Brief Announcement: Efficient Replication of Large Data Objects

Rui Fan
MIT Laboratory for Computer Science
Cambridge, MA
rfan@theory.lcs.mit.edu

Nancy A. Lynch
MIT Laboratory for Computer Science
Cambridge, MA
lynch@theory.lcs.mit.edu

Data replication is an important technique for improving the reliability and scalability of data services. To be most useful, replication should be transparent: operations should appear to execute atomically on one logical copy of the data, even when multiple physical copies exist. Current atomic replication algorithms generally fall into two categories. The first uses locking protocols or group communication, and the second replicates the data in quorum systems. Both techniques suffer from poor performance; the former from the overhead of locking or GC algorithms, and the latter because each operation involves accessing many replicas. Their performance penalty is significant enough that they are rarely used in practical systems, which opt for non-atomic but faster algorithms. In this brief announcement, we discuss *Layered Data Replication (LDR)*, an efficient and practical replication algorithm guaranteeing atomicity, using only asynchronous message-passing channels. *LDR* assumes the size of the data it replicates is large compared to metadata (e.g. tags) which it uses. This allows the algorithm to perform more cheap operations on the metadata, and avoid expensive operations on the real data. The large-data assumption is reasonable in many settings, e.g. a replicated file system. *LDR* accesses only one copy of the data per read operation, and accesses $f+1$ copies per write, where f is the number of replica failures *LDR* tolerates. Since any replication algorithm must read one copy of the data per read, and write $f+1$ copies of the data in order to survive f faults, *LDR*'s performance is optimal. Additionally, we prove two lower bounds on the costs of replication. These lower bounds show some of the constructions used in *LDR* are necessary, and they are also of independent interest.

The basic idea in *LDR* is to separately store copies of the data in *replica servers*, and store information about where the most up-to-date copies are located in *directory servers*. During a read, directories are first read to locate the most up-to-date replicas, then the data is read from one such replica. In a write, the data is first written to $f+1$ replicas, then the directories are informed that these replicas are most up-to-date. This description omits some important details, which we discuss below in the context of our lower bounds. *LDR* adopts the algorithm of [1] when accessing the directories, and writes back the latest information it reads to the directories. This behavior is not an artifact of *LDR*: our first lower bound shows that any atomic replication algorithm must sometimes perform f (physical) writes during a (logical) read operation, where f is the fault-tolerance of the algorithm. The intuition behind this result is that during the course of a write operation, the system is sometimes in an ambiguous state, when a read operation can return either an old value or the new value being written. A reader needs to write in order to record which value it decided to return, so that later reads can make a consistent decision. Since any processor the reader writes to may fail, the reader must write to at least f processors.

When there are concurrent writes in *LDR*, a replica sometimes stores several values of the data. Our second lower bound shows this behavior is inherent for the class of *selfish* atomic replication algorithms. Intuitively, a selfish replication algorithm is one in which operations can only "help" each other in "cheap" ways. We formalize these notions in the full paper [2]. Recall that helping is a crucial technique for implementing many lock-free data-structures, but carries with it a performance penalty. A selfish algorithm, then, allows lock-freedom and high performance. But, we prove that a selfish replication algorithm must use memory which is proportional to the maximum number of concurrent writers the algorithm tolerates. This is done by assuming the contrary, and constructing an execution involving multiple writers, one of which completes its write but leaves its data in an "unsafe" state. We use this fact to force consecutive reads following this execution to return different values. But eventually some two nonconsecutive reads must return the same value, violating the atomicity of the replication algorithm, and proving the lower bound. Interestingly, the "morals" of our lower bounds appear in the context of many other lock-free algorithms.

PODC'03, July 13–16, 2003, Boston, Massachusetts, USA.

1. REFERENCES

[1] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM*, 42(1):124–142, January 1995.

[2] R. Fan. Efficient replication of large data-objects. Technical Report MIT-LCS-TR-886, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, February 2003.