2015 IEEE 12th International Conference on Autonomic Computing

# Automatic Reconfiguration of Distributed Storage

Artyom Sharov,
Computer Science Department
Technion Israel Institute of Technology
Haifa, Israel 320004
Email: sharov@cs.technion.ac.il

Alexander Shraer, Arif Merchant and Murray Stokely
Google, Inc.
1600 Amphitheatre Pkwy
Mountain View, CA 94043
Email: {shralex, aamerchant}@google.com

***Abstract*—The configuration of a distributed storage system with multiple data replicas typically includes the set of servers and their roles in the replication protocol. The configuration can usually be changed manually, but in most cases, system administrators have to determine a good configuration by trial and error. We describe a new workload-driven optimization framework that dynamically determines the optimal configuration at run time. Applying the framework to a large-scale distributed storage system used internally in Google resulted in halving the operation latency in 17% of the tested databases, and reducing it by more than 90% in some cases.**

Globally distributed storage systems usually provide consistency across replicas of data (or metadata) using serialization or conflict resolution protocols. Distributed atomic commit or consensus-based protocols are often used when strong consistency is required, while simpler protocols suffice when replicas need only preserve weak or eventual consistency. Such protocols typically define multiple possible roles for the replicas, such as a leader or master replica coordinating updates, and appoint some of the replicas to participate in the commit protocol. The performance of such a system depends on the configuration: how many replicas, where they are located, and which roles they serve; for optimal performance, the configuration must be tuned to the workloads.

Applications using the same cloud storage service may have very different workloads. For example, an application responsible for access control may be read heavy, while a logging system may use the storage mostly for writes and have relatively few clients. The workloads can be extremely variable, both in the short term — for example, load may come from different parts of the world at different times of the day for a social application — and in the long term — the administrators of the service may reconfigure their servers periodically, causing different load patterns. Long term workload variation could also be due to organic changes in the demands on the Internet service itself; for example, if a shopping service becomes more popular in a region, its demands on the underlying storage system may shift.

Ideally, a cloud storage service should adapt to such changes seamlessly, since elasticity is an integral part of cloud computing and one of its most attractive premises. Reconfiguring a distributed storage system at run time while preserving its consistency and availability is a very challenging problem and can result in misconfigurations, which have been cited as a primary cause for production failures [12]. Due to its practical significance the problem has received abundant attention in recent years both in academia and in the industry (see [10], [5], [3] for tutorials on reconfiguring replicated state-machines and strongly consistent key-value stores), focusing mainly on the design and implementation of efficient and non-disruptive mechanisms for reconfiguration. Yet, little insight exists on how to set policies and use reconfiguration mechanisms in order to improve system performance. For example, dynamic reconfiguration APIs [11] have recently been added to Apache ZooKeeper [9], and users have since been asking for automatic workload-based configuration management, e.g. [2]. At Google, site reliability engineers (SREs) are masters in the dark arts of determining deployment policies and tuning system parameters. However, hand-tuning a cloud storage system that supports hundreds of distinct workloads is difficult and prone to misconfigurations.

We designed and implemented a workload-driven framework for automatically and dynamically optimizing the replication configuration of distributed storage systems to minimize operation latency. The framework was tested by optimizing a large-scale distributed storage system used internally in Google. Our approach relies on two main elements: a global monitoring infrastructure, such as the one available in Google, and a detailed knowledge of the storage operation flows. The former allows us to characterize storage workloads and estimate network latencies, while the latter is key to projecting end-to-end operation latencies with different alternative configurations.

**Storage Model.** We assume a common distributed storage model combining partitioning and replication to achieve scalability and fault tolerance: users (administrators) define databases, each database is sharded into multiple partitions, and each partition is replicated separately [1], [4], [6], [7], [8]. We call these partitions *replication groups*. Replication policies, defined by the database administrator (usually a systems administrator responsible for a specific client application), govern multiple replication groups in a database and determine the configuration of each group – the number of replicas, their locations and their roles in the replication protocol. For example, if most writers are expected to reside in western Europe, the administrator will likely place replicas in European locations and make some of them voters so that a quorum required to commit state changes can be collected in Europe without using expensive cross-continental network links. One of the voters (elected dynamically) acts as a leader and coordinates replication.

**Optimizing Leader Locations.** Leaders are involved in many of the storage operations, such as commits and consistent reads. Hence, their locations significantly affect operation latency. The first tier of our framework optimizes leader

978-1-4673-6971-8/15 $31.00 © 2015 IEEE
DOI 10.1109/ICAC.2015.22

placement, dynamically choosing a leader among eligible replicas, by taking into account the precise operation workload originating at each client location, the detailed flow of each operation type (given each potential leader location), and the observed network latencies.

We divide time into intervals and predict the optimal leader location for an interval based on the workload and network latencies observed in previous intervals. Specifically, at interval $i$ we (a) determine the average latency $t^{(i)}_{\alpha,c}(\ell)$ of each type of operation $\alpha$ from every client cluster $c$, for each potential leader location $\ell$ ranging over a set $V$ of replica locations eligible to become leader, and (b) quantify the number of operations $\mathsf{n}^{(i)}_{\alpha,c}$ of each type $\alpha$ for each client cluster $c$. Finally, we choose a leader $\lambda^{(i)}$ that minimizes the following expression:

$$\lambda^{(i)} = \arg\min_{\ell \in V}\{\mathit{score}^{(i)}(\ell)\} ,$$

where $\mathit{score}^{(i)}(\ell) = \sum_{\alpha,c} t^{(i)}_{\alpha,c}(\ell) \cdot \mathsf{n}^{(i)}_{\alpha,c}$. Location $\lambda^{(i)}$ can then serve as prediction for the optimal location $\lambda^{(i+1)}$ in the $(i+1)$-st time interval.

For simplicity of exposition, the expression above optimizes average operation latency and considers only the previous interval, but it can be extended to account for multiple intervals weighted according to recency and to minimize a given latency percentile, which can potentially be different for different databases.

Furthermore, we allow database administrators to assign weights to operation types, e.g., in order to prioritize read latency over commits, if desired, as well as output predictions for each operation type separately, to allow fine-grained placement control, if desired.

Our evaluation, using both simulations with production workloads and production experiments, shows that our methods are fast, accurate and significantly outperform previously proposed "common sense" heuristics, such as placing the leader close to the writers (e.g., in Google Megastore and Yahoo! PNUTS), which may work for one workload but not for another. Our experiments indicate that it results in a substantial speed-up for the vast majority of databases in our system, halving operation latency for 17% of the databases and reducing the latency by over 90% in some cases.

**Optimizing Replica Roles.** In many distributed storage systems different replicas may have different roles in the replication protocol. For example, in ZooKeeper (and most other Paxos-based systems) some of the replicas actively participate in the commit protocol, voting on state updates, while others passively apply committed updates to their state. While the number of voter replicas is usually determined based on availability requirements, the assignment of roles to specific replicas is flexible. For example, if 10 replicas are needed to support the expected read bandwidth and 2 simultaneous replica failures must be tolerated, then 5 voting replicas will likely be used. But which 5 out of the 10 should be voters? This question is answered dynamically, by the second optimization tier in our framework, based on the database workload. Note that once the optimal configuration has been determined, actually changing replica roles at runtime is usually inexpensive, as demonstrated in [11].

Usually, in such systems, the leader is one of the voters and thus optimizing replica roles not only affects the latency of commits but also the latency of other operations involving the leader. Our algorithm efficiently prunes sub-optimal configurations, achieving four orders of magnitude speedup compared to an exhaustive configuration search. Our evaluation demonstrate the benefits of dynamic role assignment for both write and read heavy workloads, achieving a speed-up of up to 50% on top of the first optimization tier, for some databases.

**Optimizing Replica Locations.** Finally, the third optimization tier dynamically determines the best replica locations out of a potentially large set of options. Our algorithms can further help determine the desired number of replicas. For example, if a database suddenly experiences a surge in client operations coming from Asia, we will detect the change in workload and identify the best replica locations to minimize latency. The first and second optimization tiers can then be used to assign different roles to the new set of replicas.

In summary, the main contribution of our work is the design and implementation of a new optimization framework for dynamically optimizing the replication policy in distributed storage systems. Our system facilitates self-configurable storage, freeing administrators from manually and periodically trying to adjust database replication based on the currently observed workloads, which is a very difficult task since such systems usually support hundreds or thousands of distinct workloads. Our evaluation and production experiments with a distributed storage system used internally in Google demonstrate dramatic latency improvements.

## REFERENCES


[1] Amazon dynamodb. http://aws.amazon.com/dynamodb/.

[2] Zookeeper feature request. https://issues.apache.org/jira/browse/ZOOKEEPER-2027, Retrieved February 10, 2015.

[3] M. K. Aguilera, I. Keidar, D. Malkhi, J.-P. Martin, and A. Shraer. Reconfiguring replicated atomic storage: A tutorial. *Bul. of EATCS*, 102, 2010.

[4] J. Baker et al. Megastore: Providing scalable, highly available storage for interactive services. CIDR, 2011.

[5] K. Birman, D. Malkhi, and R. van Renesse. Virtually synchronous methodology for dynamic service replication. Technical Report 151, MSR, Nov. 2010.

[6] S. Bykov, A. Geller, G. Kliot, J. R. Larus, R. Pandya, and J. Thelin. Orleans: cloud computing for everyone. In *ACM SOCC*, 2011.

[7] B. Cooper et al. PNUTS: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2), Aug. 2008.

[8] J. Corbett et al. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst.*, 31(3), Aug. 2013.

[9] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: Wait-free coordination for internet-scale systems. In *USENIX ATC*, 2010.

[10] L. Lamport, D. Malkhi, and L. Zhou. Reconfiguring a state machine. *SIGACT News*, 41(1):63–73, Mar. 2010.

[11] A. Shraer, B. Reed, D. Malkhi, and F. Junqueira. Dynamic reconfiguration of primary/backup clusters. USENIX ATC, 2012.

[12] Z. Yin et al. An empirical study on configuration errors in commercial and open source systems. In *SOSP*, 2011.