Database Caching and Sharding Strategies in Chained MR Jobs

Andrew Wong, Benjamin Hoertnagl-Pereira, Daniel Sohn, Ryan Demo

Johns Hopkins University | Whiting School of Engineering | Baltimore, MD
Cloud Computing Poster Session 2019

Introduction

Big data workloads in cloud environments must optimize resource usage to minimize job times. Big issues for job time involve servers encountering hardware bottlenecks to execute all the query requests, or lacking sufficient network bandwidth to respond with the results to the requester. Even if these are hardware limitations and not necessarily database dependent, response latency can be mitigated by a combination of sharding, replication and caching strategies dependent on workload. By routing the queries to different shards or load balancing over multiple copies of the database, parallelizing job tasks has the potential to reduce overall job time. Chained MapReduce jobs have had little profiling done to determine an optimal combination, or to provide guidance depending on workload. We look at the case of a read-heavy chained MR job reading uniformly at random from a database with a large keyspace to examine overhead and benefits of caching, sharding and replication.

Related Work

- Performance of web applications is too heavily workload dependant to have a optimal replication and caching setup. It is recommended to use virtual caching to first monitor cache hits and relative performance [2].
- While metadata persistence is still workload dependant, the frequency of database mutation has great effect on performance. Mutation frequency proportional to number of operations indicates that caching is more optimal while mutation frequency proportional to the number of jobs does better with replication [4].
- Taking advantage of the locality and correlation of user access, replication strategy for spatiotemporal data (RSSD) mines high popularity and associated files from historical user access information, generates replicas, and selects appropriate cache node for placement. Experimental results show that the RSSD algorithm is simple and efficient, and succeeds in significantly reducing user access delay [3].

References

1. Mat Keep. Scaling your MongoDB Atlas Deployment, Delivering Continuous Application Availability. MongoDB Best Practices. 2016.
2. Swaminathan Sivasubramanian, Guillaume Pierre, Maarten van Steen, and Gustavo Alonso. Analysis of Caching and Replication Strategies for Web Applications. 2007.
3. Lian Xiong, Liu Yang, Yang Tao, Juan Xu, and Lun Zhao. Replication Strategy for Spatiotemporal Data Based on Distributed Caching System. 2018.
4. Lin Xiao, Kai Ren, Qing Zheng, and Garth A. Gibson. ShardFS vs. IndexFS: Replication vs. Caching Strategies for Distributed Metadata Management in Cloud Storage Systems. 2015.

Design and Results

System Design

We built a 6-node set up on Google Compute Cloud:

- 1x MapReduce
- 1x Mongo Router + Config (running on separate ports)
- 4x shard + replica servers

MapReduce and Router:

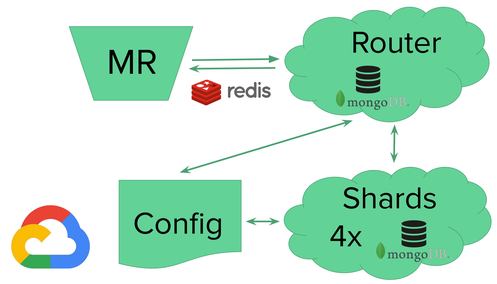
- 8 vCPUs
- 38GB RAM
- 40GB SSD
- High resources for data creation + cache

Shards:

- 1 vCPU
- 1.7GB RAM
- 40GB SSD

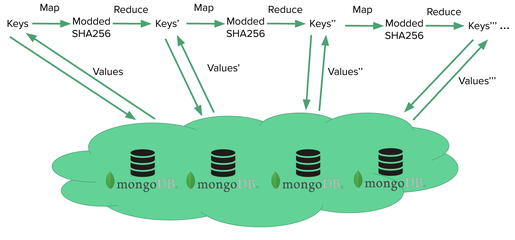
Redis:

- Up to 16GB



Each shard was initialized as a replica set with two replicas per shard. To maintain the shards the router node which acted as an interface for any incoming database requests and shards. It handled the data distribution (load balancing using the MongoDB balancer) and all queries were sent through this node. Furthermore, the router instance was built on top of a config instance which handled all the metadata of the shards and shard lookup.

Experiment Design



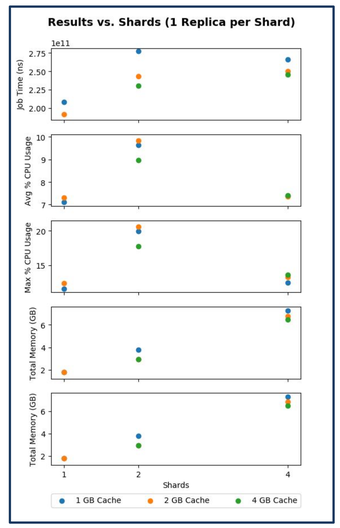
Mapper Output
Hashed DB entries mod (num entries) to approx. a random, uniformly sampled set of DB reads
Reducer Output
Unique keys

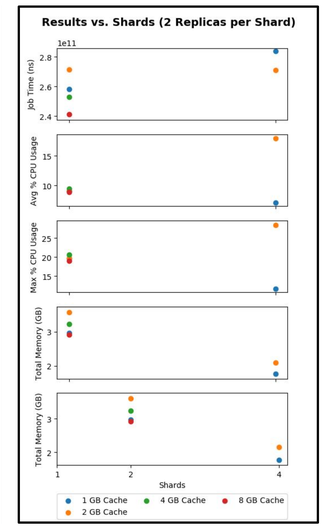
Independent Variables	Measurements (Sample Rate 200ms)
• Database Size: 8, 32 GB • Replicas per Shard: 1, 2 • Shards: 1, 2, 4 • Cache Size (GB): 0, 1, 2, 8, 16 • MR Chain Length: 1-8	• Job Completion Time (ns) • Average DB CPU % Usage • Max DB CPU % Usage • Average DB Memory Usage (KB) • Max DB Memory Usage (KB)

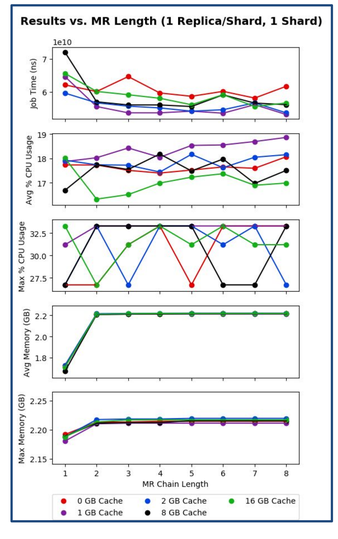
Future

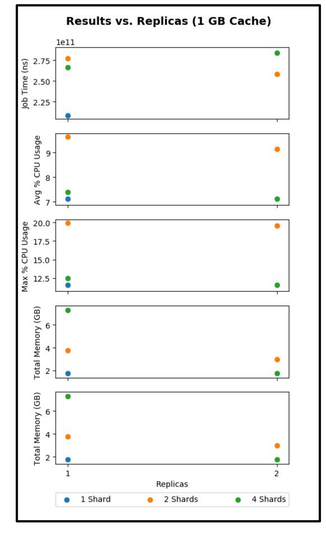
- Network: explore replica placement for fault tolerance, avoid GCP region limitation
- Database: consider MongoDB alternatives and SQL versus NoSQL on performance
- Implementation: use Hadoop Streaming and JobControl APIs

Results









- Replication had similar job time and less average CPU/memory usage compared to non-replicated
- 2, 4 shards had similar job times; both showed job time decrease as cache size went up
- Single node always won on job time, same CPU % as 4 shards
- Few cache hits for db workload with many documents
- Job time per MR chain stage trended downwards b/c of cache hits
- Larger caches had lower % CPU usage
- Cache performance improves with smaller key space, with the tradeoff being larger documents would fill the cache