

Claret: Avoiding Contention in Distributed Transactions with Abstract Data Types

Paper #120

Abstract

Interactive distributed applications like Twitter or eBay are difficult to scale because of the high degree of writes or update operations. The highly skewed access patterns exhibited by real-world systems lead to high contention in datastores, causing periods of diminished service or even catastrophic failure. There is often sufficient concurrency in these applications to scale them without resorting to weaker consistency models, but traditional concurrency control mechanisms operating on low level operations cannot detect it.

We describe the design and implementation of Claret, a Redis-like data structure store which allows high-level application semantics to be communicated through *abstract data types* (ADTs). Using this abstraction, Claret is able to avoid unnecessary conflicts and reduce communication, while programmers continue to implement applications easily using whatever data structures are natural for their use case. Claret is the first datastore to use ADTs to improve performance of distributed transactions; optimizations include transaction boosting, phasing, and operation combining. On a transaction microbenchmark, Claret’s ADT optimizations increase throughput by up to 49x over the baseline concurrency control and even up to 20% better than without transactions. Furthermore, Claret improves peak throughput on benchmarks modeling real-world high-contention scenarios: 4.3x speedup on the Rubis auction benchmark, and 3.6x on a Twitter clone, achieving 67-82% of the non-transactional performance on the same workloads.

1. Introduction

Today’s online ecosystem is a dangerous place for interactive applications. Memes propagate virally through social networks, blogs, and news sites, bringing overwhelming forces

to bear on fledgeling applications that put DDOS attackers to shame. In February 2015, a picture of a black and blue dress exploded across the internet as everyone debated whether or not it was actually white and gold, which brought unprecedented traffic spikes to BuzzFeed [31], the site responsible for sparking the viral spread. Even in its 8th year of dealing with unpredictable traffic, Twitter briefly fell victim in 2014 after Ellen Degeneres posted a selfie at the Oscars which was retweeted at a record rate [6].

These high traffic events arise due to a number of factors in real world systems such as power law distributions and live events. The increasing interactivity of modern web applications results in significant contention due to writes in datastores. Even content consumption can result in writes as providers track user behavior in order to personalize their experience, target ads, or collect statistics [8].

To avoid catastrophic failures and mitigate poor tail behavior, significant engineering effort must go into handling these challenging high-contention scenarios. The reason writes are such a problem is that they impose ordering constraints requiring synchronization in order to have any form of consistency. Luckily, many of these orderings are actually irrelevant from the perspective of the application: some actions are inherently acceptable to reorder. For example, it is not necessary to keep track of the order in which people retweeted Ellen’s selfie.

One way to avoid constraints is to use eventual consistency, but then applications must deal with inconsistent data, especially in cases with high contention. Instead, if systems could directly use these application-level constraints to expose concurrency and avoid over-synchronizing, they could eliminate many false conflicts and potentially avoid falling over during writing spikes, without sacrificing correctness. Databases and distributed systems have long used properties such as commutativity to reduce coordination and synchronization. The challenge is always in communicating these application-level properties to the system.

In this work, we propose a new way to express high-level application semantics for transactions through *abstract data types* (ADTs) and consequently avoid unnecessary synchronization in distributed transactional datastores. ADTs allow users and systems alike to reason about their logical behav-

