Chapter 5
Replicating for Performance: Case Studies

Maarten van Steen and Guillaume Pierre

Abstract In this chapter we take a look at the application of replication techniques for building scalable distributed systems. Unlike using replication for attaining dependability, replicating for scalability is generally characterized by higher replication degrees, and thus also weaker consistency. We discuss a number of cases illustrating that differentiation of replication strategies, for different levels of data granularity, is needed. This observation leads us to conclude that automated replication management is a key issue for future research if replication for scalability is to be successfully deployed.

5.1 Introduction

Building scalable distributed systems continues to be one of the more challenging tasks in systems design. There are three independent and equally important perspectives on scalability [20]:

- **Size scalability** is formulated in terms of the growth of number of users or data, such that there is no noticeable loss in performance or increase in administrative complexity.
- A system is said to be geographically scalable when components can be placed far apart without seriously affecting the perceived performance. This perspective on scalability is becoming increasingly important in the face of distributing a service across the Internet.
- **Administrative scalability** describes the extent to which a system can be put under the control of multiple administrative organizations without suffering from performance degradation or increase of complexity.

In this chapter, we will concentrate on size and geographical scalability, in particular in relation to the perceived performance of a system. More specifically, we are interested in scalability problems that manifest themselves through performance degradation. To keep matters simple, in the following we will refer to scalability in only this more narrow context.
To address scalability problems, there are essentially only two techniques that we can apply. Following the terminology as proposed in [5], we can **partition** the set of processes and the collection of data those processes operate on, and spread those parts over different nodes of the distributed system. An excellent example of where this scaling technique has been successfully applied is the Web, which can be viewed as a huge, distributed information system. Each Web site is responsible for handling its own part of the entire data set, allowing hundreds of millions of users to access the system simultaneously. As we will discuss later, numerous sites need further partitioning as a single machine can not handle the stream of requests directed to them.

Another illustrative example of where partitioning has been successfully applied is in the Internet’s Domain Name System. By October 2008, the entire name space had been partitioned across an estimated 11.9 million servers\(^1\). These servers collaborate in resolving names, and in such a way that many requests can be handled simultaneously. However, an important reason why DNS generally performs so well, is also because much of its data has been **cloned**, or more formally, **replicated**.

Cloning processes and associated data is useful for addressing geographical scalability problems. The principle is simple: by placing services close to where they are needed, we can reduce performance degradation caused by network latencies, and at the same time by placing a service everywhere it is needed, we address size scalability by dividing the load across multiple servers. In the following, we shall often use the term replication instead of cloning.

A main issue with replication is that it requires each update to be carried out at each replica. As a consequence, it may take a while before all replicas are the same again, especially when updates need to be carried out at many replicas spread across a large network such as the Internet. More problematic is when multiple updates need to be carried out concurrently, as this requires global synchronization if we wish to guarantee that in the end the replicas are indeed the same. Global synchronization requires the execution of an agreement protocol. Such an execution is generally not scalable: too many parties may need to communicate and wait for results before an update can be finally committed. An important consequence is that if we apply replication as a scaling technique, then we generally need to compromise on consistency: copies cannot be kept the same at all time.

This observation is not new. For example, it is well known among architects of very large Web-based systems such as Amazon, Google, and eBay that scalability can be attained only by “embracing inconsistency”\(^2\). A keyword here is **eventual consistency**: in the absence of further updates, replicas will converge to the same state (see also [34]). Accepting eventual consistency as the best possible option is needed when dealing with cloned services. The problem is that there is no way that one can guarantee the combination of strong consistency, availability, and coping with partitionable networks at the same time. This so-called CAP conjecture was postulated by Eric Brewer in 2000 and proved correct two years later [9].

---

1. [http://dns.measurement-factory.com/surveys/200810.html](http://dns.measurement-factory.com/surveys/200810.html)
2. eBay’s Randy Shoup at his presentation at Qcon, London, 2008.