

# Auto Scaling of Key Value stores

Raghukul Raman

IIT Kanpur

April 26, 2019

# Outline

- 1 Abstract
- 2 Introduction
- 3 Experiments
- 4 Observations
- 5 Further works

# Outline

- 1 Abstract
- 2 Introduction
- 3 Experiments
- 4 Observations
- 5 Further works

- Collect statistics for different queries on different scaling configurations and try to find possible bottlenecks in the system.
- Providing an auto-scaling solution for key value stores.

# Outline

- 1 Abstract
- 2 Introduction**
- 3 Experiments
- 4 Observations
- 5 Further works

# Key Value Stores

- Data is organized in just 2 columns - a **key**, and a **value**.
- Actual data is value - can be object, keys are used to index these data objects.
- Satisfy **BASE** property.
  - Basically Available
  - Soft state
  - Eventually consistent
- **CAP** theorem
  - Consistency
  - Availability
  - Partition tolerance
- Fall into CP category.
- Mostly In-memory - extremely fast compared to traditional DBs.

- *def*: Property of a system to handle a growing amount of work by adding resources to the system[Bondi, Andre - 2000].
- Two variants:
  - **Vertical Scaling**: Increasing the resources in the server which we are currently using, i.e increase the amount of memory, CPU etc.
  - **Horizontal Scaling**: Increasing the number of servers (instances).
- Benefits of horizontal scaling:
  - make system fault tolerant.
  - down time.
- Several load balancing schemes are used in horizontal scaling.

- developed by Salvatore Sanfilippo(*antirez*).
- Open source, in-memory data structure store, used as a database, cache and message broker.
- Supports other data structures like: strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs.
- Built in support for replication, on-disk persistence and automated partitioning with Redis cluster.
- Master-slave asynchronous replication.
- Transaction, expiration time (BigTable).



# Redis Cluster

- Collection of redis nodes
  - Able to communicate among themselves.
  - Able to respond to requests collectively.
- Data is automatically sharded across multiple Redis nodes.
- Every redis node require 2 ports:
  - Lower one is used to server clients.
  - Other used for Cluster Bus (node-to-node communication).
- CRC16 hashing system - 16384 hash slots.
- Each cluster node is responsible for a subset of hash slots.
- Hash tags.
- After cluster meet, every node contain node - hash slot mapping.

- Redis Cluster supports the ability to add and remove nodes while the cluster is running.
- Same basic mechanism can be used in order to rebalance the cluster, add or remove nodes.
- Moving hash slots through Cluster Bus.
- No down time - with the help of MOVED Error.

# Redirection

- Client knows nothing - can send request to any node.
- On receiving a request:
  - return the value if serves that hash slot.
  - MOVED error, if not served.
- MOVED error response also contain IP:PORT of node serving that hash slot.
- Still is an overhead.

# Partitioning

- The way how we shard data among different nodes of redis cluster.

# Outline

- 1 Abstract
- 2 Introduction
- 3 Experiments**
- 4 Observations
- 5 Further works

hello2.5

# Outline

- 1 Abstract
- 2 Introduction
- 3 Experiments
- 4 Observations**
- 5 Further works

intro

hello3



# Outline

- 1 Abstract
- 2 Introduction
- 3 Experiments
- 4 Observations
- 5 Further works**

hello4