

# Easy Consensus Algorithms for the Crash-Recovery Model

Felix C. Freiling, Christian Lambertz, and Mila Majster-Cederbaum

Department of Computer Science, University of Mannheim, Germany

*Problem Setting.* One of the most popular failure models for asynchronous fault-tolerant distributed systems is called *crash-stop*, which allows that a certain number of processes stops executing steps during the computation. Despite its theoretical interest, crash-stop is not expressive enough to model many realistic scenarios. In practice, processes crash but their processors reboot and the crashed process is restarted from a recovery point and rejoins the computation. This behavior is formalized as a failure model called *crash-recovery*, in which the processes can crash and recover multiple times.

Crash-recovery is a strict generalization of crash-stop and thus, any impossibility result for crash-stop also holds in crash-recovery. However, algorithms designed for crash-stop will not necessarily work in crash-recovery due to the additional behavior. While in crash-stop processes are usually classified into two distinct classes (those which eventually crash and those which do not), in crash-recovery we have four distinct classes of processes: (1) *always up* (processes that never crash), (2) *eventually up* (processes that crash at least once but eventually recover and do not crash anymore), (3) *eventually down* (processes that crash at least once and eventually do not recover anymore), and (4) *unstable* (processes that crash and recover infinitely often). (1) and (2) are called *correct* processes, and (3) and (4) *incorrect*. Since processes usually lose all state information when they crash, the notion of *stable storage* was invented to model a type of expensive persistent storage to access pre-crash data.

We choose the problem of *consensus* as benchmark problem to study the differences between crash-stop and crash-recovery. Roughly speaking, consensus requires that processes agree on a common value from a set of input values. Consensus is fundamental to many fault-tolerant problems but has mainly been studied in crash-stop. We use the *failure detector* abstraction (FD) to help solve consensus. A FD gives information about the failures of processes. We look at two classes of FDs: the class of *perfect FDs* ( $\mathcal{P}$ ) that tell exactly who has failed, and the class of *eventually perfect FDs* ( $\diamond\mathcal{P}$ ) that can make finitely many mistakes about who has failed.

*Contributions.* Similarly to Aguilera, Chen and Toueg [1], we study consensus in the crash-recovery model under varying assumptions. However, our approach is to re-use existing algorithms from crash-stop in a modular way and to improve the comprehensibility. One main task of our algorithms is to partly *emulate* a crash-stop system on top of a crash-recovery system to be able to run a crash-stop consensus algorithm.

Table 1 summarizes the cases we study along three dimensions: (1) the availability of stable storage (large columns), (2) a process state assumption (rows of the table), and (3) the availability of FDs (sub-columns within large columns). Impossibility results are denoted by “×” and solvability by “√”. Impossibility results with stronger

**Table 1.** Overview of our results [2]. An arrow depicts a logical implication

assumptions	no stable storage		stable storage	
	$\diamond\mathcal{P}$	$\mathcal{P}$	$\diamond\mathcal{P}$	$\mathcal{P}$
one correct	×	×	×	×
correct majority	×	×	✓	✓
one always up	×	✓	×	✓
correct majority & one always up	×	✓	✓	✓
more always up than incorrect	✓	✓	✓	✓
always up majority	✓	✓	✓	✓

assumptions imply impossibility for cases with weaker ones, while solvability with weak assumptions implies solutions with stronger ones (indicated by arrows). The section and theorem numbers in the table refer to the relevant parts in the full version of this work [2].

The case of  $\diamond\mathcal{P}$  and more always up than incorrect processes was proven to be the weakest for consensus [1] and thus, all weaker process state assumptions are impossible. We first focus on  $\mathcal{P}$  and the unavailability of stable storage. We argue that at least one always up process is necessary (Sect. 4.1) and sufficient (Theorem 1). Then we weaken  $\mathcal{P}$  to  $\diamond\mathcal{P}$  and present a modular algorithm for the always up majority of processes case (Sect. 4.3). We then turn to the case where processes are allowed to use stable storage. We first prove two impossibility results regarding the presence of correct processes. The first one arises in the case where we have only  $\diamond\mathcal{P}$  and one always up process (Lemma 1) and the second in the case where  $\mathcal{P}$  is available and one correct process is present (Lemma 2). We then give an algorithm for the remaining case (Theorem 2): It uses  $\diamond\mathcal{P}$ , a majority of correct processes, and some minimal insight about the used crash-stop consensus algorithm which needs to be saved on stable storage.

**References**

1. Aguilera, M.K., Chen, W., Toueg, S.: Failure detection and consensus in the crash recovery model. *Distributed Computing* 13(2), 99–125 (2000)
2. Freiling, F.C., Lambertz, C., Majster-Cederbaum, M.: Easy Consensus Algorithms for the Crash-Recovery Model. Tech. Report, TR-2008-002, Univ. of Mannheim (2008)