Separation and Equivalence results for the Crash-stop and Crash-recovery Shared Memory Models

Ohad Ben-Baruch
Ben-Gurion University, Israel
ohadben@cs.bgu.ac.il

Srivatsan Ravi
University of Southern California, USA
srivatsr@usc.edu

Summary. Linearizability, the traditional correctness condition for concurrent objects is considered insufficient for the non-volatile shared memory model where processes recover following a crash. For this crash-recovery shared memory model, strict-linearizability is considered appropriate since, unlike linearizability, it ensures operations that crash take effect prior to the crash or not at all. This work formalizes and answers the question of whether an implementation of a data type derived for the crash-stop shared memory model is also strict-linearizable in the crash-recovery model.

We present a rigorous study to prove how helping mechanisms, typically employed by non-blocking implementations, is the algorithmic abstraction that delineates linearizability from strict-linearizability. Our first contribution formalizes the crash-recovery model and how explicit process crashes and recovery introduces further dimensionalities over the standard crash-stop shared memory model. We make the following technical contributions: (i) we prove that strict-linearizability is independent of any known help definition; (ii) we present a natural definition of help-freedom to prove that any obstruction-free, linearizable and help-free implementation of a total object type is also strict-linearizable; (iii) we prove that for a large class of object types, a non-blocking strict-linearizable implementation cannot have helping. Viewed holistically, this work provides the first precise characterization of the intricacies in applying a concurrent implementation designed for the crash-stop (and resp. crash-recovery) model to the crash-recovery (and resp. crash-stop) model. A full version can be found in [4] [1].

Overview. Concurrent data structures for the standard volatile shared memory model typically adopt linearizability as the traditional safety property [7]. However, in the non-volatile shared memory model where processes *recover* following a crash, linearizability is considered insufficient since it allows object operations that crash to take effect anytime in the future. In the crash-recovery model [5], linearizability is strengthened to force crashed operations to take effect before the crash or not take effect at all, so-called *strict-linearizability* [1]. While there exists a well-studied body of linearizable data structure implementations in the crash-stop model [8], concurrent implementations in the crash-recovery model are comparatively nascent. Consequently, it is natural to ask: under what conditions is a linearizable implementation in the crash-stop also strict-linearizable in the crash-recovery model?

Non-blocking implementations in the crash-stop model employ *helping*: i.e., apart from completing their own operation, processes perform additional work to help *linearize* concurrent operations and make progress. This helping mechanism enables an operation invoked by a process p_i to be linearized by the event performed of another process p_j, but possibly after the crash of p_i. However, strict-linearizability stipulates that the operation invoked by p_i be linearized before the crash event. Intuitively, this suggests that linearizable implementations that are *help-free* must be strict-linearizable (also conjectured in [5]). This work formalizes and answers this precise question: whether a help-free implementation of a data type derived for the crash-stop model can be used *as it is* in the crash-recovery model.

Precisely answering this question necessitates the formalization of the crash-recovery shared memory model. Explicit process crashes introduces further dimensionalities to the set of executions admissible in the crash-recovery model over the well formalized crash-stop shared memory [3]. In the *individual crash-recovery model* processes may crash on an individual basis, i.e., an event in the execution corresponds to the crash of a single process, while in the *full-system crash-recovery model* a crash event corresponds to the crash of all processes participating in the concurrent implementation. Following a crash event, the local state of any crashed process is reset to its initial state when it recovers and restarts an operation assuming the *old identifiers crash-recovery model* (and resp. *new identifiers crash-recovery model*) with the original process identifier (and resp. new process identifier). Our contributions establish equivalence and separation results for crash-stop and the identified crash-recovery models, thus providing a precise characterization of the intricacies in applying a concurrent implementation designed for the crash-stop model to the crash-recovery model, and vice-versa.

Contributions. We define the crash-recovery model and its characteristics. We show that there exist sequential implementations of object types in the crash-stop model that have inconsistent sequential specifications in the old identifiers crash-recovery model.

We consider how data structures use helping in the crash-stop model by adopting the definitions of *linearization-helping* [6] and *universal-helping* [2]. When considering an execution with two concurrent operations, the linearization of these operations dictates which operation take effect first. The definition of linearization-helping considers a specific event e, in which it is *decided* which operation is linearized first. In an implementation that does not have linearization-helping, e is an event by the process whose operation is decided to be the one that comes first. Universal-helping requires that the progress of some processes eventually ensures that all pending invocations are linearized, thus forcing a process to ensure concurrent operations of other processes are eventually linearized.

The first technical contribution of this paper is proving that the following pairs of conditions are incomparable. That is, an implementation can satisfy exactly one of them, both, or none.

- linearization-helping vs. universal-helping
- strict-linearizability vs. linearization-helping

[1] paper available at https://arxiv.org/abs/2012.03692

The second technical contribution is to show that there exists a correlation in the above pairs under certain restrictions.

- Restricting the definition of linearization-helping to be *prefix-respecting*, we prove that linearization-help free implies strict-linearizability. More specifically, any *obstruction-free* implementation of a *total* object type that is linearizable and has no linearization-helping in the crash-stop model is also strict-linearizable in the new identifiers individual crash-recovery model.
- We prove that any non-blocking implementation of an *order-dependent* type that is strict-linearizable in the crash-recovery model has no universal-helping in the crash-stop model.

List of Results.

Lemma 1. *There exists an implementation of a data type that satisfies universal-helping (resp. linearization-helping), but does not satisfy linearization-helping (resp. universal-helping).*

The next set of results proves that strict-linearizability is independent of both linearization-helping and universal-helping.

Claim 1. *There exists a wait-free strict-linearizable implementation A of an object type τ in the individual crash-recovery model, such that A has universal-helping and it is linearization-help free in the crash-stop model.*

Claim 2. *There exists an implementation A of an object type τ such that A is linearizable, wait-free and has both linearization-helping and universal-helping in the crash-stop model. Moreover, A is strict-linearizable in the individual crash-recovery model.*

Claim 3. *There exists a wait-free linearizable implementation I of an object type τ such that I is linearization-help free and universal-help free in the crash-stop model, while I is not strict-linearizable in the system-wide crash-recovery model.*

Lemma 2. *Strict-linearizability and linearization-helping are independent*

Lemma 3. *Strict-linearizability and universal-helping are independent*

Lemma 4 below discuss the relation between strict-linearizability and linearization-helping under the restriction of prefix-respecting linearization functions. Roughly speaking, prefix-respecting requires that if we linearize operation π_1 before π_2 in some execution, then we must do the the same in any extension of the execution.

Lemma 4. *Let A be an obstruction-free implementation of a total object type τ such that A is not strict-linearizable in the individual crash-recovery model. Then, any prefix-respecting linearization function f of A imply linearization-helping in the crash-stop model.*

Corollary 1. *Let A be an obstruction-free implementation of a total object type τ such that A is linearizable and linearization-help free in the crash-stop model. Then A is strict-linearizable in the individual crash-recovery model.*

In Lemma 5 we prove that for a large set of object types called order-dependent strict-linearizability implies there is no universal-helping. In a nutshell, an object type is order-dependent if there exists an infinite sequence of operations H and two operations π_1, π_2 such that adding one of them or both to H effects the response of some operation in H. Moreover, if we add both before H the order in which we add them effects an operation response in H.

Lemma 5. *Let A be a non-blocking implementation of an order-dependent total type τ, such that A is strict-linearizable in the system-wide crash-recovery model. Then A is linearizable and universal-help free in the crash-stop model.*

Corollary 2. *Let A be a non-blocking implementation of an order-dependent total object τ such that A has universal-helping in the crash-stop model. Then A is not strict-linearizable in the system-wide crash-recovery model.*

Discussion of results. The model presented in this paper is an abstract model in which all writes are immediately persistent. This is useful for exploring the limitations of the crash-recovery model, and derive lower-bounds and impossibility results. However, real-world machines introduce another layer of complexity, since caches are volatile. In such machines, a data that has been written to main memory but is yet to be persisted is lost in case of a crash. Therefore, there is a need to carefully and manually regulate eviction of cache lines to main memory in order to avoid critical data loss. Izraelevitz et al. [9] defined and studied such a model, called explicit epoch persistency. In addition, [9] proposed a general durability transformation to transform any implementation from the abstract model to the more realistic model with volatile cache. Although the transformation has been proven to satisfy durable-linearizability, a similar transformation can be used for strict-linearizability. Therefore, our algorithmic results holds also for the explicit epoch persistency model.

REFERENCES

[1] Marcos K. Aguilera and S. Frølund. 2003. Strict Linearizability and the Power of Aborting.

[2] Hagit Attiya, Armando Castañeda, and Danny Hendler. 2018. Nontrivial and universal helping for wait-free queues and stacks. *J. Parallel Distributed Comput.* 121 (2018), 1–14. https://doi.org/10.1016/j.jpdc.2018.06.004

[3] Hagit Attiya and Jennifer Welch. 1998. *Distributed Computing. Fundamentals, Simulations, and Advanced Topics.* McGraw-Hill.

[4] Ohad Ben-Baruch and Srivatsan Ravi. 2020. Separation and Equivalence results for the Crash-stop and Crash-recovery Shared Memory Models. (2020). arXiv:cs.DC/2012.03692

[5] Ryan Berryhill, Wojciech M. Golab, and Mahesh Tripunitara. 2015. Robust Shared Objects for Non-Volatile Main Memory. In *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France.* 20:1–20:17. https://doi.org/10.4230/LIPIcs.OPODIS.2015.20

[6] Keren Censor-Hillel, Erez Petrank, and Shahar Timnat. 2015. Help!. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015.* 241–250. https://doi.org/10.1145/2767386.2767415

[7] Maurice Herlihy. 1991. Wait-free synchronization. 13, 1 (1991), 123–149.

[8] Maurice Herlihy and Nir Shavit. 2008. *The art of multiprocessor programming.* Morgan Kaufmann. I–XX, 1–508 pages.

[9] Joseph Izraelevitz, Hammurabi Mendes, and Michael L. Scott. 2016. Linearizability of Persistent Memory Objects Under a Full-System-Crash Failure Model. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings.* 313–327. https://doi.org/10.1007/978-3-662-53426-7_23