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 world where application response is a critical metric, in memory databases are gaining wide popularity. In-memory databases primarily relies on 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. Accessing data in memory eliminates seek time when querying the data, which provides faster and more predictable performance than disk. Some of the cases where in-memory databases perform exceptionally well are Queues, sticky sessions, caching, real-time analysis of data, etc.

Generally, these in-memory databases are implemented as key-value stores; some traditional ones are Redis, Memcached, Hazelcast, etc. Large scale caching can be done by scaling the instances of these databases. One way is running a fixed number of instances, while the other is to scale up/down based on the load that our sever gets. This project aims at analyzing and providing an auto-scaling solution to this problem.

First, 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 and try to explain the results. For analyzing clusters we use an open source tool called Twemproxy (a fast, light-weight proxy for Memcached and redis). We try to 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. Second part of the project is to give a scaling algorithm based on machine learning and some statistical predictions.

Introduction

AuctionIt allows auction designers to create and host auctions, users can place bids on items in the auction and can see the results and price they have to pay at the end of the auction process. Auction Designers can modify and create new fields which allow them to get more data from the bidders that is relevant to the auction being held. They can also choose from pre-defined auction templates to quickly set up some of the more common auctions. They can also define their custom allocation rules and pricing rules. This allows to set up complicated rules for pricing and allocation and thus provides more freedom to the designer to create a wide variety of auctions. The software, categorizes auction designers and bidders as different types of users. These accounts are password protected to prevent misuse and to ensure the auction is not compromised while it runs.
Before we started designing AuctionIt, we studied about auctions. We read papers/lectures to understand the reasons why first price and second price auctions are so popular and widely used. We also read and learnt about the auction process for large scale real life auctions take place like treasury auctions or spectrum auctions. We read about auctions with multiple objects and how these large scale auctions deal with having multiple objects which may be similar and how they setup rules to prevent allocating a majority of a commodity to a particular set of bidders.

Design Details

In this section we discuss the fabrication details for our software. First of all let's get familiar with the database structure.

Databases

- **User Table:** This table consist of all the data(login details) relevant to a particular bidder.
  - User ID: this id is unique for each user, it is initiated by a counter value and increments on addition of new user.
  - Login details: email ID, username, password (hashed) and salt.
  - Profile details: such as gender, date of birth etc.
  - Past auctions: this is stored in form of list which include all the past auction which this user has taken part in.

  We are storing the past auctions of a bidder, so that we can show his past bids with ease, without explicitly searching in all the auction table.

- **Designer Table:** This table contain information of all sellers or in other words auction creators. The features for a particular designer in our table are:
  - Designer ID: similar to user ID.
  - All the data that we have in user table (since a designer is also a user).
  - Past auction IDs: The auctions that this designer has created, along with the total revenue from that auction. Similar to user table, these are also stored in from of list of pair.
  - Pricing and Allocation rules: These contain the pricing rule ID and allocation rule ID of the custom allocation/pricing rule, that this designer has created/used.

  In this designer table we explicity store the auction ids, which this designer created in past so as to display their details, instead of checking each auction and getting all auction created by this designer explicity. Past auction will also hep the designer create new auction with slight modification of the past auctions. Same applies for storing pricing and allocation rules, he can use them in new auctions.

- **Default Auction Table:** These include the default auction that we provide in our system. A designer can easily import them, modify some of the features and make it live! Some of the features stored in this table are:
  - Default auction ID: unique to each auction.
  - Ranking table address.
  - Duration of auction.
  - Base price for the product.
  - Pricing rule: Stores the rule id for the rule which this auction uses.
  - Allocation rule: Similar to pricing rule.
  - Bid unit: Denotes the currency/mode of payment.
  - Revenue: net gain by this auction.
  - Bid step size: this signifies the net increase in bid value, in case of tie. For example in english auction the price of good is increased in step till only one user is willing to buy.
  - Item description: quantity, brand etc.
  - Organizer details.
  - Bidder qualifying criteria: that each bidder must satisfy in order to take part in.

  Note that some of the features listed here are only relevant to auctions which are live and running, for eg. revenue.

- **Live auction table:** This table stores the detail corresponding to a running auction. Some features of this table are:
    - Auction ID: unique to each auction.
    - Auction label: In case of sequential auction this feature is relevant, where subsequent auction is similar to previous one.
    - Designer ID: user ID of owner.
    - Creation date
    - Start / End date: denotes the duration when auction would be live.
    - features from default auction tables: since each auction is an instance of default acution with some new features.
    - Ranking table address: a pointer to the table which stores the live bids of this auction.
- **Ranking table:** this stores the bids for a particular auction. Note that for each auction we'll create an instance of this table, and store the pointer to this table in live auction table. it stores:
    - Serial No.
    - Username
    - Bid value

  Whenever a user want to see the current leaderboard, we'll simply apply the allocation rule, and list out bidders in sorted order, with respect to their points.
- **Allocation/Pricing rule table:** two tables containing designer ID, rule ID, raw code for the rule and the rule name (optional).

References

[1] Leonardo Bartolini and Carlo Cottarelli. *Designing Effective Auctions for Treasury Securities*. Current issues in `ECONOMICS` and `FINANCE`, *Vol. 3 No. 9, July 97*

[2] Ravi Shankar and Sanjoy Bose. *Auctions of Government Securities in India –An Analysis.* Reserve Bank of India Occasional Papers, *Vol. 29 No. 3, Winter 2008*

[3] Kenneth D. Garbade and Jeffrey F. Ingber. *The Treasury Auction Process: Objectives, Structure, and Recent Adaptations.* Current issues in `ECONOMICS` and `FINANCE`, *Vol. 11 No. 2, Feb 06*

[4] Paul R. Milgrom. *Auctions and Bidding: A Primer.* Journal of Economic Perspective, *Vol. 3 No. 3, Summer 89*

[5] Paul R. Milgrom and Robert J. Weber. *A theory of auctions and competitive bidding - II.* 1981

[6] Kevin Leyton-Brown, Paul R. Milgrom and Ilya Segal. *Economics and computer science of a radio spectrum reallocation.* PNAS, *Vol. 114 No. 28, 2017*

[7] Paul R. Milgrom. *Business Activities.*
`http://www.milgrom.net/business-activities`