utilisé pour implémenter, de manière cohérente et tolérante aux fautes, n'importe quel service distribué. Dans ce papier nous étudions la complexité du consensus anonyme en l'absence de concurrence : comptant le nombre d'emplacements mémoire et d'écritures lors d'une opération qui ne rencontre aucune concurrence. En supposant que les opérations privilégient les écritures et les lectures "simples" et ont recours à des primitives plus coûteuses, tel le CAS, seulement lorsque la concurrence est détectée, nous obtenons, pour ce type d'implémentation appelé "interval-solo-fast", une borne atteignable pour la complexité en espace.

Mots-clefs : complexité en temps, complexité en espace, borne inferieur, consensus, interval contention, solo-fast

1 Introduction

Consensus is one of the central distributed abstractions. Indeed, by enabling a collection of processes to agree on one of the values they propose, consensus can be used to implement any generic replicated service in a consistent and fault-tolerant way. However it is known that consensus cannot be solved in an asynchronous read-write shared memory system in a deterministic and fault-tolerant way [5, 13]. The difficulty stems from handling contended executions. One way to circumvent this impossibility is to only guarantee progress (using reads and writes) in executions meeting certain conditions, e.g., in the absence of *contention*. Alternatively, a process is guaranteed to decide in the *wait-free* manner, but stronger (and more expensive) synchronization primitives, such as *compare-and-swap*, can be applied in the presence of contention. We are interested in consensus algorithms in which a *propose* operation is allowed to apply primitives other than reads and writes on the base objects only in the presence of *interval contention*, i.e., when another *propose* operation is concurrently active. These algorithms are called *interval-solo-fast*.

Ideally, interval-solo-fast algorithms should have an optimized behavior in *uncontended* executions. Therefore, it appears natural to explore the *uncontended complexity* of consensus algorithms : how many memory operations (reads and writes) need to be performed and how many distinct memory locations need to be accessed in the absence of interval contention ?

In general, interval-solo-fast consensus can be solved with *constant* uncontended complexity [14]. To make things interesting, we focus here on *anonymous* consensus algorithms, i.e., algorithms not using process identifiers and, thus, programming all processes identically. Besides stimulating intellectual curiosity, the study of anonymous shared-memory algorithms is motivated by practical reasons discussed in [8].

Our results. We consider a standard asynchronous shared-memory model in which $n > 1$ processes communicate by applying atomic (or linearizable [11]) *primitive* operations on shared variables, called *base objects*. We assume that every base object maintains a *state* and exports a subset of the *Read*, *Write* and *Compare-And-Swap* (CAS) primitives. The primitive $Read(R)$ returns the state of R, and $Write(R,v)$ sets the state of R to v. The primitive $CAS(R,e,v)$ checks if the state of R is e and, if so, sets the state of R to v and returns

† Partially supported by the ANR project DISPLEXITY (ANR-11-BS02-014). This study has been carried out in the frame of the Investments for the future Programme IdEx Bordeaux-CPU (ANR-10-IDEX-03-02). The third author was supported by the ANR project DISCMAT, under grant agreement N ANR-14-CE35-0010-01.

Input-oblivious	Not input-oblivious	
$\Omega(\sqrt{n})$	$\Omega(\min(\sqrt{n}, \frac{\log m}{\log\log m}))$	
$O(\sqrt{n})$	if $\sqrt{n} \leq \frac{\log m}{\log\log m}$, $O(\sqrt{n})$	if $\sqrt{n} \geq \frac{\log m}{\log\log m}$, $O(\frac{\log m}{\log\log m})$ [14, 1]

TABLE 1: Space and solo-write complexity for anonymous interval-solo-fast consensus, where n is the number of processes and m is the number of input values that can be proposed

true ; otherwise, the state remains unchanged and *false* is returned. A *register* is a base object that exports only the Read and Write primitives.

On the lower-bound side, we show that any anonymous interval-solo-fast consensus algorithm exhibits non-trivial uncontended complexity that depends on n, the number of processes, and m, where m is the size of the set V of input values that can be proposed. More precisely some *propose* operation running *solo*, i.e., without any other process invoking *propose*, must write to $\Omega(\min(\sqrt{n}, \log m/\log\log m))$ distinct memory locations. This metrics, which we call *solo-write complexity*, is upper-bounded by the *step complexity* of the algorithm, i.e., the worst-case number of all base-object primitives applied by an individual operation. In the special case of *input-oblivious* algorithms, where the sequence of memory locations written in a solo execution does not depend on the input value, we derive a stronger lower bound of $\Omega(\sqrt{n})$ on solo-write complexity. Formally,

Theorem 1 *Any n-process m-valued interval-solo-fast anonymous consensus algorithm must have space complexity* $\Omega(\min(\sqrt{n}, \log m/\log\log m))$ *and solo-write complexity* $\Omega(\min(\sqrt{n}, \log m/\log\log m))$. *Moreover, if the algorithm is input-oblivious, then the bounds become* $\Omega(\sqrt{n})$.

Our proof only requires the algorithm to ensure that operations terminate in solo executions, so the lower bounds also hold for *abortable* [2, 9] and obstruction-free [10] consensus implementations.

On the positive side, we show that our lower bound is tight. Our matching consensus algorithms are based on our novel *value-splitter* abstraction, extending the classical *splitter* mechanism [12, 15, 3], interesting in its own right. This new abstraction and our algorithms are explained in section 2.

Overall, our results, summarized in Table 1, imply the first nontrivial *tight* lower bound on the uncontended space complexity for consensus known so far, complementing a recent result on the space complexity of *solo-terminating* anonymous consensus [6]. ‡ Our results also show that there is an inherent gap between anonymous and non-anonymous consensus algorithms : recall that non-anonymous consensus has constant uncontended complexity [14].

Related work. The idea of optimizing concurrent algorithms for uncontended executions was suggested by Lamport in his "fast" mutual exclusion algorithm [12].

Attiya et al. [2] showed that any *step-solo-fast* (where operations only apply reads and writes in the absence of interleaving steps) consensus either use $O(\sqrt{n})$ space or incur $O(\sqrt{n})$ memory *stalls* per operation. No step-solo-fast algorithm matching this lower bound is known so far : existing algorithms typically expose $O(n)$ space complexity. Recently Gelashvili [6](for the anonymous case), and Zhu [16] (for the non-anonymous case) have shown that any solo-terminating (and, as a result, obstruction-free) read-write consensus protocol must use $\Omega(n)$ registers. These bounds are tight [8]. These lower bounds focus on *step contention* and do not extend to uncontended executions, where no interval contention is encountered.

Aspnes and Ellen [1] showed that any anonymous consensus protocol has to execute $\Omega(\min(n, \log m/\log\log m))$ steps in solo executions. Our consensus algorithms have also asympotically optimal step complexity.

Our *value-splitter* abstraction is inspired by the splitter mechanism in [15, 3], originally suggested by Lamport [12]. The novel input-oblivious value-splitter implementation we present is inspired by the obstruction-free leader election algorithm proposed by Giakkoupis et al. [7].

2 Optimal interval-solo-fast consensus

Our interval-solo-fast consensus algorithm is similar to the *splitter-based* consensus algorithm in [14], except that we replace the `splitter` object with the `value-splitter` object.

‡. Informally, a solo-terminating algorithm ensures that every process running solo from any configuration eventually terminates.

Definition 2 *A value-splitter supports a single operation, split() taking a parameter in V and returning a boolean response, and ensures that, for all $v, v' \in V$, and in every execution :*

1. ***VS-Agreement.*** *If invocations split(v) and split(v') return true, then $v = v'$.*
2. ***VS-Solo execution.*** *If a split(v) operation completes before any other split(v') operation is invoked, then it returns true.*

In the following we first describe our consensus algorithm (the pseudocode can be found in [4]), then we illustrate two anonymous interval-solo-fast implementations of a value-splitter, which provide a matching upper bound to our lower bound. Due to space limitation refer to [4] for the proofs.

Consensus using value-splitter. The value decided by the consensus is written in a variable D, initially $\perp \notin V$. The first steps by a process p are to check if D stores a non-$\perp$ value and if yes, return this value. Otherwise, the process accesses the value-splitter object VS. If it obtains *true* from its invocation of $VS.split(v)$, p writes its input value v in a register F. Then, it reads a register Z to check if some other process has detected contention and if the value of Z is *false* (no contention) p decides its own value. Before returning the decided value, process p writes it in D. The write primitives on F and D, with a read of Z in between are intended to ensure that either process p detects that some other process is around and resorts to applying a CAS primitive on D, or the contending process adopts the input value of p.

If p obtains *false* from the value-splitter, it sets Z to *true* (contention is detected). Recall that this may happen if more than one process accessed the value-splitter, regardless of their input values. Then, p reads register F and, if F stores a non-$\perp$ value, adopts the value as its current proposal. Finally, it applies the CAS primitive on D with its proposal and decides the value read in D.

Our consensus algorithm incurs only a constant overhead with respect to the implementation of the value-splitter it uses and is interval-solo-fast assuming that the underlying value-splitter is interval-solo-fast.

Input-oblivious value-splitter. Algorithm 1 describes our anonymous and input-oblivious implementation of a value-splitter. The algorithm only uses an array R of k registers where $k^2 - 3k + 6 > 2n$ and is, trivially, interval-solo-fast. A process p performing operation $split(v)$ tries to write its input value to registers $R[0], \ldots, R[k-1]$. Each time, before writing to $R[i]$, p reads $i+1$ registers to verify that $R[0], \ldots, R[i-1]$ store v and $R[i]$ stores the initial value $\perp$. If this is not the case, contention is detected and the operation returns *false*. After the last write to $R[k-1]$, the operation returns *true*.

```
   Procedure: split(v)                                  Shared variables:
1  Lastwritten := −1 ;                                  Array of registers R[0...k−1] with k^2 − 3k + 6 > 2n.
2  while (Lastwritten ≤ k − 1) do                       Initially ⊥
3      for i := 0 ; i ≤ Lastwritten ; i++ do
4          if Read(R[i]) ≠ v then return false
5      end
6      if Read(R[Lastwritten + 1]) ≠ ⊥ then return false;
7      Lastwritten++;
8      Write(R[Lastwritten], v) ;
9  end
10 return true;
```

Algorithm 1: Anonymous and input-oblivious value-splitter

Theorem 3 *Algorithm 1 is an interval-solo-fast anonymous input-oblivious implementation of a value-splitter with solo-write and space complexities in $O(\sqrt{n})$.*

Non-input-oblivious value-splitter. A trivial adaptation of the `weak conflict-detector` proposed in [1] implements an interval-solo-fast value-splitter that exhibits $O(\log m / \log\log m)$ complexity (Algorithm 2).

The algorithm uses an array R of k registers, where $k! = m$. Each input value v of a *split* operation determines a unique permutation π_v of the registers in R that is used as the order in which the processes access the registers. In its i-th access, a process executing $split(v)$ first reads register $R[\pi_v(i)]$; if $\perp$ is read, the process

```
   Procedure: split(v)                          Shared variables:
1  for i := 1..k do                             Registers R[1..k], initially ⊥
2  |  t := Read(R[π_v(i)]);
3  |  if t = ⊥ then Write(R[π_v(i)], v);
4  |  if t ≠ v then return false;
5  end
6  return true;
```

Algorithm 2: Non-input-oblivious value-splitter

writes v to it ; If a value $v' \neq v$ is read, it returns *false* (contention is detected). If the process succeeds in writing v in all registers prescribed by π_v, it returns *true*.

Theorem 4 *Algorithm 2 implements anonymous interval-solo-fast m-valued value-splitter with solo-write and space complexity in $O(\log m / \log\log m)$.*

Références

[1] J. Aspnes and F. Ellen. Tight bounds for adopt-commit objects. *Theory of Computing Systems*, 55(3) :451–474, 2014.

[2] H. Attiya, R. Guerraoui, D. Hendler, and P. Kuznetsov. The complexity of obstruction-free implementations. *J. ACM*, 56(4), 2009.

[3] H. Buhrman, J. A. Garay, J.-H. Hoepman, and M. Moir. Long-lived renaming made fast. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '95, pages 194–203, 1995.

[4] C. Capdevielle, C. Johnen, P. Kuznetsov, and A. Milani. On the Uncontended Complexity of Anonymous Consensus. In *OPODIS 2015*, Rennes, France, Dec. 2015.

[5] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2) :374–382, Apr. 1985.

[6] R. Gelashvili. On the Optimal Space Complexity of Consensus for Anonymous Processes. In *DISC*, Oct. 2015.

[7] G. Giakkoupis, M. Helmi, L. Higham, and P. Woelfel. An o(sqrt n) space bound for obstruction-free leader election. In *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, pages 46–60, 2013.

[8] R. Guerraoui and E. Ruppert. Anonymous and fault-tolerant shared-memory computing. *Distributed Computing*, 20(3) :165–177, 2007.

[9] V. Hadzilacos and S. Toueg. On deterministic abortable objects. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 4–12, New York, NY, USA, 2013. ACM.

[10] M. Herlihy, V. Luchangco, and M. Moir. Obstruction-free synchronization : Double-ended queues as an example. In *ICDCS*, pages 522–529, 2003.

[11] M. Herlihy and J. M. Wing. Linearizability : A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3) :463–492, 1990.

[12] L. Lamport. A fast mutual exclusion algorithm. *ACM Trans. Comput. Syst.*, 5(1) :1–11, Jan. 1987.

[13] M. Loui and H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4 :163–183, 1987.

[14] V. Luchangco, M. Moir, and N. Shavit. On the uncontended complexity of consensus. In F. Fich, editor, *Distributed Computing*, volume 2848 of *Lecture Notes in Computer Science*, pages 45–59. Springer Berlin Heidelberg, 2003.

[15] M. Moir and J. H. Anderson. Wait-free algorithms for fast, long-lived renaming. *Sci. Comput. Program.*, 25(1) :1–39, Oct. 1995.

[16] L. Zhu. A tight space bound for consensus. 2016. `http://www.cs.toronto.edu/~lezhu/`.