The life and times of a ZooKeeper (Talk abstract)

Flavio P. Junqueira
Yahoo! Research
Barcelona, Spain
fpj@yahoo-inc.com

Benjamin C. Reed
Yahoo! Research
Santa Clara, CA
breed@yahoo-inc.com

1. ABSTRACT

A distributed application may comprise a range of processors, from two to thousands. For the processors of an application comprising many different machines to work together, they must often coordinate. Coordination may be as simple as agreeing upon a basic configuration, *e.g.*, a list of server addresses and ports for example. In the context of distributed systems, even agreeing upon a basic configuration can be complicated: servers may be down during configuration changes, servers may need to adapt quickly to dynamic configuration changes, and during changes servers may have conflicting configurations. Most distributed applications also require more sophisticated coordination primitives such as leader election, group membership, and rendezvous.

At Yahoo!, we noticed that many distributed applications were re-implementing coordination primitives in their distributed applications. In theory, this is completely reasonable: coordination protocols are well known, and usually the coordination logic is intermingled with the application logic. In practice, the story is much different; these protocols have sometimes subtle requirements that can easily be overlooked when implementing them. Further, an application developer is usually much more interested in working on application logic than the coordination protocol that the logic depends on. We found many cases of applications whose coordination primitives were buggy, a single point of failure, poorly performing, or oversimplified; in some cases the applications suffered from all of the above.

We started ZooKeeper as a system that could address all these problems in a general way so that all of our applications could use it for coordination and application developers could focus on developing their applications. By providing a general system that all applications could use, we could devote the time to making it robust, fault-tolerant, and with good enough performance to be used extensively by applications. We also needed to balance two possibly conflicting goals: ZooKeeper needed to be general enough to address our coordination needs and simple enough to implement a correct high performance service. We found that we were able to achieve our goals by trading strong synchronization for strong ordering guarantees and a wait-free interface.

Inside and outside of Yahoo! developers of distributed applications have enthusiastically embraced ZooKeeper. Developers who have previously dealt with the difficulties of implementing distributed applications quickly see the benefits of ZooKeeper, and we have many applications inside of Yahoo! that make use of it. It was developed quickly by a small group of developers in a small amount of time.

The general techniques we use in ZooKeeper are not fundamentally new. For instance, ZooKeeper uses a leader-based atomic broadcast protocol that may seem at a first glance not novel. However, a deeper inspection reveals that it has some key properties that are crucial to guarantee the properties that developers of large-scale applications require. To the best of our knowledge, no previous algorithm presented all necessary properties "out-of-the-box".

In this talk, we share previous work that influenced ZooKeeper. In particular, we compare at the protocol level with Paxos [2] and Viewstamped Replication [4], and at the service level with services like Chubby [1] and Boxwood [3]. We give a high level overview of the service itself and describe its implementation. Along the way we point out some of the design and implementation details that were key in achieving the correctness and performance that we need. Finally, we show how ZooKeeper performance has evolved over time. We have increased the performance of ZooKeeper an order of magnitude from our first implementation, and none of that increase has come from protocol changes. Indeed, our protocol has remained constant from the very beginning.

Categories and Subject Descriptors: C.2.4 [Computer-Communication Network]: Distributed Systems – distributed applications, network operating systems ; D.4.5 [Operating Systems]: Reliability – fault-tolerance

General terms: Algorithms, Design, Reliability.

Keywords: distributed systems, fault tolerance, replication, distributed coordination, concurrency

2. REFERENCES

[1] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th ACM/USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 335–350, 2006.

[2] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[3] J. MacCormick, N. Murphy, M. Najork, C. A. Thekkath, and L. Zhou. Boxwood: Abstractions as the foundation for storage infrastructure. In *Proceedings of the 6th ACM/USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 105–120, 2004.

[4] B. M. Oki and B. H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *PODC '88: Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, pages 8–17, New York, NY, USA, 1988. ACM.