CCIT REPORT #731 May 2009

Dynamic Atomic Storage Without Consensus

Marcos K. Aguilera* Idit Keidar† Dahlia Malkhi* Alexander Shraer†

June 2, 2009

Abstract

This paper deals with the emulation of atomic read/write (R/W) storage in *dynamic* asynchronous message passing systems. In static settings, it is well known that atomic R/W storage can be implemented in a fault-tolerant manner even if the system is completely asynchronous, whereas consensus is not solvable. In contrast, all existing emulations of atomic storage in dynamic systems rely on consensus or stronger primitives, leading to a popular belief that dynamic R/W storage is unattainable without consensus.

In this paper, we specify the problem of dynamic atomic read/write storage in terms of the interface available to the users of such storage. We discover that, perhaps surprisingly, dynamic R/W storage is solvable in a completely asynchronous system: we present DynaStore, an algorithm that solves this problem. Our result implies that atomic R/W storage is in fact easier than consensus, even in dynamic systems.

1 Introduction

Distributed systems provide high availability by replicating the service state at multiple processes. A fault-tolerant distributed system may be designed to tolerate failures of a minority of its processes. However, this approach is inadequate for long-lived systems, because over a long period, the chances of losing more than a minority inevitably increase. Moreover, system administrators may wish to deploy new machines due to increased workloads, and replace old, slow machines with new, faster ones. Thus, real-world distributed systems need to be *dynamic*, i.e., adjust their membership over time. Such dynamism is realized by providing users with an interface to *reconfiguration* operations that *add* or *remove* processes.

Dynamism requires some care. First, if one allows arbitrary reconfiguration, one may lose liveness. For example, say that we build a fault tolerant solution using three processes, p_1, p_2, and p_3. Normally, the adversary may crash one process at any moment in time, and the up-to-date system state is stored at a majority of the current configuration. However, if a user initiates the removal of p_1 while p_1 and p_2 are the ones holding the up-to-date system state, then the adversary may not be allowed to crash p_2, for otherwise the remaining set cannot reconstruct the up-to-date state. Providing a general characterization of allowable failures under which liveness can be ensured is a challenging problem.

A second challenge dynamism poses is ensuring safety in the face of concurrent reconfigurations, i.e., when some user invokes a new reconfiguration request while another request (potentially initiated by another user) is under way. Early work on replication with dynamic membership could violate safety in such cases [7, 22, 9] (as shown in [27]). Many later works have rectified this problem by using a centralized sequencer or some variant of consensus to agree on the order of reconfigurations (see discussion of related work in Section 2).

*Microsoft Research Silicon Valley, {aguilera, dalia}@microsoft.com

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