CCIT Report #697 April 2008

# Fork Sequential Consistency is Blocking

Christian Cachin* Idit Keidar† Alexander Shraer†

May 14, 2008

**Abstract**

We consider an untrusted server storing shared data on behalf of clients. We show that no storage access protocol can on the one hand preserve sequential consistency and wait-freedom when the server is correct, and on the other hand always preserve fork sequential consistency.

## 1 Introduction

We examine an online collaboration facility providing storage and data sharing functions for remote clients that do not communicate directly [3, 4, 13, 14]. Specifically, we consider a server that implements single-writer multi-reader registers. The storage server may be faulty, potentially exhibiting Byzantine faults [10, 8, 11, 2]. When the server is correct, strong liveness, namely *wait-freedom* [5], should be guaranteed, as a client editing a document does not want to be dependent on another client, which could even be in a different timezone [14]. In addition, although read/write operations of different clients may occur concurrently, consistency of the shared data should be provided. Specifically, we consider a service that, when the server is correct, provides *sequential consistency*, which ensures that clients have the same *view* of the order of read/write operations, which also respects the local order of operations occurring at each client [7]. Sequential consistency provides clients with a convenient abstraction of a shared storage space. It allows for more efficient implementations than stronger consistency conditions such as linearizability [6], especially when the system is not synchronized [1].

In executions where the server is faulty, liveness obviously cannot be guaranteed. Moreover, with a Byzantine server, ensuring sequential consistency is also impossible [2]. Still, it is possible to guarantee weaker semantics, in particular so-called *forking* consistency notions [8, 10]. These ensure that whenever the server causes the views of two clients to differ in a single operation, the two clients never again see each other's updates after that. In other words, if an operation appears in the views of two clients, these views are identical up to this operation.

Originally, *fork-linearizability* was considered [8, 10, 2]. In this paper, we examine the weaker *fork sequential consistency* condition, recently introduced by Oprea and Reiter [11], who showed that this new condition is sufficient for certain applications. However, to date, no fork-sequentially-consistent storage protocol has been proposed. In fact, Oprea and Reiter suggested this as a future research direction [11]. Furthermore, Cachin et al. [2] showed that the stronger notion of fork-linearizability does not allow for wait-free implementations, but conjectured that such implementations might be possible with fork sequential consistency. Surprisingly, we prove here that no storage access protocol can provide fork sequential consistency at all times and also be sequentially consistent and wait-free whenever the server is correct. This generalizes the impossibility result of Cachin et al. [2], and requires a more elaborate proof.

*IBM Research, Zurich Research Laboratory, CH-8803 Rüschlikon, Switzerland. cca@zurich.ibm.com

†Department of Electrical Engineering, Technion, Haifa 32000, Israel. {idish@ee, shralex@tx}.technion.ac.il