BRB: BetteR Batch Scheduling to Reduce Tail Latencies in Cloud Data Stores

Waleed Reda
Université catholique de Louvain

Lalith Suresh*
TU Berlin

Marco Canini
Université catholique de Louvain

Sean Braithwaite
SoundCloud

ABSTRACT

A common pattern in the architectures of modern interactive web-services is that of large request *fan-outs*, where even a single end-user request (*task*) arriving at an application server triggers tens to thousands of data accesses (*sub-tasks*) to different stateful backend servers. The overall response time of each task is bottlenecked by the completion time of the slowest sub-task, making such workloads highly sensitive to the tail of latency distribution of the backend tier. The large number of decentralized application servers and skewed workload patterns exacerbate the challenge in addressing this problem.

We address these challenges through BetteR Batch (BRB). By carefully scheduling requests in a decentralized and task-aware manner, BRB enables low-latency distributed storage systems to deliver predictable performance in the presence of large request fan-outs. Our preliminary simulation results based on production workloads show that our proposed design is at the 99^{th} percentile latency within 38% of an ideal system model while offering latency improvements over the state-of-the-art by a factor of 2.

CCS Concepts

•General and reference → Performance; •Information systems → *Distributed storage;*

1. INTRODUCTION

Recent studies [3] showed that latency distributions in Web-scale systems exhibit long-tail behaviors where the 99^{th} percentile latency can be more than one order of magnitude higher than the median latency. To make matters worse, modern applications are highly distributed. For instance, interactive web services involve parallelization and aggregation of responses across 10s-1000s of servers, all of which need to finish for an end-user request (*e.g.*, a search query) to be considered complete.

There have been several proposals for achieving latency reduction and lowering the impact of skewed performance across backend servers. These include (*i*) duplicating or reissuing requests, predicting stragglers, or trading off completeness for latency [3], (*ii*) policy-based resource allocation and admission control (with the objective of achieving fairness or satisfying SLOs) [4], and recently (*iii*) making use of adaptive load-balancing [5]. In this work, we present BetteR Batch (BRB), which complements the above mentioned approaches using task-aware scheduling — a method of scheduling tasks across stateful backend replica servers according to expected service time of the enclosed sub-tasks.

2. BETTER BATCH OVERVIEW

In BRB, we focus on improving the performance of replicated and partitioned data stores. The system we aim to optimize consists of a set S of flexible servers and a set C of clients. Servers are flexible in that every server belongs to R replica groups and can service requests for any of the replica groups it is part of. A replica group is a collection of servers each of which contains a replica of a data partition. For simplicity, we also take R as the replication factor in the system and consider read operations where 1 out R servers is used. With the term client we mean an application server that receives user requests (*e.g.*, a request for a web page). Based on this system model, our end-goal is to provide the lowest latencies for accesses to the data store.

Our approach is based on two key observations. First, replicated data stores provide the opportunity to lower latencies via intelligent replica selection: that is, selecting one out of multiple replica servers to serve a request in a load-aware fashion, similarly to our prior work in [5]. Second, in many real-world workloads, applications have large fan-outs: that is, they access several elements from the data store as a single batched request (*e.g.*, requesting all tracks in a playlist). We call the set of logically-related operations on the data store a task. Typically a task is complete only once all of its operations complete.

The nature of these workloads presents opportunities to minimize latency: in the spatial dimension, one can jointly optimize replica selection across all operations in a task; in the temporal dimension, one can optimize the schedule of operations by considering what bottleneck would affect the task completion time. Task-aware approaches have recently been applied for reducing the flow completion time in network transfers [2] but, to the best of our knowledge, no prior

*Work partly done when the author was visiting at Université catholique de Louvain and while interning at SoundCloud.

SIGCOMM '15 August 17–21, London, United Kingdom

ACM ISBN 978-1-4503-3542-3/15/08.

DOI: http://dx.doi.org/10.1145/2785956.2790023

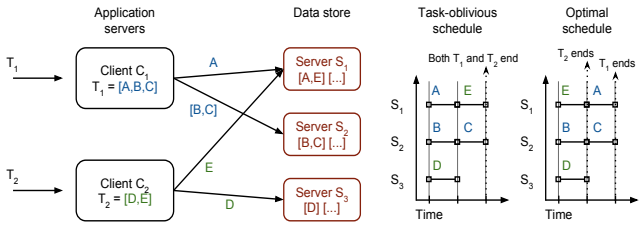

Figure 1: Intelligent replica selection and scheduling are effective techniques for reducing latencies.

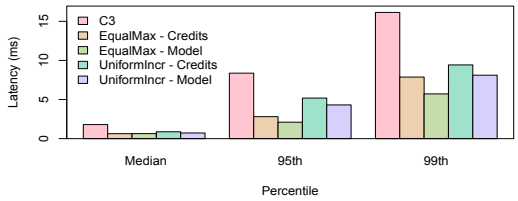

Figure 2: Task latency comparisons between BRB strategies and C3 [5].

work has adopted them for replicated and partitioned data stores (such as Cassandra or Riak).

Figure 1 illustrates these optimization opportunities. Assume S_1, S_2 and S_3 are the available replicas for processing tasks T_1 and T_2. Because task T_1 is complete only once all of its operations complete, it implies that there is some slack for accessing A — in particular, to cause no extra delay to T_1, the access to A can be flexibly delayed for as long as it completes within 2 time units. Crucially, if information about the deadline of a task is available, a server can choose to serve operations to meet their deadlines while minimizing the delay of other operations. In our example, S_1 can optimize the data accesses so that E is serviced before A — doing otherwise results in a suboptimal schedule where T_2 completes in 2 time units whereas in the optimal schedule the completion time of T_2 is just 1 time unit. In this simple example, it is evident that a system that makes decisions greedily for each request in a task-oblivious fashion could easily perform sub-optimally as opposed to using a task-aware approach, which is our goal in this work.

2.1 Task-Aware Scheduling Algorithms

A key insight to reduce tail latencies is that a task's response time depends on the last of its requests to complete. BRB is based on a class of algorithms for task-aware scheduling that exploit this fact. At a high-level, these algorithms run at the clients of the data store as follows. When receiving a task, clients subdivide it into a set of sub-tasks, one for each replica group; therefore, a sub-task contains all requests for a distinct replica group. Clients then determine the bottleneck sub-task based on the costliest sub-task and assign a priority to every request in the task. Then, the priority information is propagated to the servers, which based on this can decide what request to serve next. To this end, we have designed two simple yet effective priority assignment algorithms:

EqualMax: Requests are given the same priority as that of the bottleneck sub-task. The intuition is that tasks with shorter bottlenecks should be given precedence in order to minimize their makespan. This is equivalent to Shortest Job First (SJF) scheduling, however, in our case, given that task completion times are determined by the last request to finish, the bottleneck is used instead of the individual service time of requests.

UnifIncr: Requests are ranked based on the difference between the cost of the bottleneck sub-task and their individual cost. In other words, this effectively prioritizes requests according to how long they are allowed to slack behind the bottleneck. The main idea is that requests that have longer forecasted service times (based on the size of the value they are requesting) should be given a higher priority, given that they are more likely to bottleneck their respective tasks.

2.2 Preliminary Evaluation

We resort to simulation to evaluate the potential benefits of our proposal. We assess BRB's latency reductions versus ideal as well as state-of-the-art approaches — in our case, C3 [5]. In an ideal case, referred to as *model*, servers utilize a work-pulling mechanism to fetch requests from a single global priority-based queue shared by all clients. However, such a model is unrealizable since it assumes perfect knowledge of global state. Hence, we develop a *credits* strategy where clients report their demands at measurement intervals and are assigned credits (*i.e.*, shares of server capacity) proportionally to demands via a logically-centralized controller; once demand exceeds server capacity, a congestion signal is sent to the controller and the credits allocations are adapted accordingly at 1s intervals. In such a realization, each server maintains a separate priority-queue.

For our evaluation, we simulate a system with 18 clients and 9 servers at a concurrency level of 4 cores, each operating at an average service rate of 3500 requests/s. We set our one-way network latency to 50 μs. The workload is gathered from SoundCloud and comprises of approximately 500,000 tasks, with an average fan-out of 8.6 requests per task. The value sizes for the requests are generated using a Pareto distribution based on a study conducted on Facebook's Memcached deployment [1]. We then generate task inter-arrival times using a Poisson process where the mean rate is set to match 70% of system capacity. The experiments are then repeated 6 times with different random seeds.

Figure 2 depicts the read latencies averaged across experiments for different percentiles. The standard deviation is not shown as it is largely negligible. As shown, the credits strategy is at most 38% of an ideal model across different simulation settings. In addition, BRB outperforms C3 across all percentiles for both EqualMax and UnifIncr and improves the latencies by up to a factor of 3 at the median and 95^{th} percentiles and up to 2 times at the 99^{th} percentile.

3. REFERENCES

[1] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload Analysis of a Large-scale Key-value Store. In *SIGMETRICS*, 2012.

[2] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient Coflow Scheduling with Varys. In *SIGCOMM*, 2014.

[3] J. Dean and L. A. Barroso. The Tail At Scale. *Communications of the ACM*, 56(2):74–80, 2013.

[4] D. Shue, M. J. Freedman, and A. Shaikh. Performance Isolation and Fairness for Multi-tenant Cloud Storage. In *OSDI*, 2012.

[5] L. Suresh, M. Canini, S. Schmid, and A. Feldmann. C3: Cutting Tail Latency in Cloud Data Stores via Adaptive Replica Selection. In *NSDI*, 2015.