Brief Announcement: Wait-Free Universality of Consensus in the Infinite Arrival Model

Grégoire Bonin
LS2N, Université de Nantes, France
gregoire.bonin@etu.univ-nantes.fr

Achour Mostéfaoui
LS2N, Université de Nantes, France
achour.mostefaoui@univ-nantes.fr

Matthieu Perrin
LS2N, Université de Nantes, France
matthieu.perrin@univ-nantes.fr

Abstract

In classical asynchronous distributed systems composed of a fixed number n of processes where some proportion may fail by crashing, many objects do not have a wait-free linearizable implementation (e.g. stacks, queues, etc.). It has been proved that consensus is universal in such systems, which means that this system augmented with consensus objects allows to implement any object that has a sequential specification. In this paper, we consider a more general system model called infinite arrival model where infinitely many processes may arrive and leave or crash during a run. We prove that consensus is still universal in this more general model. For that, we propose a universal construction based on a weak log that can be implementated using consensus objects.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Concurrent object, Consensus, Infinite arrival model, Linearizability, Universal construction, Wait-freedom

Digital Object Identifier 10.4230/LIPIcs.DISC.2019.38

Related Version A full version of the paper is available at `https://arxiv.org/abs/1908.02063`.

Funding This work was partially supported by the French ANR project 16-CE25-0005 O'Browser devoted to the study of decentralized applications on Web browsers.

1 Introduction

Maurice Herlihy proved in [3] that consensus is universal in classical distributed systems composed of a set of n processes. Namely, any object having a sequential specification has a wait-free and linearizable implementation using only read/write registers (memory locations) and some number of consensus objects. For proving the universality of consensus, Herlihy introduced the notion of universal construction. It is a generic algorithm that, given a deterministic sequential specification of any object, provides a concurrent implementation of this object. Since then, many universal constructions have been proposed for several objects [5], assuming the availability of hardware special instructions that provide the same computing power as consensus, like compare&swap, Load-Link/Store-Conditional etc.

This last decade, first with peer-to-peer systems, and then with multi-core machines and the multi-threading model, the assumption of a closed system with a fixed number n of processes and where every process knows the identifiers of all processes became too restrictive. Hence the infinite arrival model introduced in [4]. In this model, any number of processes can crash (or leave, in a same way as in the other model), but any number (be it finite or not) of processes can also join the network. When a process joins such a system, it is not known to the already running processes, so no fixed number of processes can be used in the implementations as a parameter. Let us note that, at any time, the number of processes that have already joined the system is finite, but can be infinitely growing.

Problem statement. The aim of this paper is to extend universality of consensus to the infinite arrival model. The question is thus "is it possible to build a universal wait-free linearizable construction based on consensus objects and read/write atomic registers?" This is not trivial for different reasons. First, although the lock-free universal constructions still work in the infinite arrival model because they ensure a global progress condition, this is no more the case for wait-free universal constructions. Second, wait-free implementations rely on what is called help mechanism, that has been recently formalized in [1]. This mechanism requires any process, before terminating its operation, to help processes having pending operations, in order to reach wait-freedom. One of the difficulties in the infinite arrival model is that helping is not obvious. Indeed, helping requires at least that a process needing to be helped is able to announce its existence to other processes willing to help it. Due to the infinite number of potential participating processes over time, it is not reasonable to assume that each process can write in a dedicated register, and to require helping processes to read them all. When only consensus and read/write registers are accessible to a process, a newly arriving process must compete with a potentially infinite number of other arriving processes on either a consensus object or a same memory location; and may fail on all its attempts.

2 The Weak Log Abstraction

Similarly to [2] which first proposes a Collect object that will be used as a building block for a universal construction, we propose a weak log object that is used as a list of presence where a process that arrives registers. A weak log can then be used in Herlihy's universal construction [3] instead of the array of registers to achieve wait-freedom. In an instance of the weak log, each process p_i proposes a value through an operation `append`(v_i), that returns the sequence of all the values previously appended. The weak log is wait-free but not linearizable. Instead, it is specified by the following properties.

▶ **Definition 1** (weak log). *All processes $p_0, p_1, \ldots$ propose distinct values $v_0, v_1, \ldots$ by invoking* `append`(v_i)*, that returns a finite sequence $w_i = w_{i,1} \cdot w_{i,2} \cdots w_{i,|w_i|}$ such that:*
Validity. *All values in a sequence w_i have been appended by some process.*
Suffixing. *The last value of the sequence returned by p_i is its own.*
Total order. *All pairs of values contained in both w_i and w_j appear in the same order.*
Eventual visibility. *If p_i terminates, finitely many returned sequences do not contain v_i.*
Wait-freedom. *No process takes an infinite number of steps in an execution.*

The main difficulty in the implementation of a weak log lies in the allocation of one memory location per process, where it can safely announce its invoked operation. As it is impossible to allocate an infinite array at once, it is necessary to build a data structure in which processes allocate their own piece of memory, and make it reachable to other processes, by winning a consensus. A linked list in which processes compete to append their value at the end follows a similar pattern, but it poses a challenge: as an infinite number of processes access the same sequence of consensus objects, one process may loose all its attempts to insert its own node, breaking wait-freedom.

Algorithm 1 solves this issue by using a novel feature, that we call *passive helping*: when a process wins a consensus, it creates a side list to host values of processes concurrently competing on the same consensus object. As only a finite number of processes have arrived

Algorithm 1 Wait-free weak log using consensus.

```
1 operation append($v$) is:
2     $\mathsf{node}_i \leftarrow$ last.read().propose($\langle\langle v, \bot\rangle, \bot\rangle$) ;          // add $v$ to the log
3     last.write($\mathsf{node}_i$.tail);
4     while $\mathsf{node}_i$.head $\neq v$ do  $\mathsf{node}_i \leftarrow \mathsf{node}_i$.tail.propose($\langle v, \bot\rangle$);
5     $\mathsf{log}_i \leftarrow \varepsilon$; $\mathsf{list}_i \leftarrow$ first; $\mathsf{node}_i \leftarrow \mathsf{list}_i$.head;          // read the log
6     while true do
7         $\mathsf{log}_i \leftarrow \mathsf{log}_i \oplus \mathsf{node}_i$.head;
8         if $\mathsf{node}_i$.head $= v$ then return $\mathsf{log}_i$;
9         $\mathsf{node}_i \leftarrow \mathsf{node}_i$.tail; if $\mathsf{node}_i = \bot$ then  $\mathsf{list}_i \leftarrow \mathsf{list}_i$.tail; $\mathsf{node}_i \leftarrow \mathsf{list}_i$.head;
```

in the system when the consensus is won, a finite number of processes will try to insert their value in the side list, which ensures termination.

Processes executing Algorithm 1 build a linked list of linked lists of nodes of the form $\langle$list.head, list.tail$\rangle$ where list.tail is a consensus object that references nodes of the same form, and list.head $= \langle$node.head, node.tail$\rangle$ is a node of the side list, where node.head is a value appended by some process and node.tail is a consensus object accepting values of the same type as list.head. Processes share a consensus object, first, that references the first node of the list of lists, and a read/write register, last, that references a consensus object list.tail.

In absence of concurrency, last references the end of the list starting with first. However, as the consensus and the write on lines 2 and 3 are not done atomically, a very old value can be written in last, in which case its value could move backward. The central property of the algorithm is that last eventually moves forward, allowing very slow processes to find some place in a side list.

3 Conclusion

Consensus is a central problem in distributed computing, because it allows wait-free linearizable implementations of all objects with a sequential specification, in systems composed of n asynchronous processes that may crash. In this paper, we asked the question of whether the result still holds in the infinite arrival model, in which a potentially infinite number of processes can arrive and leave during an execution. We answered this question positively by introducing a weak log abstraction, that can be implemented using only consensus objects and read/write registers and can be used in a wait-free and linearizable universal construction.

References

1 Keren Censor-Hillel, Erez Petrank, and Shahar Timnat. Help! In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 241–250. ACM, 2015.

2 Panagiota Fatourou and Nikolaos D. Kallimanis. Highly-Efficient Wait-Free Synchronization. *Theory Comput. Syst.*, 55(3):475–520, 2014.

3 Maurice Herlihy. Impossibility and Universality Results for Wait-Free Synchronization. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada, August 15-17, 1988*, pages 276–290, 1988.

4 Michael Merritt and Gadi Taubenfeld. Resilient consensus for infinitely many processes. In *International Symposium on Distributed Computing*, pages 1–15. Springer, 2003.

5 Michel Raynal. Distributed Universal Constructions: a Guided Tour. *Bulletin of the EATCS*, 121, 2017.