INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

UGP REPORT

Auto Scaling of Key Value stores

Author:
Raghukul RAMAN

Supervisor:
Dr. Debadatta MISHRA

A report submitted in fulfillment of the requirements for Under Graduate Project (CS396A)

Department of Computer Science and Engineering

Abstract

In the modern day computing where application response is a critical metric, in memory databases are gaining popularity. In-memory databases primarily use main memory for data storage. These databases are becoming a crucial part of many applications due to the availability of less expensive RAM, and the demand for high response time. Some of the cases where in-memory databases perform exceptionally well are Queues, sticky sessions, caching, real-time data analysis, etc.

Generally, these in-memory databases are implemented as key-value stores; some well know key-value stores are Redis, Memcached, Hazelcast, etc. Large scale caching can be done by scaling the instances of these databases. Dynamic scaling of instances can provide cost-effective key-value storage services.This project aims at analyzing and providing an auto-scaling solution to this problem.

We try to collect statistics for different queries on different node configurations and try to find possible bottlenecks in the cluster based system. We analyze by scaling redis horizontally as well as vertically. For analyzing clusters we use an open source tool called Twemproxy (a fast, light-weight proxy for Memcached and redis). We limit resources and measure performance for these configurations. We also try to build a system which can be used to interact with the cluster and efficiently shard the requests among different instances.

Introduction

Key Value Stores

Key value stores are a special kind of database management system, where the data is organized in just 2 columns - a key, and a value. Here the key is mostly text, while the value can be anything and is simply an object. The actual data essentially becomes "value", which is indexed with "key", so whole data becomes schema-less and is just organized in just 2 columns. One important property is that since the data is indexed by just the keys, there is a lot of scope of distributing keys over different instances, making the system highly scalable.

These kind of databases fall into the category of nosql data stores, satisfying the BASE properties (Basically Available, Soft state, Eventually consistent). In the contezt of the CAP theorem (Consistency, Availability, Partition tolerance), these databases fall into the category of CP. Since these databases are generally implemented in cluster mode, we cannot gurantee availability on data.

These databases are generally in-memory which reside in main memory. In-memory databases are faster than disk-optimized databases because disk access is slower than memory access, the internal optimization algorithms are simpler and execute fewer CPU instructions.

Scalability

Scalability is the property of a system to handle a growing amount of work by adding resources to the system [*Bondi, Andre - Characteristics of scalability and their impact on performance, 2000*]. In layman terms, when you realise your system is getting slow and is unable to handle the current number of requests, you need to scale the system. Scaling can be done in two ways:

- **Vertical Scaling:** Increasing the resources in the server which we are currently using, i.e increase the amount of CPU, GPU, CPU, etc. Vertical scaling generally costly, also it may require the system to go down for a moment when the scaling takes place (need for downtime).
- **Horizontal Scaling:** Increasing the number of servers (instances). This would lead to decrease in the overall load on the system, if requests are sharded properly. This kind of scaling solution is typically popular in tech industries, as it makes the system fault tolerant (single point of failure reduced). Important advantage of this method is that it can provide administrators with the ability to increase capacity on the fly.

Redis

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker [*redis.io*] [**?**]

Design Details

References

[1] Redis documentation, `redis.io`

[2] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.