Abstract

We present here a synopsis of a keynote presentation given by Idit Keidar at OPODIS 2015, the International Conference on Principles of Distributed Systems, which took place in Rennes, France, on December 14-17 2015. More details may be found in [9].

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases distributed storage, impossibility

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2015.4

Category Keynote

1 Introduction

In recent years we see an exponential increase in storage capacity demands, creating a need for big data storage solutions. Additionally, today’s economy emphasizes consolidation, giving rise to massive data centers and clouds. In this era, distributed storage plays a key role. Data is typically stored on a collection of storage nodes and is accessed by clients over the network. Due to the geographical spread of such systems, communication is usually modeled as asynchronous.

Given the inherent failure-prone nature of storage and network components, a reliable distributed storage algorithm must store redundant information in order to allow data to remain available when storage nodes fail or go offline. The most common approach to achieve this is via replication [2], i.e., storing copies of each data block on multiple nodes. In asynchronous settings, \(2f+1\) replicas are needed in order to tolerate \(f\) failures [2]. Given the immense size of data, the storage cost of replication is significant. Some previous works have attempted to mitigate this cost via the use of erasure codes [1, 3, 6, 4, 10, 5].

Indeed, code-based solutions can reduce the storage cost as long as data is not accessed concurrently by multiple clients. For example, if the data size is \(D\) bits and a single failure needs to be tolerated, erasure-coded storage ideally requires \((k+2)D/k\) bits for some parameter \(k > 1\) instead of the \(3D\) bits needed for replication. But as concurrency grows, the cost of erasure-coded storage grows with it: when \(c\) clients access the storage concurrently, existing code-based algorithms store \(O(cD)\) bits. Intuitively, this occurs because coded data...
cannot be reconstructed from a single storage node. Therefore, writing coded data requires coordination – old data cannot be deleted before ensuring that sufficiently many blocks of the new data are in place. This is in contrast with replication, where data can always be read coherently from a single copy, and so old data may be safely overwritten without coordination.

In this work we prove that this extra cost is inherent, by showing a bound on the storage complexity of asynchronous reliable distributed storage algorithms. Our bound takes into account three problem parameters: $f, c$, and $D$, where $f$ is the number of storage node failures tolerated (client failures are unrestricted), $c$ is the concurrency allowed by the algorithm, and $D$ is the data size. For these parameters, we prove that the storage complexity is $\Theta(D \cdot \min(f, c))$. Asymptotically, this means either a storage cost as high as that of replication, or as high as keeping as many versions of the data as the concurrency level.

## 2 Lower bound

Our formal results are proven for emulations of a lock-free multi-reader multi-writer regular register [7, 8].

For our lower bound, we consider algorithms that use (arbitrary) black-box encoding schemes, which produce coded blocks of a given stored value independently of other values. We assume that the storage consists of such coded blocks, in addition to possibly unbounded data-independent meta-data, which we neglect. Given our storage model, every data bit in the storage can be associated with a unique written value. Therefore, we measure the storage cost of every value as the total number of bits in the storage that are associated with this value; the total storage cost as the sum of the costs of all values.

We define a parameter $0 \leq \ell \leq D$, and observe that if a value $v$ is associated with fewer than $D - \ell$ bits in the storage, then more than $\ell$ bits (associated with $v$) still needed to be written to the storage before $v$ can be read.

Note that we use here a fundamental information-theoretic “pigeonhole” argument that any representation, either coded or un-coded, cannot guarantee to recover a value $v \in \mathcal{V}$ precisely from fewer than $D = \log_2 |\mathcal{V}|$ bits. This argument excludes common storage-reduction techniques like compression and de-duplication, which only work in probabilistic setups and with assumptions on the written data.

Given the above observation, we define a particular adversary behavior and prove that it drives the storage to a state where either (1) $f + 1$ storage nodes hold at least $\ell + 1$ bits each, or (2) the storage holds at least $D - \ell$ bits of $c$ different values. Now, picking $\ell = D/2$ implies our lower bound on storage cost:

▶ **Theorem 1.** The storage cost of any algorithm that simulates a regular lock-free register with up to $f$ storage node failures, $c$ concurrent writes, and a value domain of $2^D$ bits is $\Omega(D \cdot \min(f, c))$ bits.

## 3 Algorithm

Finally, we present an adaptive reliable storage algorithm whose storage cost is $O(D \cdot \min(f, c))$. We achieve this by combining the advantages of replication and erasure coding. Our algorithm does not assume any a priori bound on concurrency; rather, it uses erasure codes when concurrency is low and switches to replication when it is high.
References


