D-Tunes: Self Tuning Datastores for Geo-distributed Interactive Applications

Shankaranarayanan P N
Purdue University
spuzhava@purdue.edu

Ashiwan Sivakumar
Purdue University
asivakum@purdue.edu

Sanjay Rao
Purdue University
sanjay@purdue.edu

Mohit Tawarmalani
Purdue University
mtawarma@purdue.edu

ABSTRACT

Modern internet applications have resulted in users sharing data with each other in an interactive fashion. These applications have very stringent service level agreements (SLAs) which place tight constraints on the performance of the underlying geo-distributed datastores. Deploying these systems in the cloud to meet such constraints is a challenging task, as application architects have to strike an optimal balance among different contrasting objectives such as maintaining consistency between multiple replicas, minimizing access latency and ensuring high availability. Achieving these objectives requires carefully configuring a number of low-level parameters of the datastores, such as the number of replicas, which DCs contain which data, and the underlying consistency protocol parameters. In this work, we adopt a systematic approach where we develop analytical models that capture the performance of a datastore based on application workload and build a system that can automatically configure the datastore for optimal performance.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed databases

General Terms

Design, Performance, Reliability

Keywords

Storage networks, Wide-area replication

1 Introduction

Geo-replicated datastores have increasingly become a critical component of large scale online and internet applications. The interactive nature of these applications require very stringent service level agreements (SLAs) which place tight constraints on the performance the underlying datastores. Deploying these datastores to meet such constraints is a challenging task due to factors like (i) scale of applications (hundreds of millions of data items) (ii) heterogeneity in workloads within the same application (e.g., celebrities vs normal user in Twitter), (iii) contrasting objectives of maintaining consistency between multiple replicas, minimizing access latency and (iv) ensuring high availability. These challenges necessitate a system that can generate optimal replication strategies for the datastores from higher level application requirements.

SIGCOMM'13, August 12–16, 2013, Hong Kong, China.
ACM 978-1-4503-2056-6/13/08.

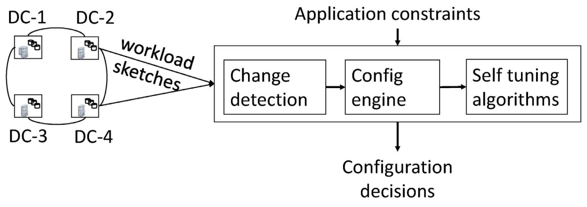


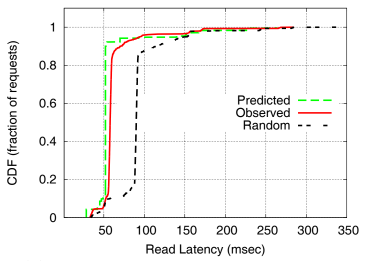
Figure 1: System Design

Contributions and related work: In this abstract, we present the design of our system that can automatically configure geo-replicated datastores in a manner adaptive to workload and network dynamics and customize strategies for different classes of data items. The novel contributions include: (i) systematic development of analytical frameworks that can optimally choose replication strategies that meet desired latency SLAs, consistency requirements and are robust to failures and network dynamics and (ii) design self-tuning algorithms that can adapt to workload changes over short timescales by automatically configuring parameters like replication factor, consistency levels and readjusting read/write priorities while judiciously recommending data placements over longer time-scales. While we initially focus on quorum-based systems [8], our approach is general and can be easily applied to a variety of datastores including Spanner [2].

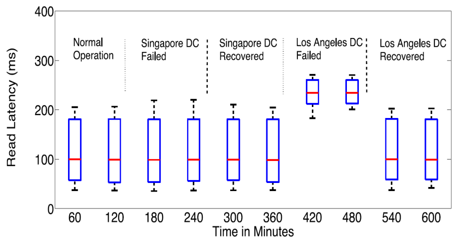
Replication strategies in geo-replicated datastores such as Spanner [2] and Cassandra [8] are manually configured by the application developers today. While prior works [7, 5] automate storage system designs, we focus on issues unique to geo-replicated datastores including low latency, stronger consistency and availability. Unlike replica placements in Content Delivery Networks (CDNs), and Facility Location problems, our problem is significantly different as we have to include stronger consistency requirements in our frameworks. In contrast to many theoretical works [6, 1, 9] on quorum protocols, our work is distinguished by its focus on geo-distributed datastores, adaptation to workload dynamics and performance under failure conditions.

2 System design

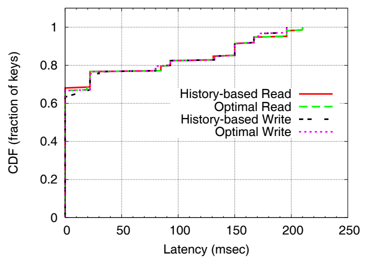
Figure 1 shows the architecture of our system. The key components and their functions are as follows. (i) The monitoring and change detection components monitor the accesses to the various *buckets* of data items and determine when the configuration changes are



(a) Benefits over random partitioner



(b) Impact of failures



(c) History-based vs Optimal

Figure 2: Insights from our experiments

necessary. We leverage techniques that were developed in the data streaming community called *sketches*, which is used for tracking statistics such as frequencies, quantiles, and for dynamically identifying popular items that account for a large fraction of requests [3, 4]. (ii) The configuration engine makes use of the analytical models we develop to determine the optimal replication strategy (number of replicas, placements, quorum sizes, etc.) and generate the appropriate configurations. A distinguishing aspect of our formulations is that it allows developers to specify different priorities and delay requirements on read/write traffic and generates configuration that ensure good performance even under failures. (iii) Finally, our self tuning algorithms push these configuration changes to the datastore while ensuring that there are no oscillations or rapid data migrations occurring in the system.

Analytical models and framework: Our framework models all parameters including the number of replicas, location of replicas and read/write quorum sizes which affect the read/write latency in quorum-based datastores. Our models also capture application constraints (e.g., read/write priorities, latency thresholds, availability) and workload dynamics (e.g., latency percentiles, read/write asymmetry). For our initial experiments, we developed two models that optimize replica configuration with the objective of minimizing access latency (i) *Basic Availability* model which is resilient to failures, and (ii) *N-1 Contingency* model which is not only resilient to failures but also ensures good performance under normal and failure conditions. Finally, we also include cost as an objective that can be minimized along with latency using our models.

3 Insights from experiments

We performed our initial experiments by deploying a Cassandra cluster (27 nodes, each corresponding to the AWS Edge locations) on an EC2 testbed and emulated the inter DC delays using Dummynet. We evaluated our approach using real-world application traces of Twitter, Wikipedia and Gowalla. We also performed extensive simulation studies using our models to study the benefits of our approach. Based on the above experiments, we present the following key insights.

- Configuration decisions at a lower granularity (say, *buckets* of data-items than for the entire database) can lead to significant benefits.
- Optimizing for different percentiles of requests significantly improves the overall performance.
- Diversity in the workloads like access locations, asymmetry in access patterns and relative priority between reads and writes can help improve the performance.
- Optimizing the configuration explicitly for performance under failures is critical.

Figure 2(a) shows the CDF of the read latency observed on the EC2 test-bed (with the Twitter trace) when using our optimal configurations and when using random partitioner of Cassandra. We see that our approach performs significantly better than random placements highlighting the importance of judicious configuration. Also, the observed latency on EC2 test-bed is very close to the model predictions which validates the accuracy of our models on a real deployment. Our experiments also show that it is critical to consider DC failures when configuring datastores. Figure 2(b) shows the read latency observed from our experiment on a Wiki article for which read and writes arrive from all over the world. Minimizing the access latency locates the replicas at the US and Asia (with a quorum of size 2). However, the failure of different replicas can degrade the performance of such configurations.

Finally, Figure 2(c) shows the validity of configurations across successive quarters for the Wikipedia trace. We find that the history based configuration (previous quarter) performs almost as good as the optimal configurations (in fact the curves overlap) for the current quarter indicating that frequent data migration is not required for most data items.

4 Conclusions

Our initial experiments highlight the importance and benefits of our framework in configuring geo-replicated datastores for good performance. We extend our experimental insights to build a system that optimally configures the replication parameters of geo-replicated datastores from the high-level application requirements, and continually tune them according to the network and workload dynamics.

5 References

[1] Y. Amir and A. Wool. Evaluating quorum systems over the internet. In *Proc. of FTCS*, 1996.

[2] J. C. Corbett, J. Dean, et al. Spanner: Google's globally-distributed database. *Proceedings of OSDI*, 2012.

[3] G. Cormode. Continuous distributed monitoring: a short survey. In *AlMoDEP*, 2011.

[4] G. Cormode and S. Muthukrishnan. Approximating data with the count-min data structure. *IEEE Software*, 2012.

[5] G. R. Ganger, J. D. Strunk, and A. J. Klosterman. Self-* storage: brick-based storage with automated administration. Technical report, DTIC Document, 2003.

[6] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *Journal of the ACM (JACM)*, 1985.

[7] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes. Designing for disasters. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, 2004.

[8] A. Lakshman and P. Malik. Cassandra:a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 2010.

[9] M. M.G., F. Oprea, and M. Reiter. When and how to change quorums on wide area networks. In *In Proc. SRDS*, 2009.