Mitigating Contention in Distributed Systems

General Exam

Brandon Holt
University of Washington
bholt@cs.uw.edu

Abstract

The online world is a dangerous place for interactive applications. A single post by Justin Bieber can slow Instagram to a crawl; a black and blue dress going viral sends Buzzfeed scrambling to stay afloat; Star Wars movie ticket pre-sales cause service interruptions for Fandango and crash others. Developers of distributed applications must work hard to handle these high-contention situations because though they are not the average case, they affect a large fraction of users.

Techniques for mitigating contention abound, but many require programmers to make tradeoffs between performance and consistency. In order to meet performance requirements such as high availability or latency targets, applications typically use weaker consistency models, shifting the burden of reasoning about replication to programmers. Developers must balance mechanisms that reign in consistency against their effect on performance and make difficult decisions about where precision is most critical.

Abstract data types (ADTs) hide implementation details behind a clean abstract representation of state and behavior. This high-level representation is easy for programmers to build applications with and provides distributed systems with knowledge of application semantics, such as commutativity, which are necessary to apply contention-mitigating optimizations. With *inconsistent, probabilistic, and approximate (IPA) types*, we can express weaker semantics than with traditional ADTs, enabling techniques for weak consistency to be used and allowing precision to be explicitly traded for performance where applications can tolerate error. In previous work, we have demonstrated that ADT awareness can result in significant performance improvements: 3-50x speedup in transaction throughput for high-contention workloads such as an eBay-like auction service and a Twitter-like social network. IPA types are proposed for future work.

1. Introduction

Imagine a young ticket selling app, embarking on a mission to help people everywhere get a seat to watch their favorite shows. Bright-eyed and ready to face the world, it stores everything in a key/value store so that it will not forget a thing. It uses a distributed key/value store, so when it expands into new cities and reaches its millionth customer, the datastore continues to grow with it. The app uses strong consistency and transactions to ensure that no tickets are sold twice or lost, and its customers praise its reliability. "Five stars. That little app always gets my tickets, fast!" they say.

Then one day, pre-sales for the 7th Star Wars movie come out and suddenly it is inundated under a surge over 7 times the usual load, as a record-breaking number of people try to purchase tickets for this one movie. This concentrated traffic causes *hot spots* in the datastore; while most nodes are handling typical traffic, the traffic spike ends up being funneled to a handful of nodes which are responsible for this movie. These hotspots overload this app's "scalable" datastore, causing latencies to spike, users' connections to time out, and disappointed users