Key-Value Store Databases

Advanced Databases

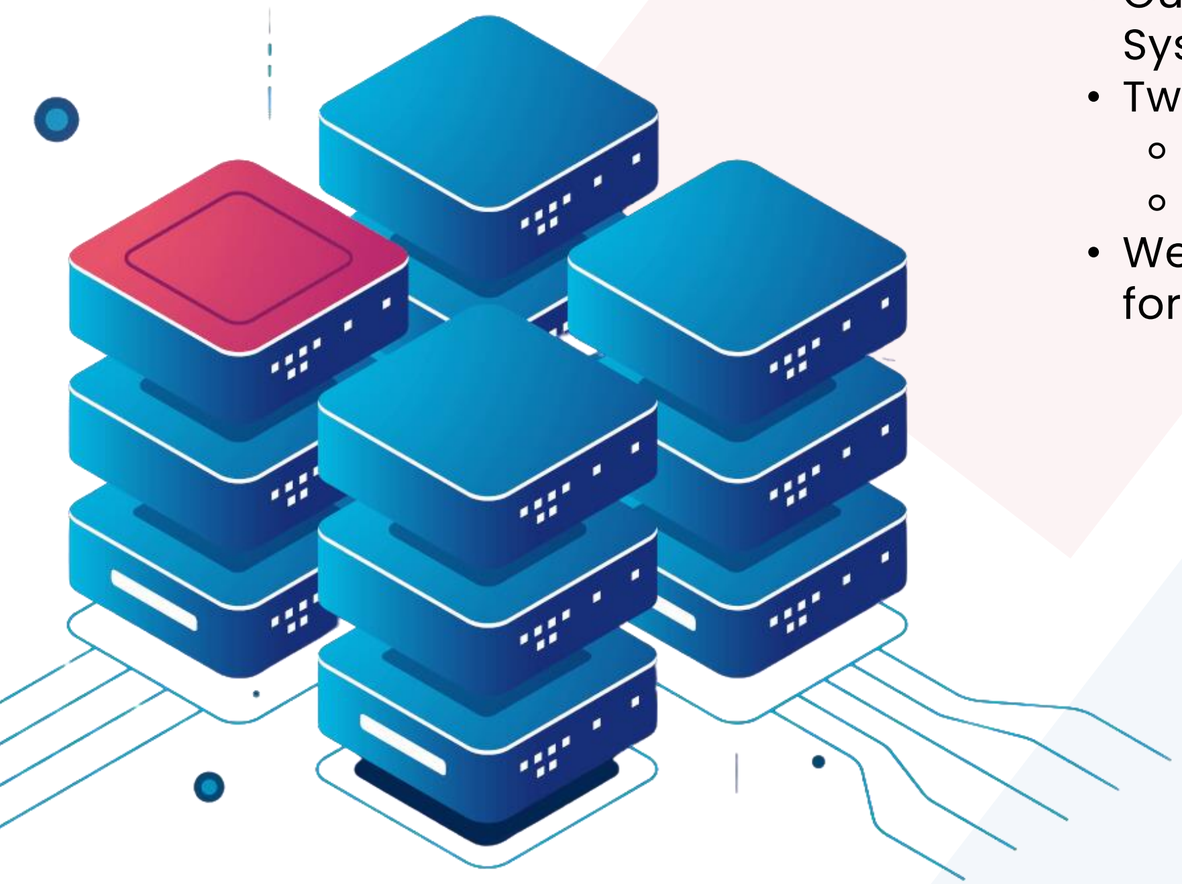
INFO-H415

Presented by:
Mayr Dionisius
Elezi Herma
Suau Thomas
İşlek Rana

Table of Content

~ Part 1 ~
Introduction to technology: Key-Value Storing

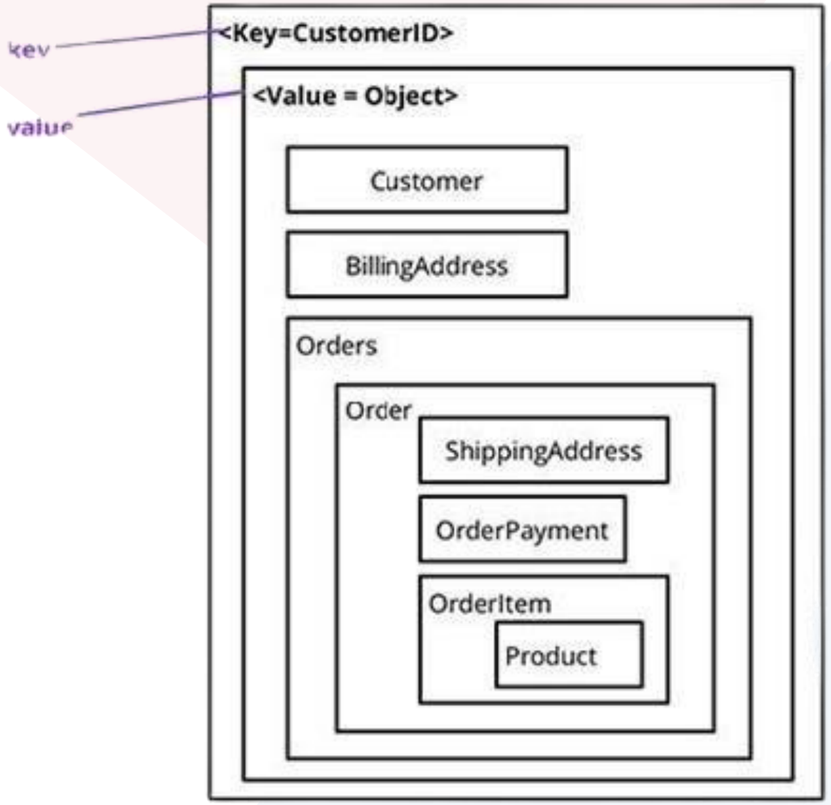
Introduction



- Our project aims to select the right Database Management System for a caching application;
- Two technologies are being compared:
  - Traditional relational databases
  - Key-value stores
- We aim to answer which technology would be more suitable for our application.

Key-Value Stores

Key-value stores pair unique keys with specific values. They're schema-less for data flexibility, and are known for fast data retrieval, scalability, and simple data models.



Key-Value Stores

Importance in Modern Database Systems

- Crucial Role
- Large Volumes of Data
- Quick and Efficient Data Access
- Use Cases
- Simplicity

Key-Value Stores

Advantages

High Performance

Scalability

Flexibility

Key-Value Stores

Use Cases

Caching

Session Storage

~ Part 2 ~
Introduction to the two tools:
Redis and etcd
+
Advantages and Disadvantages

Redis

ReThink Caching

- Remote Dictionary Server
- Open-source
- In-memory data structure store
- Various data structures:
  - strings
  - hashes
  - lists
  - sets
  - ...
- Designed for high performance:
  - Low-latency access

Redis

FEATURES

Data Structures

Redis supports a variety of data providing flexibility for modeling

Persistence

Redis offers options for persistence, allowing data to be saved to disk for durability

Replication

Redis supports master-slave replication, enabling redundancy and high availability.

Partitioning

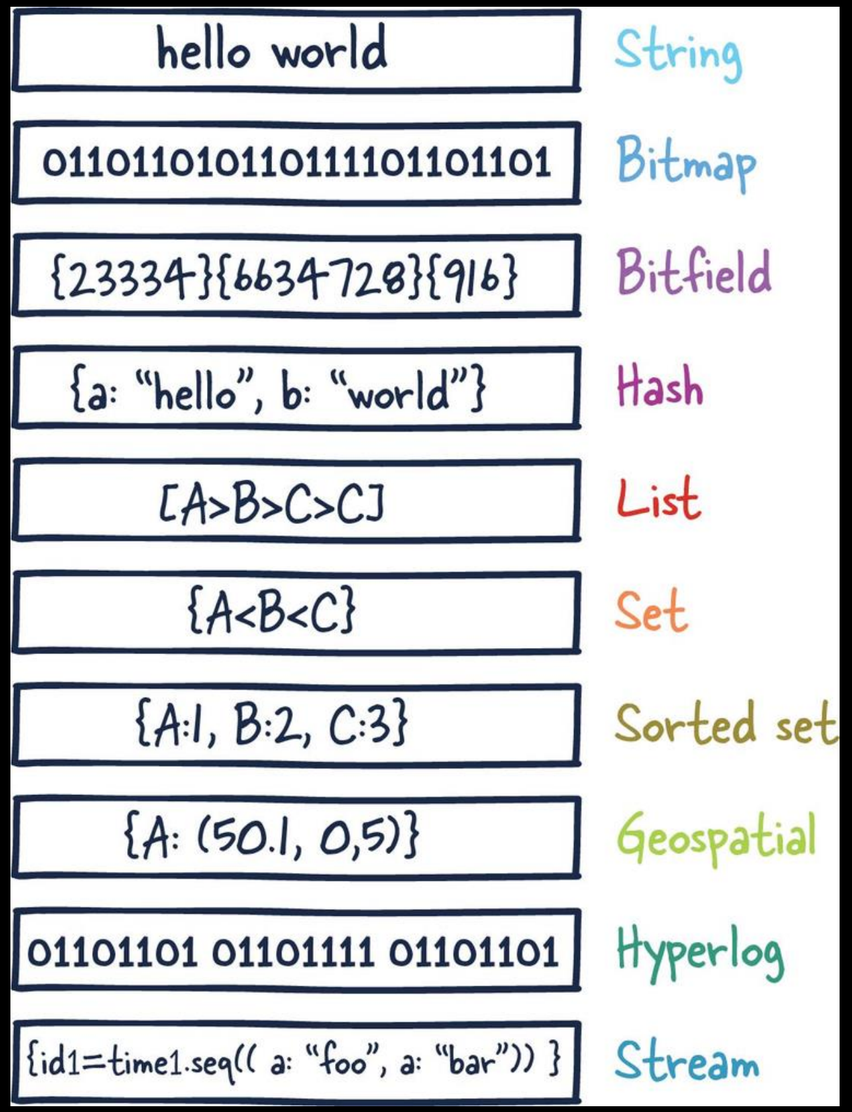
Redis can be horizontally scaled through sharding, distributing data across multiple nodes.

Redis 

(MOSTLY) A SINGLE THREADED APPLICATION

- Generally IO bound -> Parallelism wouldn't help in many cases
- It makes it simple
  - No thread synchronization (fast!)
  - Lockless atomic actions
  - Easier to maintain
- Where is it multi-thread?
  - Some small thread safe contexts are being moved to separate threads
  - Still, at any time Redis will still be serving a **single** request.
- Sharding can help CPU-bound applications

Redis



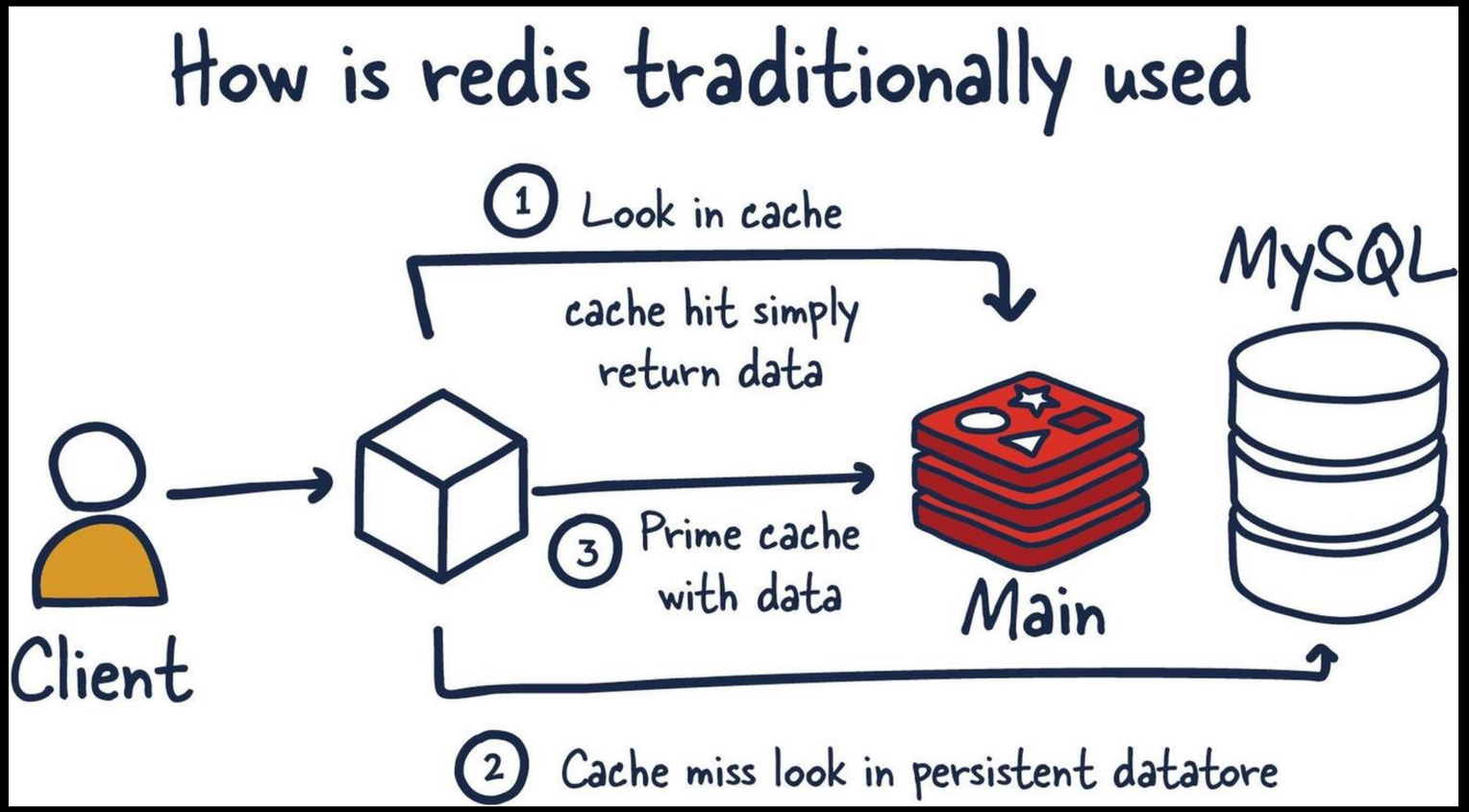
Source:

Redis 

SIMPLE

- Initial version developed in a short amount of time
- Easy to use
  - `SET bike:1 "My Bike"`
  - `HSET bike:1 model Deimos brand Ergonom type 'Enduro bikes' price 4972`
  - `HGET bike:1 price`
  - `BITFIELD pla:1:stats INCRBY u32 #0 50 INCRBY u32 #11`
  - `EXPIRE bike:1 10`
- Quite popular

Redis



Source:

Redis

USE CASES

Caching

Due to its in-memory nature, Redis is widely used as a caching layer to store frequently accessed data for quick retrieval

Session Storage

Storing session data in Redis ensures fast access and efficient management of user sessions in web applications

Autocomplete

The fast read and write capabilities, together with the data structures provided make Redis suitable for auto-completing applications (such as recent contacts)

Redis

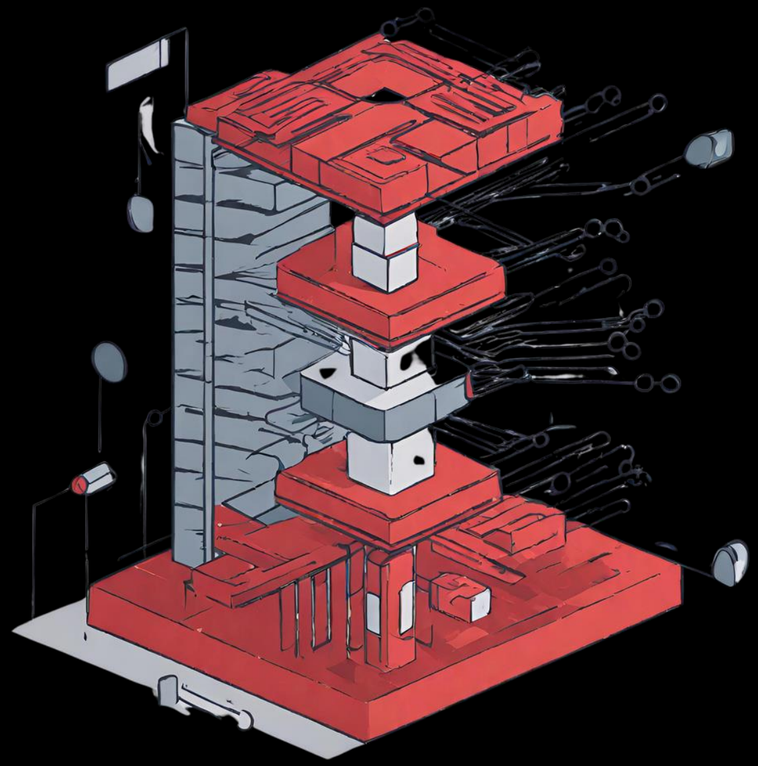
STRENGTHS

Performance

Redis excels in terms of performance, with low-latency access to data.

Versatility

Leveraging the data structures provided makes it versatile enough to be used in multiple use cases.



WEAKNESSES

Single-threaded

For some workloads , the single-threaded nature may become a bottleneck.

Complex Queries

Redis may not be the best choice for scenarios that involve complex queries, as it prioritizes simplicity and speed.

etcd

Distributed Reliable Key-Value Store

A distributed, reliable key-value store for the most critical data of a distributed system

etcd

WHAT IS etcd?

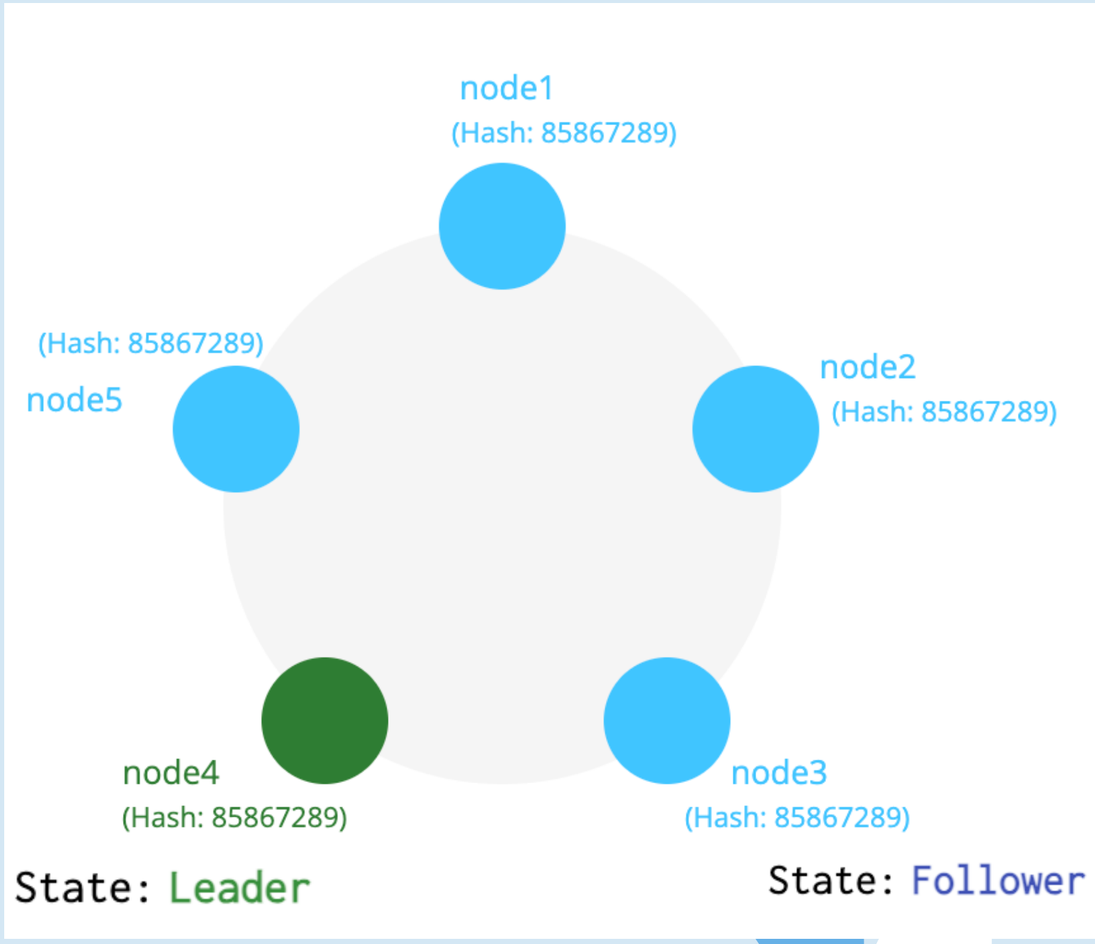
etcd, short for "**distributed et cetera**," is an open-source distributed key-value store designed for reliability, consistency, and simplicity in distributed systems.
The distributed key-value store etcd is specifically designed for managing configuration data, coordinating distributed systems (clusters), and ensuring strong consistency through **the Raft consensus algorithm**.
It’s the core component of *Kubernetes* -> mainly used stores and manage state data, configuration data and metadata.

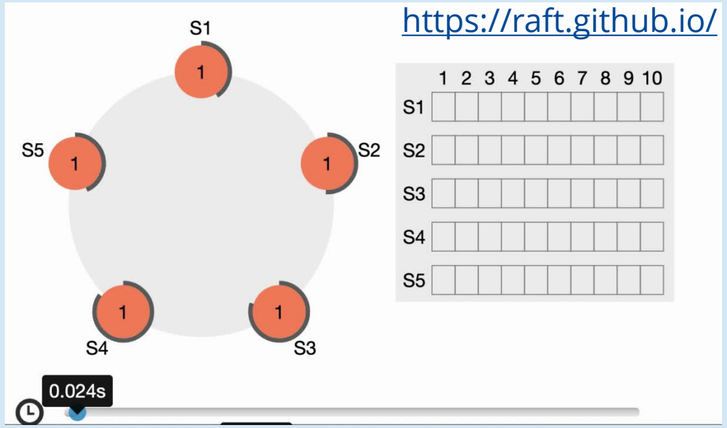
etcd

Architecture

- etcd is built around the **Raft consensus algorithm -›** providing a fault-tolerant and consistent distributed key-value store.
- It operates in a cluster model to maintain data consistency.
- The Raft algorithm ensures that a **leader** is elected to *coordinate* and *replicate* data across the cluster

The architecture is **very robust**, but there is a *loss in IO write speed*.





etcd

Key Features

Raft Consensus Algorithm

etcd employs the Raft algorithm to ensure strong consistency and fault tolerance, providing a reliable foundation for distributed systems.

Distributed Configuration

etcd excels in managing dynamic configurations across distributed applications, ensuring that all nodes have access to the latest configuration settings.

Atomic Transactions

etcd supports atomic transactions, allowing a series of operations to be executed as a single, indivisible unit based on If/Then/Else block statements.

Watch Notifications

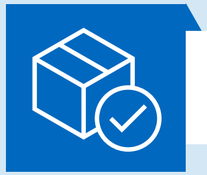
Developers can watch for changes in etcd key-value pairs, enabling real-time notifications when configuration data is updated.

etcd

Use Cases

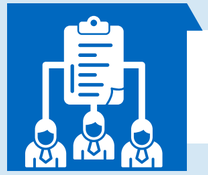
Kubernetes clusters management

etcd is the core component of Kubernetes to store and manage state data, configuration data and metadata.



Service Discovery and Load Balancing

etcd can be used as a central registry to store information about available services, their locations, and health status. Each service instance registers itself in etcd, making it discoverable by other services.



Distributed Locking and Coordination

etcd can coordinate distributed systems by providing a consistent view of shared data and managing distributed locks.

etcd

Strengths and Weaknesses

Consistency and reliably

etcd ensures strong consistency through the Raft consensus algorithm, making it reliable for critical data management.

Fault Tolerance

distributed nature of etcd, combined with Raft, provides fault tolerance, ensuring system reliability in the face of node failures.

Performance for Key-Value Storage

While etcd is robust in consistency and coordination, it may not match the raw performance of in-memory key-value stores like Redis, especially in scenarios prioritizing low-latency access.

Specialized Use Case

etcd is designed for specific use cases. etcd won't be efficient for other application purposes.

~ Part 3 ~
Methodology and Sample Application

Methodology

Methodology

Yahoo! Cloud Serving Benchmark (YCSB)

- Industry standard benchmark for over a decade
  - Focus on key-value and cloud serving stores
- Simulates multiple load scenarios
  - Time and throughput
- go-ycsb wrapper

Methodology

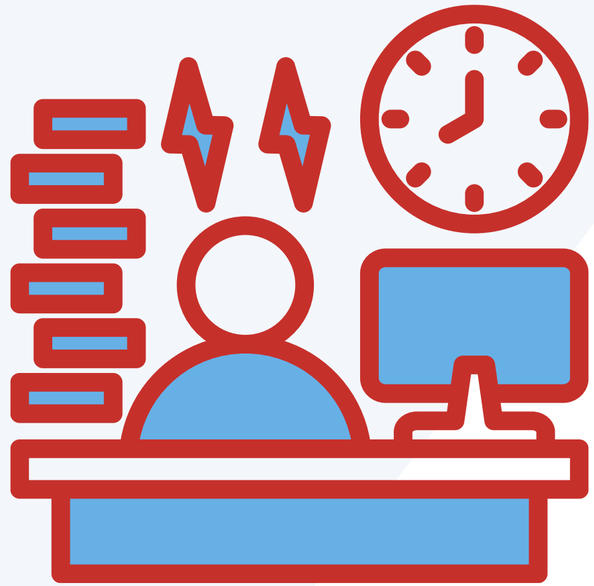
Workloads

- Other parameters:
  - Number of columns generated
  - Number of threads

	Proportion				
Workload	**Read**	**Update**	**Scan**	**Insert**	**Read Modify Write**
A: Update heavy workload	0.5	0.5	0	0	0
B: Read mostly workload	0.95	0.05	0	0	0
C: Read only	1	0	0	0	0
D: Read latest workload	0.95	0	0	0.05	0
E: Short range	0	0	0.95	0.05	0
F: Read-modify-write	0.5	0	0	0	0.5

Methodology

Core workloads flow



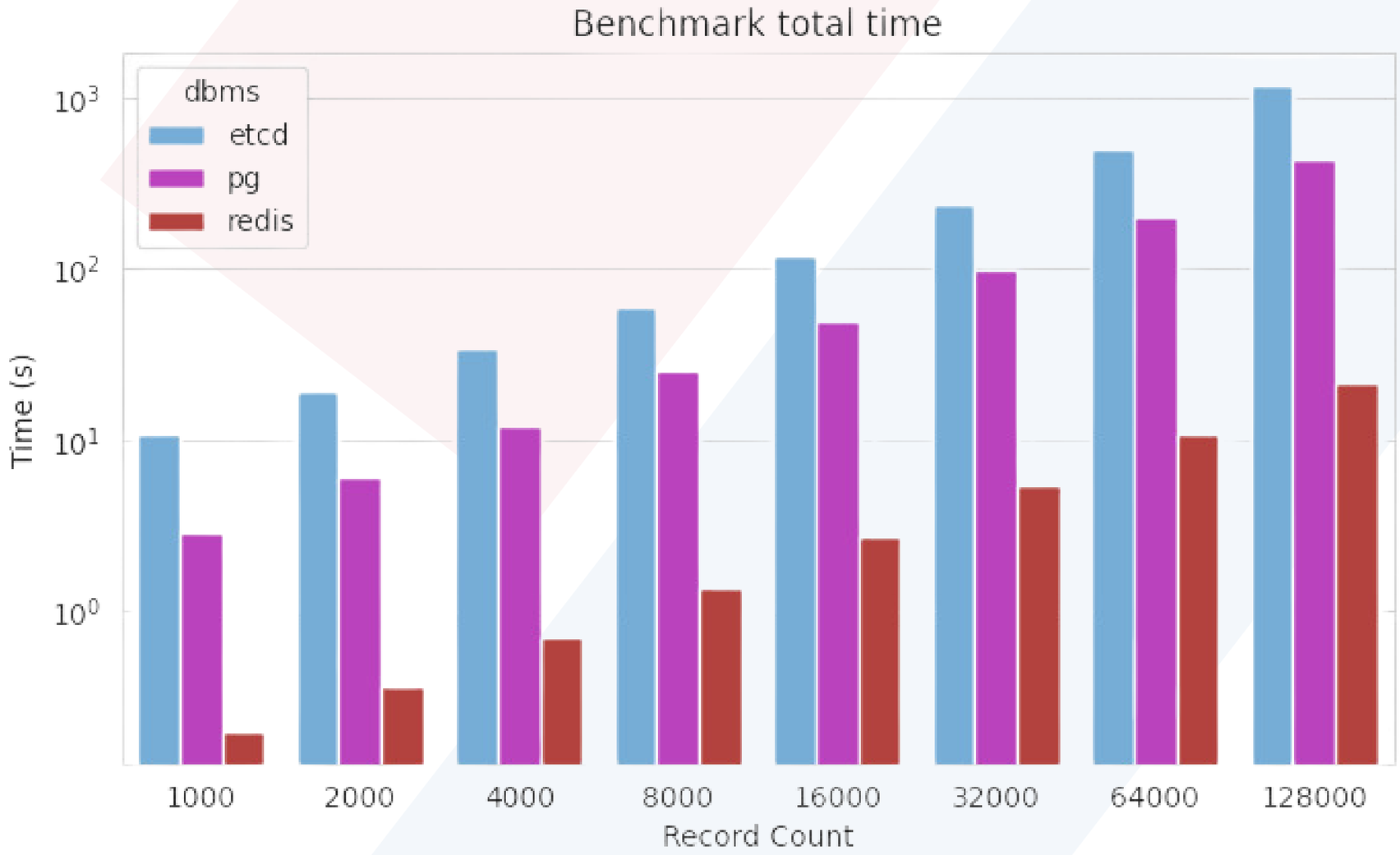
1. Delete the data in the database (if any);
2. Load the database, using workload A’s parameter file;
3. Run workload A;
4. Run workload B;
5. Run workload C;
6. Run workload F;
7. Run workload D (This workload has side-effects on the database);
8. Delete the data in the database;
9. (Re-)Load the database, using workload E’s parameter file;
10. Run workload E (This workload has side-effects on the database).

Methodology

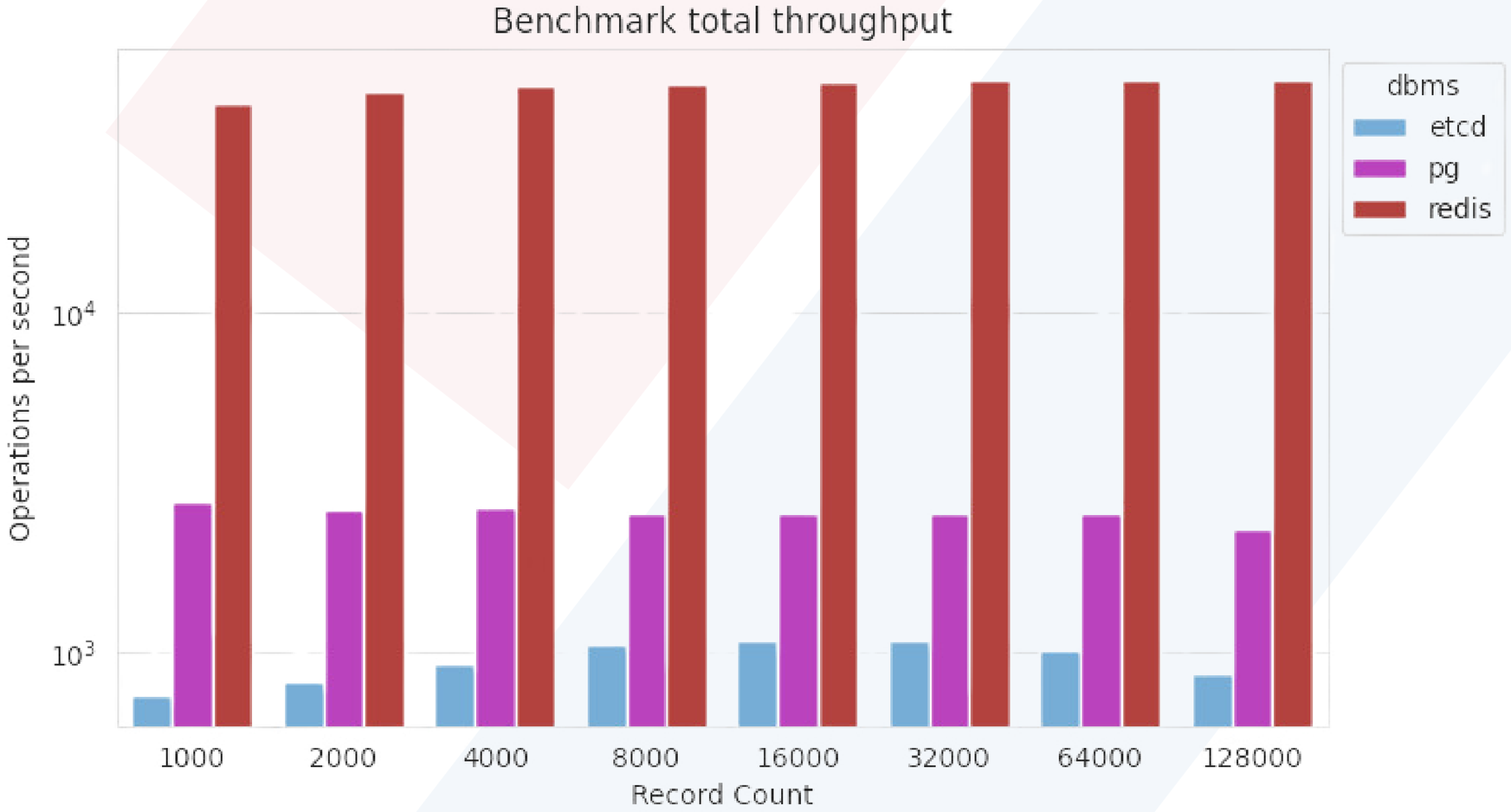
Execution of Benchmark

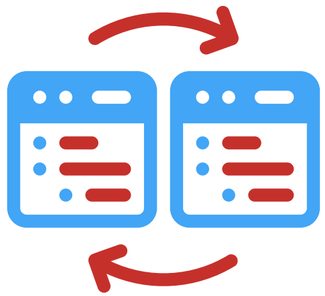
```
set -eu
for rep in { 1 .. 5 } ; do
  echo "Iteration ${rep}"
    for dbms in 'etcd' 'pg' 'redis' ; do
      for records in 1000 2000 4000 8000 16000 32000 64000 128000; do
        operations=$ { records }
       echo -ne 'date +"%Y/%m/%d %H:%M:%S"' "./run_benchmark ${dbms} ${operations} ${records}\n"
    ./run_benchmark.sh ${dbms} ${operations} ${records}
   done
 done
done
```

Methodology



Methodology





Our Sample Console Application

Weather & Quality of Life Explorer

Weather & Quality of Life Explorer

Description of application

Sample application designed to helps users/developers to explore cities around the world.

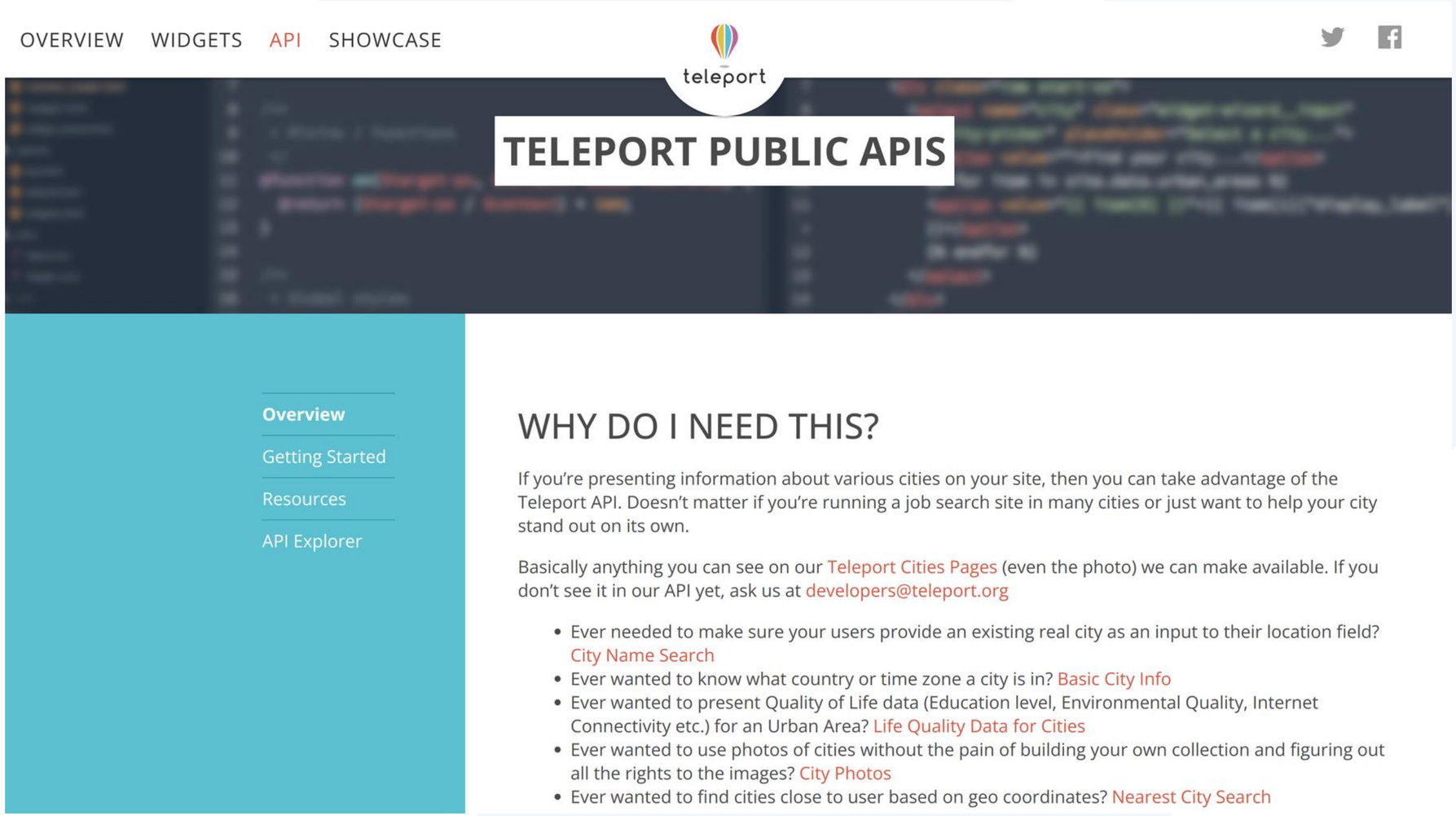
Goal: Improving execution time and performance of retrieving data when user needs to fetch data repetitively in short time.

Leverage ***caching system*** offers by key-value store DBMS

The application fetches data from **Teleport API**.

Weather & Quality of Life Explorer

Teleport API



Weather & Quality of Life Explorer

Teleport API

BASIC INFORMATION ABOUT A CITY

Once you have the city from the previous section, you might want to check what country or timezone it's in or how many people live there. This can easily be accomplished with the following example query for San Francisco:

Following the first linked city URL (https://api.teleport.org/api/cities/geonameid:5391959/) we can see details about San Francisco:


```
{
  "_links": {
    "city:admin1_division": {
      "href": "https://api.teleport.org/api/countries/iso_alpha2:US/admin1_divisions/geonames:CA/"
    },
    "city:country": {
      "href": "https://api.teleport.org/api/countries/iso_alpha2:US/"
    },
    "city:timezone": {
      "href": "https://api.teleport.org/api/timezones/iana:America%2FLos_Angeles/"
    },
    "city:urban_area": {
      "href": "https://api.teleport.org/api/urban_areas/slug:san-francisco-bay-area/"
    },
    "self": {
      "href": "https://api.teleport.org/api/cities/geonameid:5391959/"
    }
  },
  "alternate_names": [
    {
      "name": "Franciscopolis"
    },
    {
      "name": "Frisco"
    },
    ...
  ],
  "geoname_id": 5391959,
  "location": {
    "geohash": "9q8yyk8yuv26emr0cctm",
    "latlon": {
      "latitude": 37.77493,
      "longitude": -122.41942
    }
  },
  "name": "San Francisco",
  "population": 805235
}
```

Weather & Quality of Life Explorer

Redis integration

Fetching and displaying information *quickly and efficiently* = *Redis*

- *Frequently* accessed city information
- *Optimizing* data retrieval
- Enhacing overall system *efficiency*

Weather & Quality of Life Explorer

Redis integration

```python
redis_client = redis.StrictRedis(host='localhost', port=6379, decode_responses=True)
teleport_base_url = 'https://api.teleport.org/api/'

def fetch_city_info(city_name):
    url = f"{teleport_base_url}cities/?search={city_name}"
    response = requests.get(url)
    data = response.json()

    if "_embedded" in data and "city:search-results" in data["_embedded"]:
        search_results = data["_embedded"]["city:search-results"]
        if search_results:
            city_link = search_results[0]["_links"]["city:item"]["href"]
            city_info_response = requests.get(city_link)
            city_info = city_info_response.json()

            # Redis caching for city information
            ret = {
                'id': city_info["geoname_id"],
                'name': city_info["name"],
                'coordinates': city_info["location"]["latlon"],
                'population': city_info["population"]
            }

            redis_client.hset("cities", str.lower(city_name), str(ret))
            redis_client.expire("cities", 60)  # Set expiration time to 60 seconds
            print(f"City information stored in Redis cache for {city_name}")

            return ret
    return
```

Weather & Quality of Life Explorer

Redis integration

```python
def get_cached_data():
    cached_cities = redis_client.hgetall("cities")
    cached_quality_of_life = redis_client.hgetall("quality_of_life")

    if cached_cities or cached_quality_of_life:
        return cached_cities, cached_quality_of_life
    else:
        return None, None
```

Weather & Quality of Life Explorer

Redis usage

Reduced API Calls

Improved responsiveness

Weather & Quality of Life Explorer

Redis usage

```
############################################################
1. Fetch City Information
2. Fetch Quality of Life Scores
3. Show Cached Cities
4. Clear Cache
5. Exit
Enter your choice (1-5): 1

Enter the name of the city: Izmir
City information stored in Redis cache for Izmir
City information retrieved from API:
{'id': 311046, 'name': 'İzmir', 'coordinates': {'latitude': 38.41273, 'longitude': 27.13838}, 'population': 2500603}
Time taken: 0.70233 seconds

############################################################
1. Fetch City Information
2. Fetch Quality of Life Scores
3. Show Cached Cities
4. Clear Cache
5. Exit
Enter your choice (1-5): 1

Enter the name of the city: Izmir
City information retrieved from cache:
{'id': 311046, 'name': 'İzmir', 'coordinates': {'latitude': 38.41273, 'longitude': 27.13838}, 'population': 2500603}
Time taken: 0.00240 seconds

############################################################
1. Fetch City Information
2. Fetch Quality of Life Scores
3. Show Cached Cities
4. Clear Cache
5. Exit
Enter your choice (1-5):
```

Weather & Quality of Life Explorer

Redis usage

Reduced API Calls

Improved responsiveness

Weather & Quality of Life Explorer

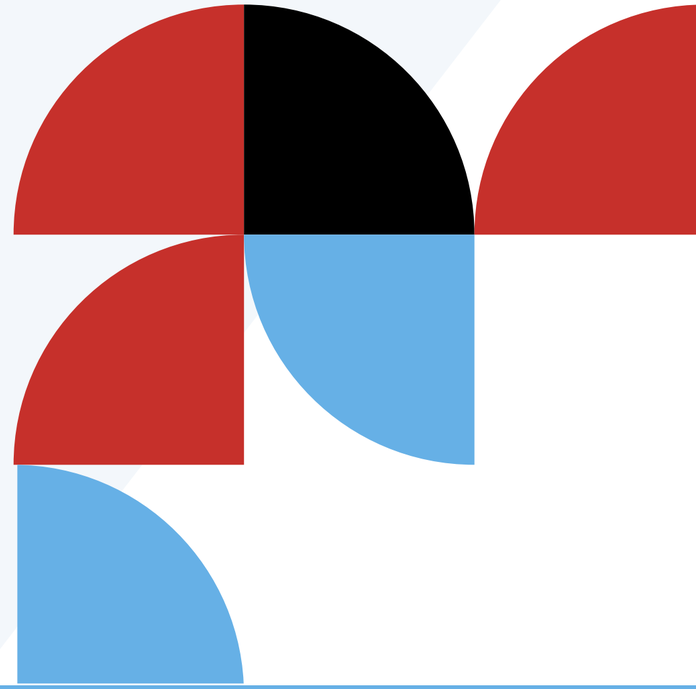
Redis usage

```
Enter the name of the urban area you want to fetch: Brussels
Quality of Life Scores stored in Redis cache for brussels
Quality of Life Scores:
Housing: 6.3549999999999995
Cost of Living: 4.477
Startups: 5.6915000000000004
Venture Capital: 3.3739999999999997
Travel Connectivity: 7.4675
Commute: 5.386750000000001
Business Freedom: 8.876999999999999
Safety: 6.703
Healthcare: 8.816333333333333
Education: 6.6530000000000005
Environmental Quality: 4.591000000000001
Economy: 4.8115000000000006
Taxation: 1.8830000000000002
Internet Access: 7.5715
Leisure & Culture: 8.143999999999998
Tolerance: 6.481
Outdoors: 4.123
Time taken: 0.49997 seconds
```

```
Enter the name of the urban area you want to fetch: Brussels
Quality of Life Scores retrieved from cache for brussels
Quality of Life Scores:
Housing: 6.3549999999999995
Cost of Living: 4.477
Startups: 5.6915000000000004
Venture Capital: 3.3739999999999997
Travel Connectivity: 7.4675
Commute: 5.386750000000001
Business Freedom: 8.876999999999999
Safety: 6.703
Healthcare: 8.816333333333333
Education: 6.6530000000000005
Environmental Quality: 4.591000000000001
Economy: 4.8115000000000006
Taxation: 1.8830000000000002
Internet Access: 7.5715
Leisure & Culture: 8.143999999999998
Tolerance: 6.481
Outdoors: 4.123
Time taken: 0.00093 seconds
```

Conclusion

- Proved Redis’s performance in repetitive jobs using a cache for an API.
- Redis outperformed PostgreSQL and etcd in the YCSB benchmark.
- Redis showed low-latency and efficient caching, recommended for the application.
- Future work could integrate etcd for consistency and transactional capabilities.
- The project aids developers in database selection and contributes to database management systems.



Thank You !

+

Let’s proceed with our short app demo!