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

Aim

- 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 - extremy fast compared to traditional DBs.

Scalability

- *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.

Redis

- 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 persistance 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.

Resharding

- 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 recieving 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.

Partioning

- The way how we shard data among different nodes of redis cluster.
- **Client side partitioning:** redis clients select the node to which read/write request need to be made.
- **Proxy assisted partitioning:** client sends request to a proxy, which analyzes the request and forwards it to the correct node.
- **Query Routing:** can send request to any node, and the node will forward our request to the desired correct node.
- Examples: Jedis, twemproxy, Redis Cluster.

Twemproxy

- Proxy assisted partitioning implementation.
- It maintains **persistent** server connections.
- Shard data automatically across multiple servers, keeps copy of node configurations.
- Not a single point of failure.
- Stats monitoring port.

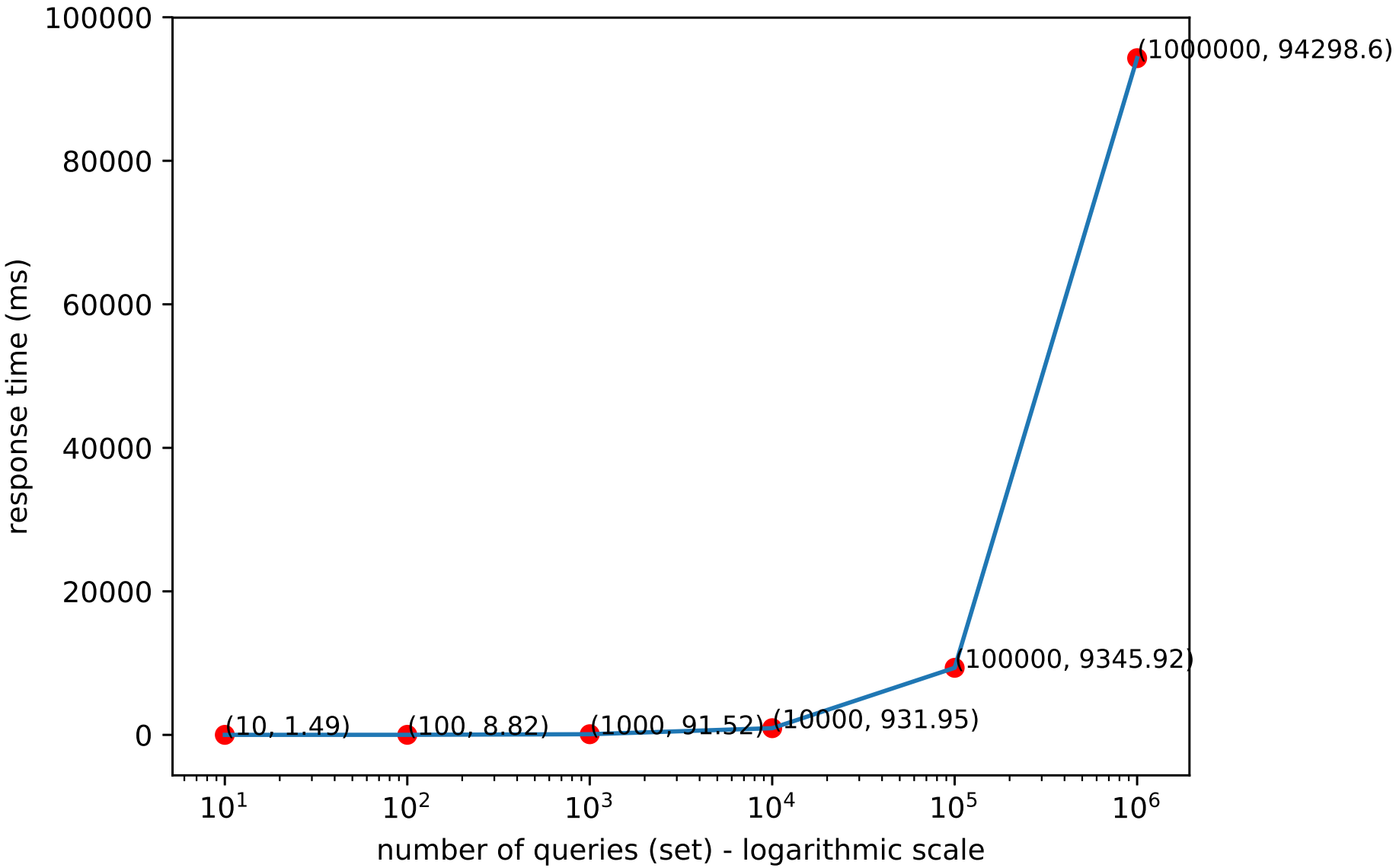
Outline

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

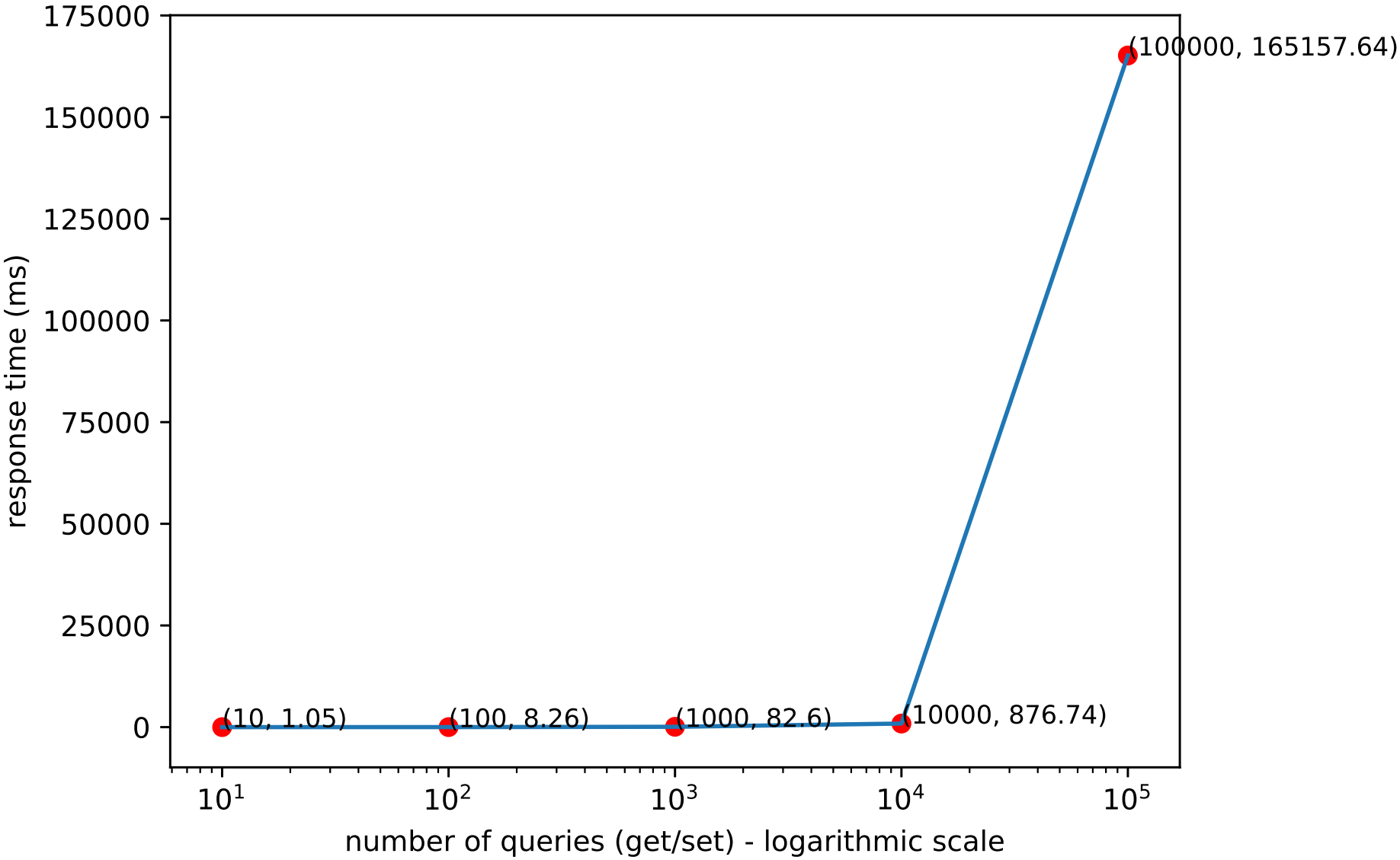
Experiments

- Collecting stats - by limiting resources.
- Generate random strings of length 64 - mimic SHA2.
- Tests based on get/set commands.
- Resource limitation enforced using docker.

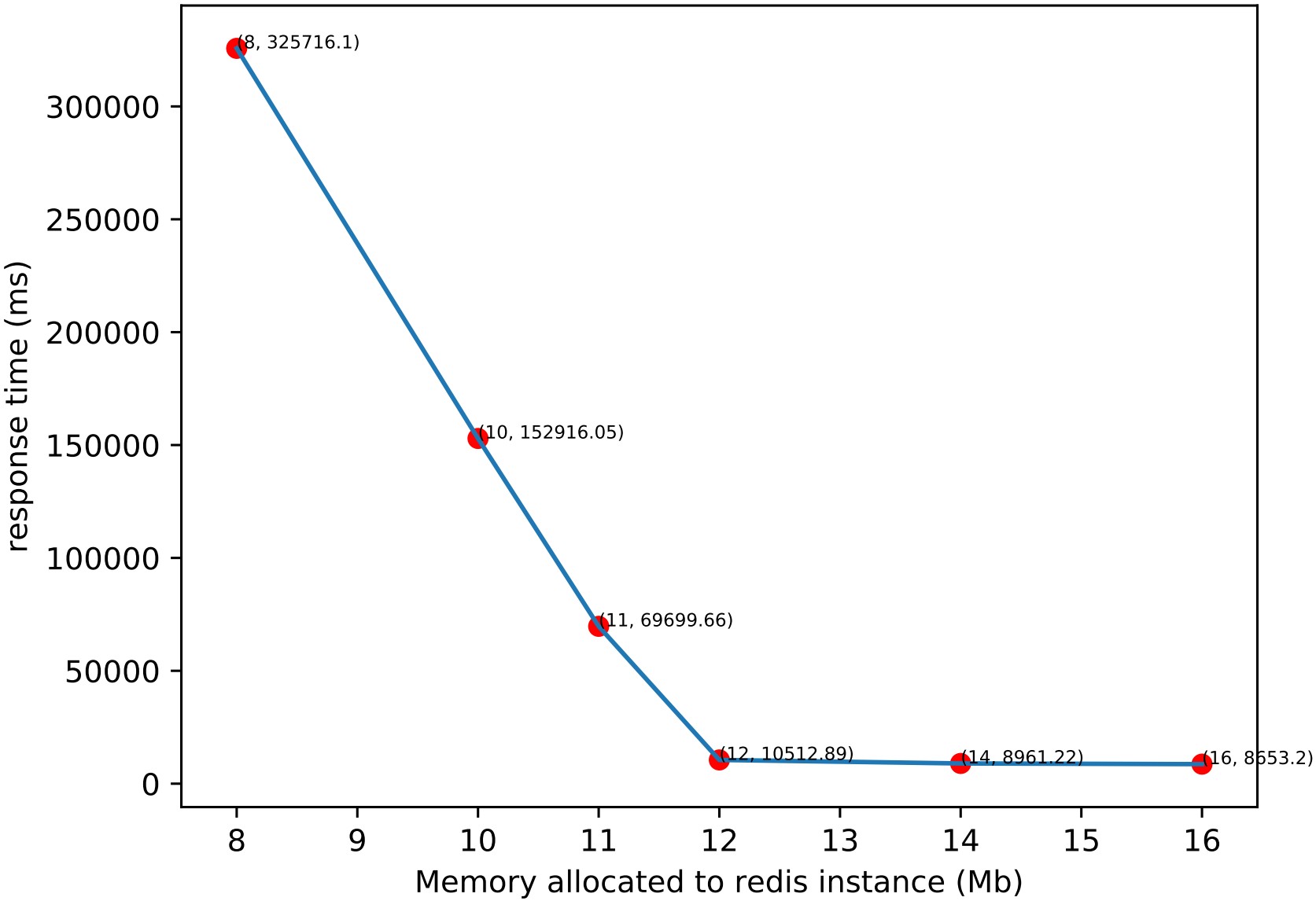
Single redis instance, with no memory/cpu restriction



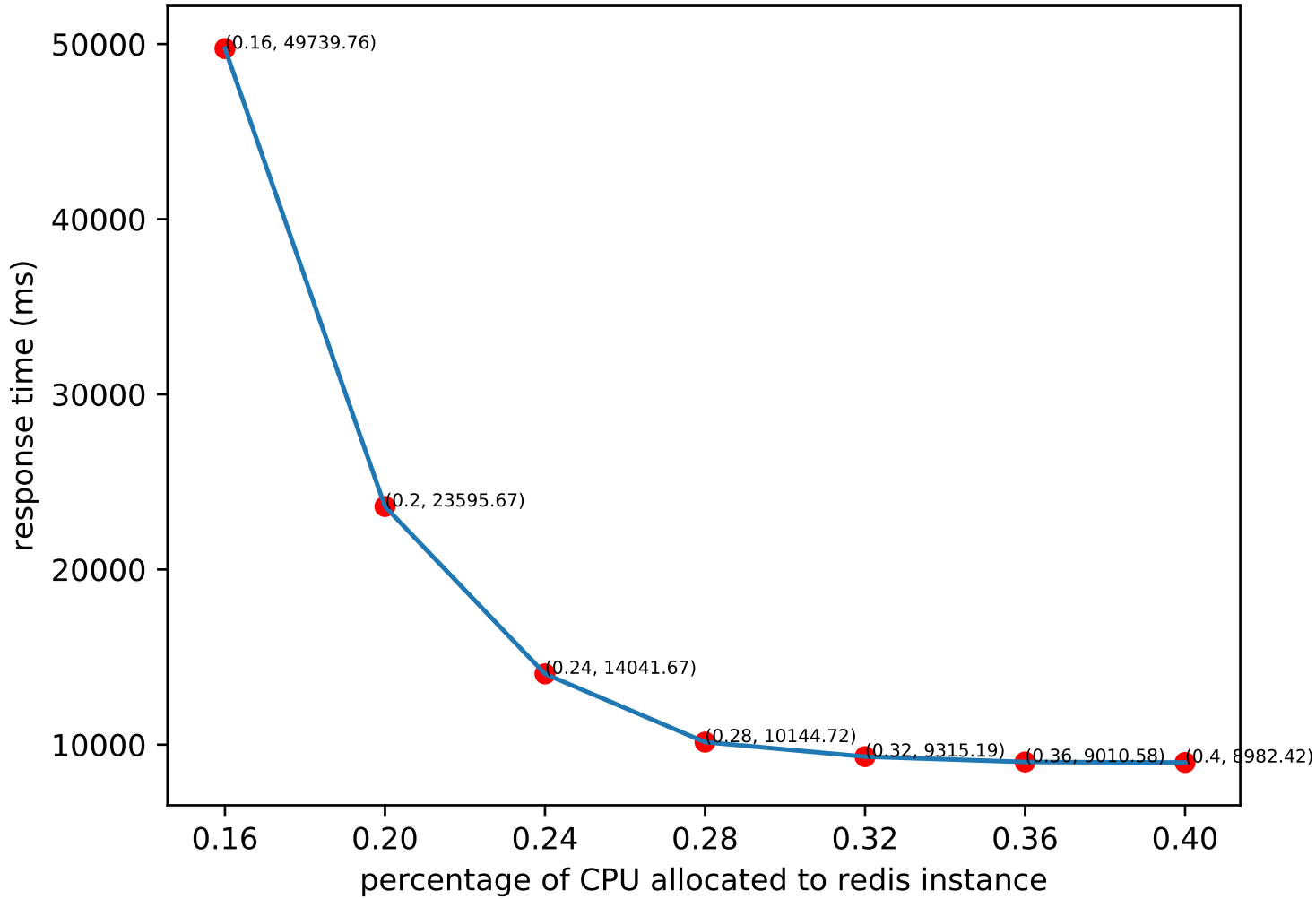
Single redis instance, with max main memory = 10Mb



10^5 queries on redis instance, with varying main memory



10^5 queries on redis instance, with varying CPU



- fractional CPUs?

Outline

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

hello3

Outline

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

hello4