An Analysis of Replica Control

Shu-Wie F Chen Calton Pu
Department of Computer Science
Columbia University
New York, NY 10027

Abstract

Early replica control work focused on maintaining the availability of an object despite the unavailability of its copies. More recent work has focused on improving the response time, autonomy, and scalability of the earlier work while preserving their availability characteristics. In this paper, we analyze the several ways the new algorithms make such improvements.

1 Introduction

Replica control (RC) is responsible for mapping the values of objects to the values of their copies for write operations. Conversely, RC maps values of copies to values of objects for read operations. Early RC work focused on maintaining the availability of an object despite the unavailability of its copies. Examples include primary copy [2, 22], quorum consensus [12], and available copies [3]. To the best of our knowledge, Ceri et al's classification [8] is the only systematic attempt to analyze and compare the structure of RC algorithms. Specifically, they have analyzed RC algorithms according to the number of copies read and written for each logical operation in conjunction with the kind of agreement protocol used to ensure operation atomicity. The classification illustrates that the focus of RC work has been on reducing failure unavailability while maintaining mutual consistency.

A recent trend we have observed in RC work has been their focus on conflict unavailability. It may be that the study of failure unavailability has matured, as shown by Ceri et al's classification. Also, it may be that system components have become more resistant to failures, as shown by Gray's empirical study [13].

2 Availability in Replica Control

We say that an object is available if the required number of copies (as specified by the RC protocol) are accessible to the operations of a program. A copy may be inaccessible because its storage site is down or has been partitioned from the site of the requesting program. We call the unavailability caused by site and communication failures failure unavailability.

A copy can be unavailable even if it is physically accessible to a program. For instance, a program that locks a logical file for writing by obtaining locks on all copies of the file prevents other programs from simultaneously accessing any copy. In other words, though a program may be able to physically access a copy, it may not be allowed to do so because of interactions with other programs. We call the unavailability caused by conflicting programs conflict unavailability.

The goal of traditional RC algorithms has been to reduce failure unavailability. The fundamental correctness criterion for these algorithms has been mutual consistency. The general structure for RC algorithms that guarantee mutual consistency has consisted of (1) a protocol that defined which copies must be physically read or written for each logical operation and (2) a mechanism that implemented the protocol. To guarantee mutual consistency, the protocols had to satisfy the requirement of quorum intersection [14] which ensured that at least one of the copies accessed by a read operation was a copy that had been most recently written with the latest value. Similarly, the mechanisms were required to be synchronous in the sense that they atomically updated a protocol-defined number of copies.

A recent trend we have observed in RC work has been their focus on conflict unavailability. It may be that the study of failure unavailability has matured, as shown by Ceri et al's classification. Also, it may be that system components have become more resistant to failures, as shown by Gray's empirical study [13].
In both cases, the primary unavailability problem in the system is shifting away from failures to conflicts.

We divide the techniques to reduce conflict availability into two groups: (1) those refining the implementation of RC algorithms, and (2) those relaxing the requirement of mutual consistency. These techniques are independent and described in the following sections. Although they can be combined in principle, existing papers seem to focus on one or the other.

3 Refining RC Implementations

In distributed database systems, the additional correctness criterion of serializability (SR) is required and is enforced by a concurrency control (CC) algorithm. Like RC, a CC algorithm can be thought to be composed of protocol and mechanism components. A system that guarantees both mutual consistency and serializability is said to provide one-copy serializability (1SR) [4].

In ISR systems, RC and CC can be thought as logically separate entities, but in implementations, they are often intertwined. We have found it useful to view such a replicated system as a layered architecture comprised of, from top to bottom, an RC protocol, an RC mechanism, a CC mechanism, and a communication mechanism. In addition, the mapping from logical operations to physical operations suggests a CC protocol as the top-most layer.

Combining the mechanism layers is the main approach to the refinement of many RC implementations. The basic idea is to make use of synchronization mechanisms in one layer to achieve synchronization in other layers. For instance, ISIS uses the causal multicast CBCAST with CC synchronization and the atomic multicast ABCAST to guarantee proper ordering of RC operations [5, 6]. The piggybacked and concurrent update techniques piggyback RC operation requests on CC requests [15]. Non-blocking quorum consensus (NBQC) uses Gifford's weighted voting RC protocol, but depends on the CC mechanism and gossip messages for correctly ordered delivery of RC operations [1]. Lazy replication also relies on gossip messages [17].

4 Relaxing Consistency Requirements

Another approach to reducing conflict unavailability has been to relax the correctness criterion of mutual consistency and provide a weaker form of consistency. These weaker criteria can be specified along the two dimensions of time and space [21]. These weaker criteria have been achieved by (1) relaxing the RC protocol requirement so that read and write operations do not necessarily overlap on copies and (2) relaxing the RC mechanism requirement so that update operations are no longer synchronous.

One example of recent RC work that has relaxed the protocol requirement is N-ignorance [16] which uses a variant of consensus locking [14] but does not require that the quorum sets of programs intersect. Therefore, it guarantees neither mutual consistency nor serializability. We have also found that many practical replicated systems, especially distributed file systems, use a read-one, write-one (ROWO) RC protocol that does not guarantee mutual consistency [9].

There has also been much RC work that has relaxed the RC mechanism requirement. Grapevine, Clearinghouse, and Global Name Service (GNS) use asynchronous propagation mechanisms. Each system is an improvement on the previous one with respect to the reliability and timeliness of delivery, but the updates are still performed asynchronously. Siphon introduces some CC locking to provide a closer approximation to mutual consistency. Since synchronous mechanisms are no longer desired, much of the responsibility for update propagation has been pushed from the RC mechanism layer to the communication mechanism layer. Epidemic propagation schemes such as anti-entropy and direct mail [10] are used in systems such as OSCAR [11] to propagate updates.

As described above, RC algorithms provide a spectrum of object consistency depending on the characteristics of the RC protocol and the underlying mechanisms that implement the protocol. There is a trade-off between the desired consistency and the performance and availability characteristics of the implementation. The further the divergence from mutual consistency, the greater the potential performance and availability. There has been recent work on exploiting this trade-off. In particular, the notion of epsilon-serializability (ESR) [18] has been proposed to address limitations of SR. ESR provides users with a high-level and simple mechanism for specifying inconsistency, thereby allowing users control over the trade-off between performance and availability and a bounded divergence from mutual consistency and serializability. ESR algorithms have been proposed for asynchronous replica control to improve performance and availability [19] and have also been shown to be applicable in improving autonomy [20].
We have identified conflict unavailability (due to the scheduling of conflicting read/write operations) as the focus of recent replica control (RC) work. This is in contrast to failure unavailability (due to the node and network failures) as the focus of traditional RC research. The shift may be motivated by the growing reliability of system components, or by the maturation of failure unavailability field, or both.

The recent RC algorithms on reducing conflict unavailability are divided into two groups: (1) those refining the implementation of RC algorithms, and (2) those relaxing the requirement of mutual consistency. We introduce a five-layer architecture to explain the different ways an implementation of RC algorithm can be refined by combination of adjacent layers. From top to bottom, the architecture consists of concurrency control protocol, RC protocol, RC implementation, concurrency control implementation, and communication mechanism. Typical refinements combine some or all of the three bottom layers. Relaxing consistency requirements is another dimension in reducing conflict unavailability. In many applications, mutual consistency or ISR is not required. Several techniques to relax the consistency requirements in a controlled way are outlined and compared.

Figure 1 summarizes the refinements and techniques presented above. It is not a comprehensive summary of all RC work and is meant merely to serve as an illustration of our classifications. References to the systems can be found in [9]. It is easy to see that if a system cannot guarantee mutual consistency, then it cannot guarantee serializability. Further, if the replicated system does not use an RC protocol that supports mutual consistency, then there is no need to implement a synchronous RC mechanism since there is no guarantee that mutual consistency can be achieved.

A related work [9] shows that much of practical replication work (especially in distributed file systems) read and write one copy. Thus the RC algorithms that read and write one copy are more relevant than voting algorithms. This position paper makes a similar observation, that recent RC work has focused on conflict unavailability. We also point out that the two approaches to reducing conflict unavailability, the refinement of RC implementation and the relaxation of consistency requirements, can be combined to further improve system performance, availability, scalability, and autonomy.

References


