Parqua: Online Reconfigurations in Virtual Ring-Based NoSQL Systems

Yosub Shin, Mainak Ghosh, Indranil Gupta
Department of Computer Science
University of Illinois, Urbana-Champaign
Urbana, IL
{*shin14, mghosh4, indy*}*@illinois.edu*

***Abstract*—The performance of key-value/NoSQL storage systems is highly tied to the choice of (primary) key for the database table. As application requirements change over time, system administrators need to change the primary key of the table to improve performance. The primary key change is a specific example of a broader class of reconfiguration operations that affect a lot of data all at once. In this paper we propose a system called Parqua, which imbues ring-based key-value/NoSQL stores with the ability to perform reconfiguration operations in an online and efficient manner. We present the design and implementation of Parqua. Experiments based on our cluster deployments show that during reconfiguration Parqua maintains high availability, and with a small impact on read and write latencies.**

***Keywords*-reconfiguration; primary key; cassandra**

I. Introduction

Key-value/NoSQL systems today fall into two categories: 1) (virtual) ring-based and 2) sharded databases. Dynamo [1] and Apache Cassandra [2] rely on the use of a "virtual ring" to place servers as well as keys; keys are assigned to servers whose segment they fall into. On the other hand, MongoDB [3] and BigTable [4] rely on a fully flexible assignment of shards across servers.

In these databases, performing reconfiguration operations such as changing the primary key or the structure of the ring is a major pain point. They affect all of the data inside the table. Today's "state of the art" approach involves exporting and then shutting down the entire database, making the configuration change, and then re-importing the data. During this time the data is completely unavailable for reads and writes. This can be prohibitively expensive – for instance, anecdotes suggest that every second of outage costs $1.1K at Amazon and $1.6K at Google [5].

Reconfiguration operation is used extensively during database creation time. As workloads evolve, it is used infrequently but affects a lot of data on a live database. Thus, lack of an efficient online reconfiguration operation can lead to outages [6], and active discussion on JIRA [7], and blog posts [8], [9].

In our past work, we proposed a system Morphus [10] which solves this problem for *sharded* NoSQL databases such as MongoDB. It leverages the full flexibility of being able to assign any shards to any server, in order to derive an optimal allocation of shards to servers. Unfortunately, this also makes it unsuitable for *ring-based* key-value/NoSQL stores like Cassandra, since ring-based systems place data strictly in a deterministic fashion around the ring. Additionally, the ring-based systems do not allow isolating a set of servers for reconfiguration, which is integral to Morphus design.

As a result, we built Parqua[1], which enables online reconfigurations in virtual ring-based key-value/NoSQL systems. We have integrated Parqua into Apache Cassandra. Our experiments show that Parqua provides high nines of availability with little impact on read and write latency. The system scales well with data and cluster size.

II. System Model

Parqua is applicable to any key-value/NoSQL store that satisfies the following assumptions. First, we assume a distributed key-value store that is fully decentralized without the notion of a single master node or replica. Second, each node in the cluster must be able to deterministically decide the destination of the entries that are being moved due to the reconfiguration. This is necessary because there is no notion of the master in a fully decentralized distributed key-value store, and for each entry all replicas should be preserved after the reconfiguration is finished. Third, we require the key-value store to utilize *SSTable (Sorted String Table)* to persist the entries permanently. An SSTable is essentially an immutable sorted list of entries stored on disk [4]. Fourth, each write operation accompanies a timestamp or a version number which can be used to resolve a conflict. Finally, we assume the operations issued are idempotent. Therefore, supported operations are insert, update, and read operations, and non-idempotent operations such as counter incrementation are not supported.

III. System Design and Implementation

Parqua runs reconfiguration in four phases. The graphical overview of Parqua phases is shown in Fig. 1.

Next, we discuss these individual phases in detail.

This work was supported in part by the following grants: NSF CNS 1319527, NSF CNS 1409416, NSF CCF 0964471, and AFOSR/AFRL FA8750-11-2-0084.

[1]The Korean word for "change."

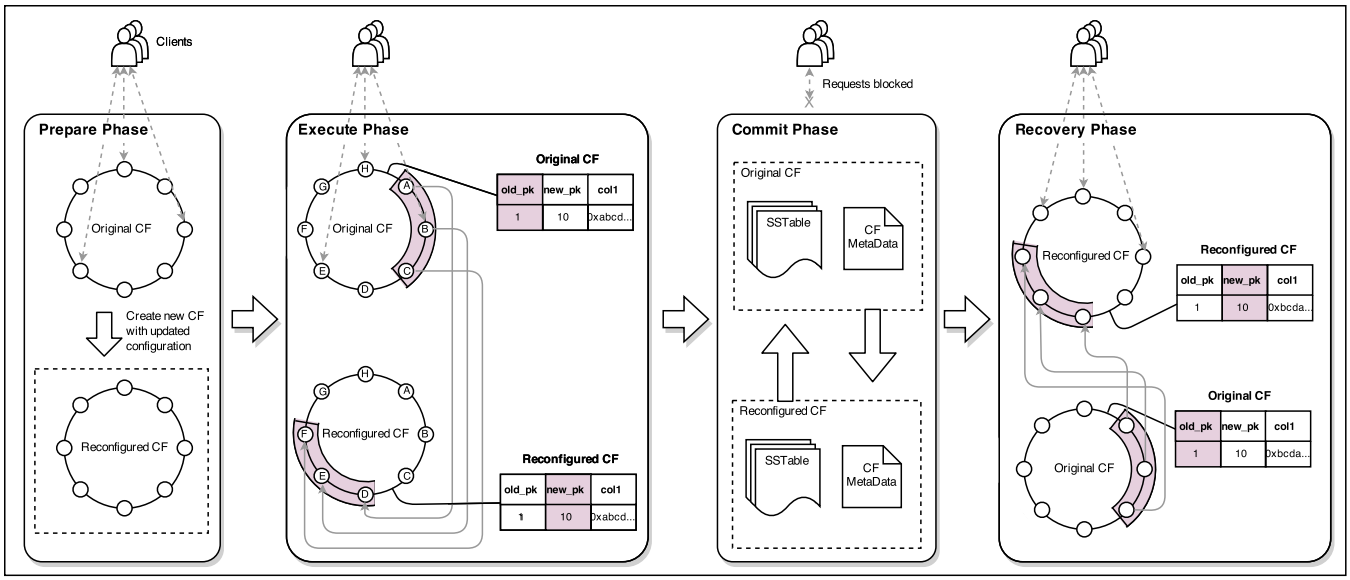


Figure 1: **Overview of Parqua phases. The gray solid lines represent internal entry transfers, and the gray dashed lines mean client requests. The phases progress from left to right.**

Prepare phase: In this phase, the initiator node – the node in which the reconfiguration command is run – creates a new (and empty) column family (database table), denoted as *Reconfigured CF (column family)*, using a schema derived from the Original CF except it uses the desired key as the new primary key. The Reconfigured CF enables reconfiguration to happen in the background while the Original CF continues to serve reads and writes using the old reconfiguration. We also record the timestamp of the last operation before the Reconfigured CF is created so that all operations which arrive while the Execute phase is running, can be applied later in the Recovery phase.

Execute phase: The initiator node notifies all other nodes to start copying data from the Original CF to the Reconfigured CF. Read and write requests from clients continue to be served normally during this phase. At each node, Parqua iterates through all entries that it is responsible for, and sends them to the appropriate new destination nodes. The destination node for an entry is determined by: 1) hashing the new primary key value on the hash ring, and 2) using the *replica number* associated with the entry. Key-value pairs are transferred between corresponding nodes that have matching replica numbers in the old configuration and the new configuration.

For example, in the Execute phase of Fig. 1, the entry with the old primary key '1' and the new primary key '10' have replica number of 1 at node A, 2 at B, and 3 at C. In this example, after primary key is changed, the new position of the entry on the ring is between node C and D, where node D, E, and F are replica numbers 1, 2, and 3, respectively. Thus, in the Execute phase, the said entry in node A is sent to node D, and similarly the entry in B is sent to E, and from C to F.

Commit phase: After the Execute phase, the Reconfigured CF has the new configuration and the entries from Original CF have been copied to Reconfigured CF. Now, Parqua atomically swaps both the schema and the SSTables between the Original CF and the Reconfigured CF. The write requests are locked in this phase while reads still continue to be served. To implement the SSTable swap, we leverage the fact that SSTables are maintained as files on disk, stored in a directory named after the column family. Therefore, we move SSTable files from one directory to another. This does not cause disk I/O as we only update the *inodes* when moving files.

At the end of the Commit phase, the write lock is released at each node. At this point, all client facing requests are processed according to the new configuration. In our case, the new primary key is now in effect, and the read requests must use the new primary key.

Recovery phase: During this phase, the system catches up with the recent writes that are not transferred to Reconfigured CF in the Execute phase. Read/write requests are processed normally. The difference is that until the recovery is done, the read requests may return stale results. [2] At each node, Parqua iterates through the SSTables of Original CF to recover the entries that were written during the reconfiguration. We limit the amount of disk accesses required for recovery by only iterating the SSTables that are created after the reconfiguration has started. The iterated entries are routed to appropriate destinations in the same way

[2]This is acceptable as Cassandra only guarantees eventual consistency.

as the Execute phase.

Since all writes in Cassandra carry a timestamp [11], Parqua can ensure that the recovery of an entry does not overshadow newer updates, thus guaranteeing the eventual consistency.

IV. Experimental Evaluation

A. Setup

We used the Yahoo! Cloud Service Benchmark(YCSB) [12] to generate the dataset, and used 'uniform', 'zipfian', and 'latest' key access distributions to generate CRUD workloads. Our default database size is 10 GB in all experiments. The operations consist of 40% reads, 40% updates, and 20% inserts.

	Read (%)	Write (%)
Read only	99.17	-
Uniform	99.27	99.01
Latest	96.07	98.92
Zipfian	99.02	98.92

Table I: **Percentage of reads and writes that succeed during reconfiguration.**

B. Availability

In this experiment, we measure the availability of our system during reconfiguration. The slight degradation in availability is due to the Commit phase when writes are rejected. The total unavailability time is a few seconds which is orders of magnitude better than the current state of the art.

The lowest availability is observed for the latest distribution. This is because YCSB does not wait for the database to acknowledge an insert of a key. As these keys are further read and updated because of the temporal nature of the distribution, the operations fail because the insert is still in progress.

C. Effect on Read Latency

Fig. 2 shows the CDF of read latencies under various workloads while reconfiguration is being executed. As a baseline, we also plot the CDF of read latency when no reconfiguration is being run using the Uniform key access distribution. We plot the latencies of successful reads only.

The median (50_{th} percentile) latencies for read-only workload and the baseline are similar because they both use uniform distribution. Under reconfiguration, 20% of the reads take longer. With writes in the workload, the observed latencies for the uniform curve are higher overall.

Compared to other workloads, latest has the smallest median latency. Due to its temporal nature, recently inserted keys are present in Memtables, which is a data structure maintained in memory. As a result, reads are faster compared to other distributions which require disk accesses.

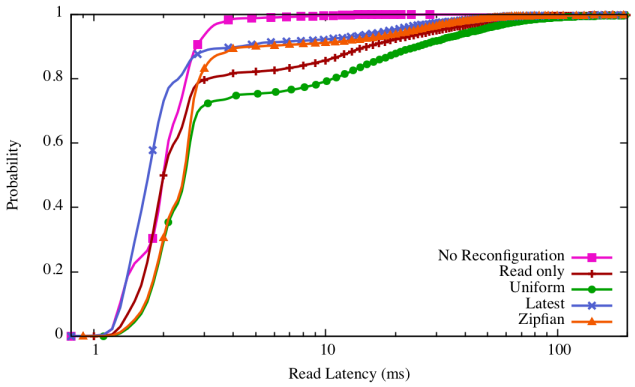


Figure 2: **CDF of read latencies for different workloads under reconfiguration, and the CDF of read latency under no reconfiguration for baseline measurement. The x-axis is read latency in logarithmic scale, while the y-axis is the cumulative probability.**

Overall, Parqua affects median read latency minimally across all the distributions. Our observations for write latency are similar. We refer the reader to our tech-report [13].

D. Scalability

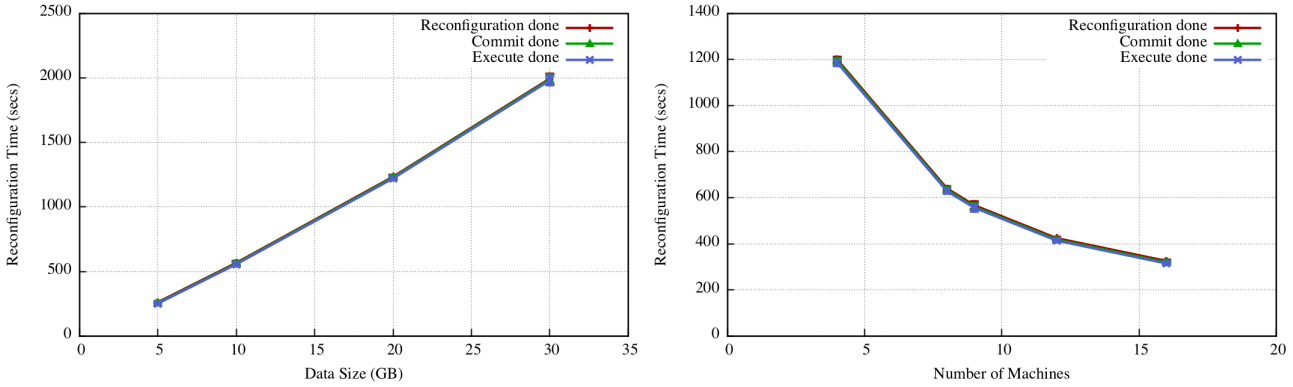
Next, we measure how well Parqua scales with: (1) database size, (2) cluster size, (3) operation injection rate, and (4) replication factor. Due to lack of space, we omit the plots for the last two experiments and refer the reader our tech-report [13]. To evaluate our system's scalability, we measured the total reconfiguration times along with the time spent in each phase. We do not inject operations for the experiments presented next.

1) Database Size: Fig. 3a depicts the reconfiguration time as the database size is increased up to 30 GB. Since we use a replication factor (number of copies of the same entry across the cluster) of 3 for fault-tolerance, 30 GB of data implies 90 GB of total data in the database. In this plot, we observe the total reconfiguration time scales linearly with database size. The bulk of the reconfiguration time is spent in transferring data in the Execute phase.

2) Cluster Size: In Fig. 3b, we observe that the reconfiguration time *decreases* as the number of Cassandra peers increases. The decrease occurs because as the number of machines increases, the same amount of data divided in smaller chunks gets transferred by a larger number of peers. Again, here the duration for the Execute phase dominated the reconfiguration time.

V. Summary

In this paper, we introduced Parqua, a system which enables online reconfiguration in a ring-based distributed key-value store. We introduced the general system assumptions for Parqua, and proposed its detailed design. Next, we integrated Parqua in Cassandra, and implemented a



(a) Data Size

(b) Number of Cassandra Machines

Figure 3: **Parqua Scalability with: (a) Data Size, and (b) Number of machines**

reconfiguration that changes the primary key of a column family. We experimentally demonstrated Parqua achieves good availability and scales well with database and cluster size.

REFERENCES

[1] "DynamoDB," http://aws.amazon.com/dynamodb/, visited on 2015-5-5.

[2] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[3] "MongoDB," http://www.mongodb.org, visited on 2014-04-29.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267308.1267323

[5] W. G. Yee, "Orbitz: Technical challenges and opportunities in a leading online travel business," https://sites.google.com/site/gcasrworkshop/2015/program/wai-gen-yee, 2015.

[6] "Troubles With Sharding - What Can We Learn From The Foursquare Incident?" http://highscalability.com/blog/2010/10/15/troubles-with-sharding-what-can-we-learn-from-the-foursquare.html, visited on 2015-04-11.

[7] "Command to change shard key of a collection," https://jira.mongodb.org/browse/SERVER-4000, visited on 2015-1-5.

[8] Stackoverflow, "Altering Cassandra column family primary key," http://stackoverflow.com/questions/18421668/alter-cassandra-column-family-primary-key-using-cassandra-cli-or-cql, visited on 2014-04-29.

[9] TechRepublic, "The great primary key debate," http://www.techrepublic.com/article/the-great-primary-key-debate/, visited on 2014-04-29.

[10] M. Ghosh, W. Wang, G. Holla, and I. Gupta, "Morphus: Supporting online reconfigurations in sharded nosql systems," in *12th IEEE International Conference on Autonomic Computing (ICAC 15)*. Grenoble, France: IEEE, 2015.

[11] "An Introduction to using Custom Timestamps in CQL3," http://planetcassandra.org/blog/an-introduction-to-using-custom-timestamps-in-cql3/, visited on 2015-04-25.

[12] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807152

[13] Y. Shin, M. Ghosh, and I. Gupta, "Parqua: Online reconfigurations in virtual ring-based nosql systems," University of Illinois, Urbana-Champaign, Tech. Rep., 2015, http://hdl.handle.net/2142/78185.