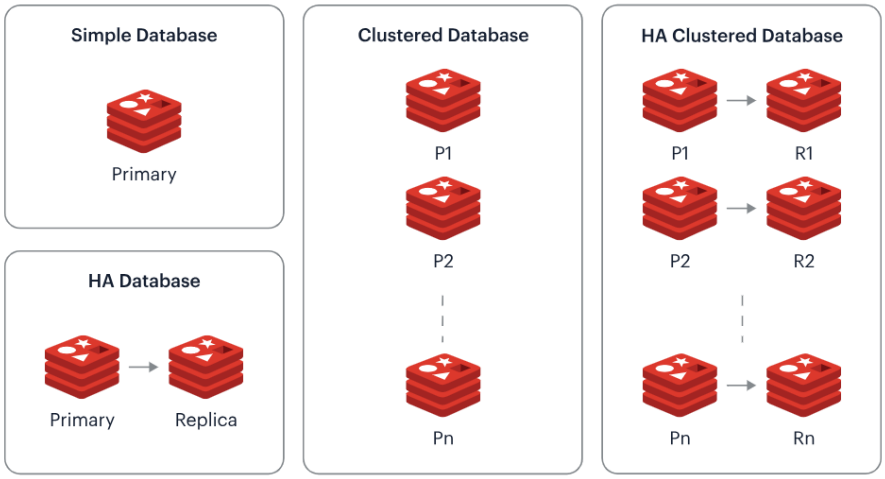
Redis
Esposito Claudia Antonella Erasmus Student,
Software engineer at FILS
Computer Science Engineer in Naples, Federico II

The paper presents a Flask application written in Python to communicate with the Redis server. After this, functions were implemented to test its capabilities in various scenarios. The idea is to perform the same functions with a single server database and with a cluster, and then compare the execution times.



The first step is to initialize a single server from the shell using the command `redis-server`, specifying the port used to interface with clients. In this case the port is 6386.

```
root@claudiaespo:~# redis-server --port 6386
39603:C 27 May 2024 18:04:35.323 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
39603:C 27 May 2024 18:04:35.323 * Redis version=7.2.4, bits=64, commit=00000000, modified=0, pid=39603, just started
39603:C 27 May 2024 18:04:35.323 * Configuration loaded
39603:M 27 May 2024 18:04:35.324 * Increased maximum number of open files to 10032 (it was originally set to 1024).
39603:M 27 May 2024 18:04:35.324 * monotonic clock: POSIX clock_gettime
                _.-``__ ''-._
           _.-``    `.  `_.  ''-._           Redis 7.2.4 (00000000/0) 64 bit
       .-`` .-```.  ```\/    _.,_ ''-._
  (    '      ,       .-`  | `,    )     Running in standalone mode
  |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6386
  |    `-._   `._    /     _.-'    |     PID: 39603
   `-._    `-._  `-./  _.-'    _.-'
  |`-._`-._    `-.__.-'    _.-'_.-'|
  |    `-._`-._        _.-'_.-'    |           https://redis.io
   `-._    `-._`-.__.-'_.-'    _.-'
  |`-._`-._    `-.__.-'    _.-'_.-'|
  |    `-._`-._        _.-'_.-'    |
   `-._    `-._`-.__.-'_.-'    _.-'
       `-._    `-.__.-'    _.-'
           `-._        _.-'
               `-.__.-'

39603:M 27 May 2024 18:04:35.326 * Server initialized
```

In Python, a function was implemented using the `redis-py` library to connect to the Redis server. This allows as client to connect to the Redis server and perform queries.

```
redis_host = 'localhost'
redis_port = 6386
r = redis.Redis(host=redis_host, port=redis_port)
```

First, we move a dataset file into the database.
Using the pandas library, it is possible to access a CSV file and then copy the data from the file into the Redis server database. To save the data, a hash was used, and by using the 'name' as a key, the corresponding data value can be efficiently retrieved from the appropriate slot.

```
df = pd.read_csv("/mnt/c/Users/espoc/Desktop/DSE/phase 3/movies.csv",
encoding="latin1")
for index, row in df.iterrows():
    movie_id = row['name']
    movie_data = row.to_dict()
    for campo, valore in movie_data.items():
        r.hset(f'film:{movie_id}', campo, valore)
```

The chosen CVS files in this project are multiple to test different dataset size, but they all consist of several columns representing different information about the movies (or Video Games) , such as the name, genre, director, etc. Each line corresponds to a different movie. In this case, the key for the hash table is the name of the movie.

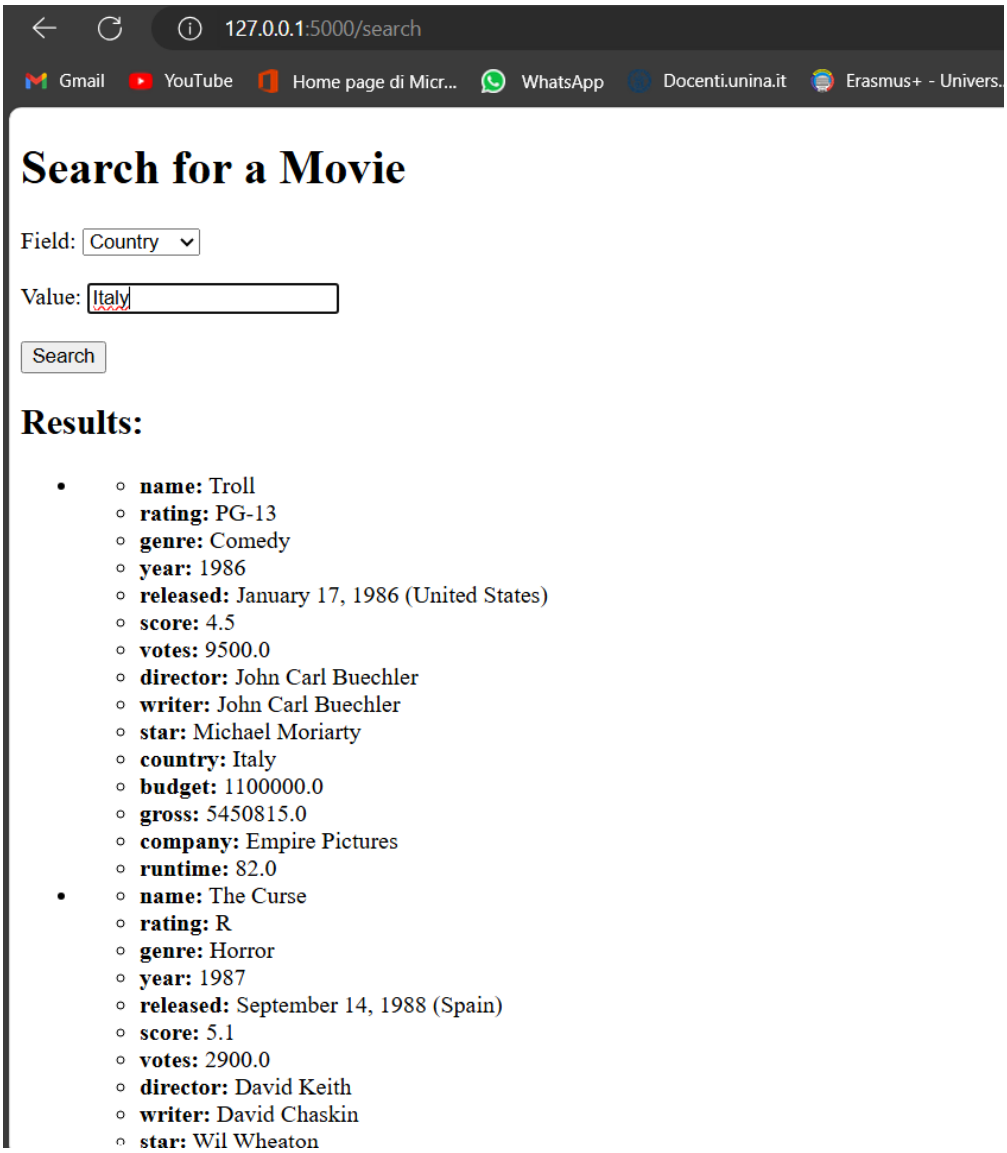
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	name,rating,genre,year,released,score,votes,director,writer,star,country,budget,gross,company,runtime																		
2	The Shining,R,Drama,1980,"June 13, 1980 (United States)",8.4,927000.0,Stanley Kubrick,Stephen King,Jack Nicholson,United Kingdom,19000000.0,46998772.0,Warner Bros.,146.0																		
3	The Blue Lagoon,R,Adventure,1980,"July 2, 1980 (United States)",5.8,65000.0,Randal Kleiser,Henry De Vere Stacpoole,Brooke Shields,United States,4500000.0,58853106.0,Columbia Pictures,104.0																		
4	Star Wars: Episode V - The Empire Strikes Back,PG,Action,1980,"June 20, 1980 (United States)",8.7,1200000.0,Irvin Kershner,Leigh Brackett,Mark Hamill,United States,18000000.0,538375067.0,Lucasfilm,124.0																		
5	Airplane!,PG,Comedy,1980,"July 2, 1980 (United States)",7.7,221000.0,Jim Abrahams,Jim Abrahams,Robert Hays,United States,3500000.0,83453539.0,Paramount Pictures,88.0																		
6	Caddyshack,R,Comedy,1980,"July 25, 1980 (United States)",7.3,108000.0,Harold Ramis,Brian Doyle-Murray,Chevy Chase,United States,6000000.0,39846344.0,Orion Pictures,98.0																		

The code below sets up a web application using Flask. It provides a web interface to interact with redis to search for movies based on various fields (such as name, rating, genre, etc.).
It needs a index.html file as the front-end of the application, providing the HTML form for users to input their search criteria and displaying the search results.

```
from flask import Flask, request, render_template, jsonify
import redis

app = Flask(__name__)

redis_host = 'localhost'
redis_port = 6386
r = redis.Redis(host=redis_host, port=redis_port)

@app.route('/')
def index():
    return render_template('index.html')


@app.route('/search', methods=['POST'])
def search():
    field = request.form.get('field')
    value = request.form.get('value')

    if not field or not value:
        return render_template('index.html', error='Field and value parameters are
required')


    result = []
    keys = r.keys('film:*')

    for key in keys:
        if r.hget(key, field) == value.encode():
            movie_data = r.hgetall(key)
            movie_data = {k.decode(): v.decode() for k, v in movie_data.items()}
            result.append(movie_data)

    if result:
        return render_template('index.html', results=result)
    else:
        return render_template('index.html', message='No results found')
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

The image shows the application implemented in Python, where users can search for a movie in the database by entering a field (such as name, country, ratings, etc.) and a corresponding value. The application returns all matching movies along with their details.



In a separate python project, the capabilities of the server have been tested.
All the function implemented measure execution time in seconds, which will be used to compare results across different scenarios.
The function below, ‘filtering_rating’, is implemented to filter all the data in the database based on the "rating" of the movies.
It takes the Redis connection and `min_rating’ as inputs and returns the number of elements with a rating above the specified threshold. The `filtering_rating` function searches through all the elements in the database using the `redis_connection.keys('*’) ` function, retrieving the "score" for each movie. ‘hget’ retrieves the value associated with the key 'score' within the hash table identified by ‘film_key’. The retrieved value is stored in ‘rating_bytes’ as bytes since Redis stores data in bytes format. The retrieved bytes are decoded in a floating number and if the score is above the specified `min_rating`, the counter is incremented.

```
def filtering_rating(redis_connection, min_rating):
    start_time = time.time()
    high_rating_films = []
    for film_key in redis_connection.keys('*'):
        rating_bytes = redis_connection.hget(film_key, 'score')
        if rating_bytes is not None:
            rating = float(rating_bytes.decode())
            if rating > min_rating:
                high_rating_films.append(film_key.decode())
        else:
            print(f"Warning: No rating found for film {film_key.decode()}")
    end_time = time.time()
    execution_time = end_time - start_time
    print("Films with IMDb rating>=",min_rating," are ", len(high_rating_films))
    print("Execution Time for the function is :", execution_time, "seconds")
    return high_rating_films
```

Another filtering function, `find_movie_by_writer`, has been implemented. This function searches for all films based on the writer of the movies. The implementation is similar to the previously mentioned function, searching through all the keys of the movies and returning the corresponding writer of the film. If it corresponds to the input writer, it returns the movie.

```
def find_movie_by_writer(redis_connection, writer_name):
    print("Reserch of the movies with writer : ", writer_name)
    start_time = time.time()
    movie_keys = []
    writer_name_lower = writer_name.lower()
    for film_key in redis_connection.keys("*"):
        stored_movie_name_bytes = redis_connection.hget(film_key, 'writer')
        if stored_movie_name_bytes is not None:
            stored_movie_name = stored_movie_name_bytes.decode().lower()
            if stored_movie_name == writer_name_lower:
                movie_data = {field.decode(): redis_connection.hget(film_key,
field).decode() for field in
                              redis_connection.hkeys(film_key)}
                movie_keys.append(movie_data)
    end_time = time.time()
    execution_time = end_time - start_time
    if movie_keys:
        print(f"Movie details: {movie_keys}")
        print(f"Execution Time for the function is: {execution_time} seconds")
    else:
        return f"No movies found with writer '{writer_name}'", execution_time
```

The function `find_movie_by_writer_director` returns the movie details by searching for both the director and the writer as input.

```
def find_movie_by_writer_director(redis_connection, director, writer):
    print("Reserch of the movies with writer : ", writer, " and director :", director)
    start_time = time.time()
    movie_keys = []
    split_director = director.lower().split()
    split_writer = writer.lower().split()
    for film_key in redis_connection.keys("*"):
        stored_director_bytes = redis_connection.hget(film_key, "director")
        if stored_director_bytes is not None:
            stored_director = stored_director_bytes.decode().lower()
            split_stored_director = stored_director.split()
            all_director_words_found = all(word in split_stored_director for word in
split_director)
```

```
        else:
            all_director_words_found = False
        stored_writer_bytes = redis_connection.hget(film_key, "writer")
        if stored_writer_bytes is not None:
            stored_writer = stored_writer_bytes.decode().lower()
            split_stored_writer = stored_writer.split()
            all_writer_words_found = all(word in split_stored_writer for word in
split_writer)
        else:
            all_writer_words_found = False
        if all_director_words_found and all_writer_words_found:
            movie_data = {field.decode(): redis_connection.hget(film_key,
field).decode() for field in redis_connection.hkeys(film_key)}
            end_time = time.time()
            execution_time = end_time - start_time
            print(f"Movie details: {movie_data}")
            print(f"Execution Time for the function is: {execution_time} seconds")
            break
    end_time = time.time()
    execution_time = end_time - start_time
    return f"Movie not found with director '{director}' and writer '{writer}'",
execution_time
```

The function `aggregate_high_rating_films` takes a list of movies as input and returns the average rating.

```
def aggregate_filtering(high_rating_films):
    start_time = time.time()

    num_high_rating_films = len(high_rating_films)
    if num_high_rating_films == 0:
        average_rating = 0
    else:
        total_rating = sum(float(r.hget(film_key, 'score').decode()) for film_key in
high_rating_films)
        average_rating = total_rating / num_high_rating_films
    end_time = time.time()
    execution_time = end_time - start_time
    print("Average rating  of the film:", average_rating)
    print("Execution Time for the calculate the average  is :", execution_time,
"seconds")
```

The `calculate_metrics` function provide some detailed insights into the operational status and performance of the Redis server. By returning various server metrics, it helps monitor the overall health and efficiency of the server. For example, it shows the current memory usage, uptime, total commands processed, and number of connected clients, offering valuable information for assessing workload and concurrency handling.

```
def calculate_metrics(redis_connection):
    print("Metrics calculator about the server")
    info_data = redis_connection.info()
    used_memory = info_data['used_memory']
    uptime_in_seconds = info_data['uptime_in_seconds']
    total_commands_processed = info_data['total_commands_processed']
    connected_clients = info_data['connected_clients']
    print(f"Used Memory: {used_memory} bytes")
    print(f"Uptime: {uptime_in_seconds} seconds")
    print(f"Total Commands Processed: {total_commands_processed}")
    print(f"Connected Clients: {connected_clients}")
    used_memory_bytes = info_data['used_memory']
```

```
    total_system_memory_bytes = info_data['total_system_memory']
    memory_usage_percent = (used_memory_bytes / total_system_memory_bytes) * 100
    print(f"Memory Usage Percentage: {memory_usage_percent:.2f}%")
```

Using Threads, multiple requests have been simulated simultaneously to the server. Each thread takes as input the function to be executed (in this case, the filtering function for movie ratings), the client ID (ranging from 1 to 10), and the rating value. Threads effectively simulate multiple requests occurring at the same time. Testing the server with multiple concurrent requests is important because it helps ensure the server can handle real-world scenarios. This type of testing, known as stress testing, pushes the system beyond its normal operational capacity to observe how it behaves under extreme conditions.

```
num_clients = 10
print("simulation with multiple clients")
min_ratings = [8.0, 4.0, 6.0]
threads = []
for i in range(num_clients):
    client_id = i + 1
    min_rating = min_ratings[i % len(min_ratings)]  # round-robin

    thread = threading.Thread(target=client_task1, args=(client_id, min_rating))
    thread.start()
    threads.append(thread)

for thread in threads:
    thread.join()

def client_task1(client_id, min_rating):
    start_time = time.time()
    with print_lock:
        print(f"Client {client_id}: ")
        filtering_rating(r, min_rating)
    end_time = time.time()
    execution_time = end_time - start_time
    with print_lock:
        print(f"Client {client_id} completed in {execution_time:.2f} seconds")
```

As follow it is possible to see the results of different scenario:

- Single server with a dataset of 1646 elements

```
New database, Number of elements in the database: 1646
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.19573426246643066 seconds
Average rating  of the film: 8.255882352941176
Execution Time for the calculate the average  is : 0.003679990768432617 seconds
Films with IMDb rating>= 4.0  are  1603
Execution Time for the function is : 0.20404815673828125 seconds
Average rating  of the film: 6.29519650655021
Execution Time for the calculate the average  is : 0.20116925239562988 seconds
Films with IMDb rating>= 6.0  are  977
Execution Time for the function is : 0.1890878677368164 seconds
Average rating  of the film: 6.87697031729785
Execution Time for the calculate the average  is : 0.11507701873779297 seconds
Reserch of the movies with writer :  Stephen King
Movie details: [{'name': 'The Running Man', 'rating': 'R', 'genre': 'Action', 'year':
'1987', 'released': 'November 13, 1987 (United States)', 'score': '6.7', 'votes':
'144000.0', 'director': 'Paul Michael Glaser', 'writer': 'Stephen King', 'star':
'Arnold Schwarzenegger', 'country': 'United States', 'budget': '27000000.0', 'gross':
'38122105.0', 'company': 'TAFT Entertainment Pictures', 'runtime': '101.0'}, ..]
```

```
Execution Time for the function is: 0.2078564167022705 seconds
Research of the movies with writer :  Dan Aykroyd
Movie details: [{'name': 'Ghostbusters', 'rating': 'PG', 'genre': 'Action', 'year':
'1984', 'released': 'June 8, 1984 (United States)', 'score': '7.8', 'votes':
'365000.0', 'director': 'Ivan Reitman', 'writer': 'Dan Aykroyd', 'star': 'Bill Murray',
'country': 'United States', 'budget': '30000000.0', 'gross': '296187079.0', 'company':
'Columbia Pictures', 'runtime': '105.0'}, ..]
Execution Time for the function is: 0.19800400733947754 seconds
Research of the movies with writer :  Stephen King  and director : Stanley Kubrick
Movie details: {'name': 'The Shining', 'rating': 'R', 'genre': 'Drama', 'year': '1980',
'released': 'June 13, 1980 (United States)', 'score': '8.4', 'votes': '927000.0',
'director': 'Stanley Kubrick', 'writer': 'Stephen King', 'star': 'Jack Nicholson',
'country': 'United Kingdom', 'budget': '19000000.0', 'gross': '46998772.0', 'company':
'Warner Bros.', 'runtime': '146.0'}
Execution Time for the function is: 0.8950650691986084 seconds
```

It is possible to notice that the search for movies with 2 inputs takes longer than with 1 input. This is because the filtering process is more complex, requiring the system to check every movie for both conditions, which takes more processing time.

- Single server with a dataset of 1646 elements, simulation with 10 simultaneous clients for the function of filtering

```
simulation with multiple clients
Client 1:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.20052838325500488 seconds
Client 1 completed in 0.20 seconds
Client 3:
Films with IMDb rating>= 6.0  are  977
Execution Time for the function is : 0.5098516941070557 seconds
Client 3 completed in 0.71 seconds
Client 4:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.3575117588043213 seconds
Client 4 completed in 1.07 seconds
Client 6:
Films with IMDb rating>= 6.0  are  977
Execution Time for the function is : 0.1883533000946045 seconds
Client 6 completed in 1.25 seconds
Client 9:
Films with IMDb rating>= 6.0  are  977
Execution Time for the function is : 0.5207197666168213 seconds
Client 9 completed in 1.77 seconds
Client 2:
Films with IMDb rating>= 4.0  are  1603
Execution Time for the function is : 0.3333714008331299 seconds
Client 2 completed in 2.11 seconds
Client 5:
Films with IMDb rating>= 4.0  are  1603
Execution Time for the function is : 0.18558526039123535 seconds
Client 5 completed in 2.29 seconds
Client 10:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.4704890251159668 seconds
Client 10 completed in 2.76 seconds
Client 7:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.3296496868133545 seconds
Client 7 completed in 3.09 seconds
```

```
Client 8:
Films with IMDb rating>= 4.0  are  1603
Execution Time for the function is : 0.18586182594299316 seconds
Client 8 completed in 3.28 seconds
```

For each client, we measured both the execution time of the filtering operation itself and the overall thread execution time**,** covering the entire process from the initiation of the client call to the final printing.
When clients make simultaneous requests, execution times tend to increase compared to a single client request. This is expected because server resources (CPU, memory, I/O) are shared among multiple concurrent requests, leading to longer processing times.
The variability in execution times among different clients in this result is significant. Some clients complete their operations quickly (for example Client 1 in 0.20 seconds), while others take much longer (e.g., Client 8 in 3.28 seconds). This is due to the amount of data filtered or resource contention on the server.
With more simultaneous clients, there is increased contention for server resources. This can cause delays in processing requests as resources are shared among all active threads.

- Single server with a dataset of 7512 elements

```
New database, Number of elements in the database: 7512
Films with IMDb rating>= 8.0  are  191
Execution Time for the function is : 1.17757248878479 seconds
Average rating  of the film: 8.280104712041881
Execution Time for the calculate the average  is : 0.030361175537109375 seconds
Films with IMDb rating>= 4.0  are  7366
Execution Time for the function is : 1.553006887435913 seconds
Average rating  of the film: 6.448859625305445
Execution Time for the calculate the average  is : 0.9028570652008057 seconds
Films with IMDb rating>= 6.0  are  5053
Execution Time for the function is : 1.2213072776794434 seconds
Average rating  of the film: 6.9167227389669295
Execution Time for the calculate the average  is : 0.5752980709075928 seconds
Reserch of the movies with writer :  Stephen King
Movie details: [{'name': 'The Running Man', 'rating': 'R', 'genre': 'Action', 'year':
'1987', 'released': 'November 13, 1987 (United States)', 'score': '6.7', 'votes':
'144000.0', 'director': 'Paul Michael Glaser', 'writer': 'Stephen King', 'star':
'Arnold Schwarzenegger', 'country': 'United States', 'budget': '27000000.0', 'gross':
'38122105.0', 'company': 'TAFT Entertainment Pictures', 'runtime': '101.0'}, ..]
Execution Time for the function is: 1.4185049533843994 seconds
Reserch of the movies with writer :  Dan Aykroyd
Movie details: [{'name': 'The Blues Brothers', 'rating': 'R', 'genre': 'Action',
'year': '1980', 'released': 'June 20, 1980 (United States)', 'score': '7.9', 'votes':
'188000.0', 'director': 'John Landis', 'writer': 'Dan Aykroyd', 'star': 'John Belushi',
'country': 'United States', 'budget': '27000000.0', 'gross': '115229890.0', 'company':
'Universal Pictures', 'runtime': '133.0'}, ..]
Execution Time for the function is: 1.591078758239746 seconds
Reserch of the movies with writer :  Stephen King  and director : Stanley Kubrick
Movie details: {'name': 'The Shining', 'rating': 'R', 'genre': 'Drama', 'year': '1980',
'released': 'June 13, 1980 (United States)', 'score': '8.4', 'votes': '927000.0',
'director': 'Stanley Kubrick', 'writer': 'Stephen King', 'star': 'Jack Nicholson',
'country': 'United Kingdom', 'budget': '19000000.0', 'gross': '46998772.0', 'company':
'Warner Bros.', 'runtime': '146.0'}
Execution Time for the function is: 2.1094272136688232 seconds
Metrics calculator about the server
Used Memory: 4406912 bytes
Uptime: 3258 seconds
Total Commands Processed: 12224596
```

```
Connected Clients: 4
Memory Usage Percentage: 0.05%
```

- Single server with a dataset of 7512 elements, simulation with 10 simultaneous clients for the function of filtering

```
Client 1:
Films with IMDb rating>= 8.0  are  191
Execution Time for the function is : 1.3703389167785645 seconds
Client 1 completed in 1.37 seconds
Client 3:
Films with IMDb rating>= 6.0  are  5053
Execution Time for the function is : 1.2258179187774658 seconds
Client 3 completed in 2.60 seconds
Client 5:
Films with IMDb rating>= 4.0  are  7366
Execution Time for the function is : 1.3011326789855957 seconds
Client 5 completed in 3.90 seconds
Client 6:
Films with IMDb rating>= 6.0  are  5053
Execution Time for the function is : 1.3618812561035156 seconds
Client 6 completed in 5.26 seconds
Client 9:
Films with IMDb rating>= 6.0  are  5053
Execution Time for the function is : 1.3253254890441895 seconds
Client 9 completed in 6.57 seconds
Client 2:
Films with IMDb rating>= 4.0  are  7366
Execution Time for the function is : 1.5138161182403564 seconds
Client 2 completed in 8.10 seconds
Client 7:
Films with IMDb rating>= 8.0  are  191
Execution Time for the function is : 1.3659250736236572 seconds
Client 7 completed in 9.46 seconds
Client 8:
Films with IMDb rating>= 4.0  are  7366
Execution Time for the function is : 1.3517799377441406 seconds
Client 8 completed in 10.81 seconds
Client 10:
Films with IMDb rating>= 8.0  are  191
Execution Time for the function is : 1.4733483791351318 seconds
Client 10 completed in 12.28 seconds
Client 4:
Films with IMDb rating>= 8.0  are  191
Execution Time for the function is : 1.8547186851501465 seconds
Client 4 completed in 14.15 seconds
```

- Single server with a dataset of 11563 elements

```
New Database, Number of elements in the database: 11563
Films with IMDb rating>= 80.0  are  928
Execution Time for the function is : 1.4197728633880615 seconds
total number of elements with ratings >= 8.0: 928
Average rating  of fil with rate>= 8: 85.62068965517241
Execution Time for the function is : 0.14196109771728516 seconds
Films with IMDb rating>= 40.0  are  4511
Execution Time for the function is : 1.4318649768829346 seconds
total number of elements with ratings >= 8.0: 4511
Average rating  of fil with rate>= 8: 69.75105298160054
```

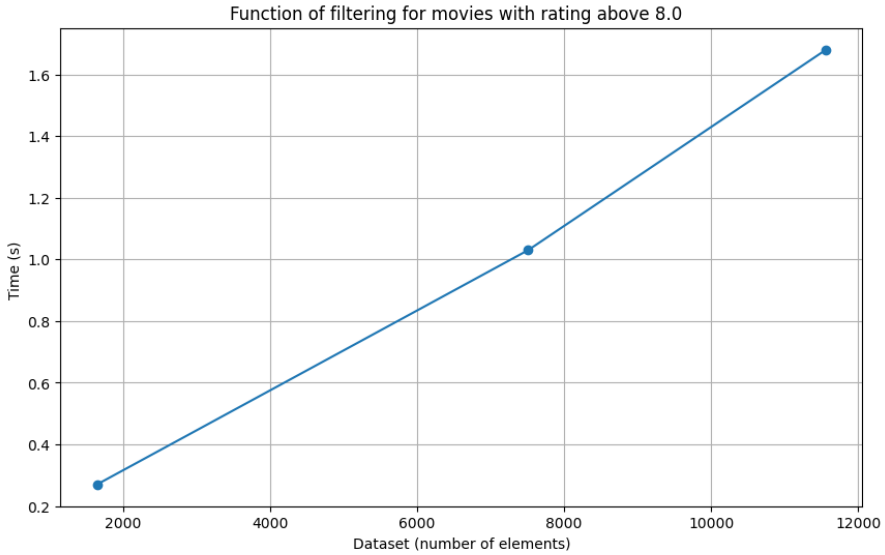
```
Execution Time for the function is : 0.6050026416778564 seconds
Films with IMDb rating>= 60.0  are  3461
Execution Time for the function is : 3.6697816848754883 seconds
total number of elements with ratings >= 8.0: 3461
Average rating  of fil with rate>= 8: 74.84830973707021
Execution Time for the function is : 0.7498395442962646 seconds
Movie with Publisher:Nintendo, are 658
Time taken for search: 3.9017398357391357 seconds
Movie details: {'Name': 'Pokemon Dash', 'Platform': 'DS', 'Year_of_Release': '2004.0',
'Genre': 'Racing', 'Publisher': 'Nintendo', 'NA_Sales': '0.21', 'EU_Sales': '0.14',
'JP_Sales': '0.38', 'Other_Sales': '0.04', 'Global_Sales': '0.77', 'Critic_Score':
'46.0', 'Critic_Count': '28.0', 'User_Score': '5.9', 'User_Count': '34.0', 'Developer':
'Ambrella', 'Rating': 'E'}
Time taken for search: 0.21025896072387695 seconds
Metrics calculator about the server
Used Memory: 6368648 bytes
Uptime: 3332 seconds
Total Commands Processed: 12634417
Connected Clients: 4
Memory Usage Percentage: 0.08%
```

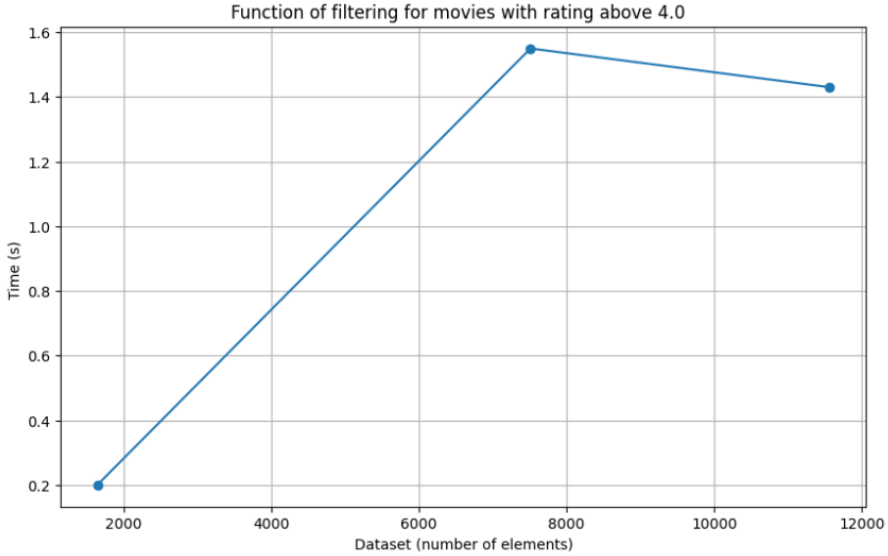
- Single server with a dataset of 11563 elements, simulation with 10 simultaneous clients for the function of filtering

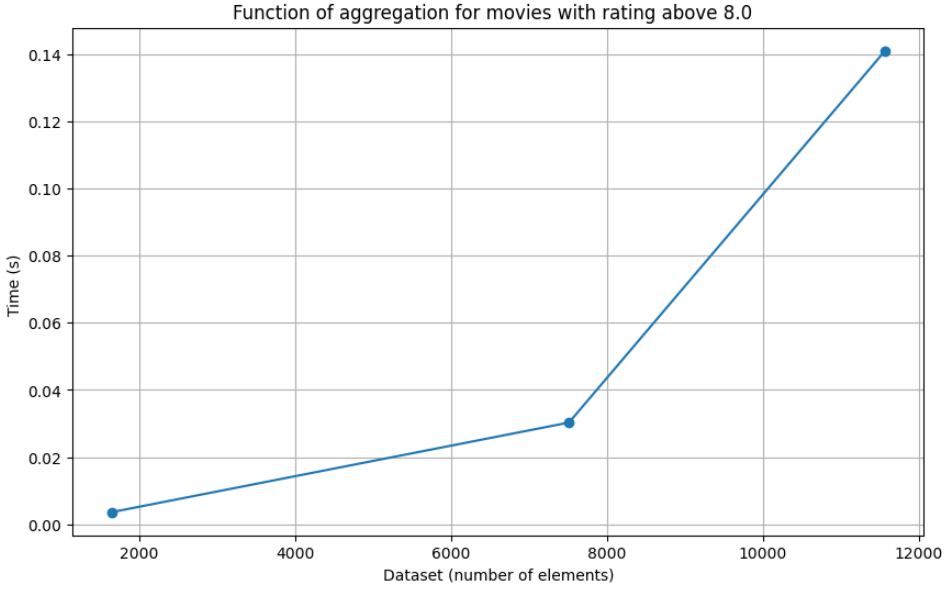
```
Client 1:
Films with IMDb rating>= 80.0  are  928
Execution Time for the function is : 1.8709793090820312 seconds
Client 1 completed in 1.87 seconds
Client 3:
Films with IMDb rating>= 60.0  are  3461
Execution Time for the function is : 1.7970197200775146 seconds
Client 3 completed in 3.67 seconds
Client 5:
Films with IMDb rating>= 40.0  are  4511
Execution Time for the function is : 1.957707166671753 seconds
Client 5 completed in 5.63 seconds
Client 7:
Films with IMDb rating>= 80.0  are  928
Execution Time for the function is : 1.8095133304595947 seconds
Client 7 completed in 7.43 seconds
Client 9:
Films with IMDb rating>= 60.0  are  3461
Execution Time for the function is : 2.36116623878479 seconds
Client 9 completed in 9.80 seconds
Client 2:
Films with IMDb rating>= 40.0  are  4511
Execution Time for the function is : 2.0684080123901367 seconds
Client 2 completed in 11.87 seconds
Client 6:
Films with IMDb rating>= 60.0  are  3461
Execution Time for the function is : 1.5319108963012695 seconds
Client 6 completed in 13.40 seconds
Client 10:
Films with IMDb rating>= 80.0  are  928
Execution Time for the function is : 1.571002721786499 seconds
Client 10 completed in 14.97 seconds
Client 8:
Films with IMDb rating>= 40.0  are  4511
Execution Time for the function is : 1.8768129348754883 seconds
Client 8 completed in 16.85 seconds
Client 4:
Films with IMDb rating>= 80.0  are  928
```

```
Execution Time for the function is : 1.9830679893493652 seconds
Client 4 completed in 18.83 seconds
```

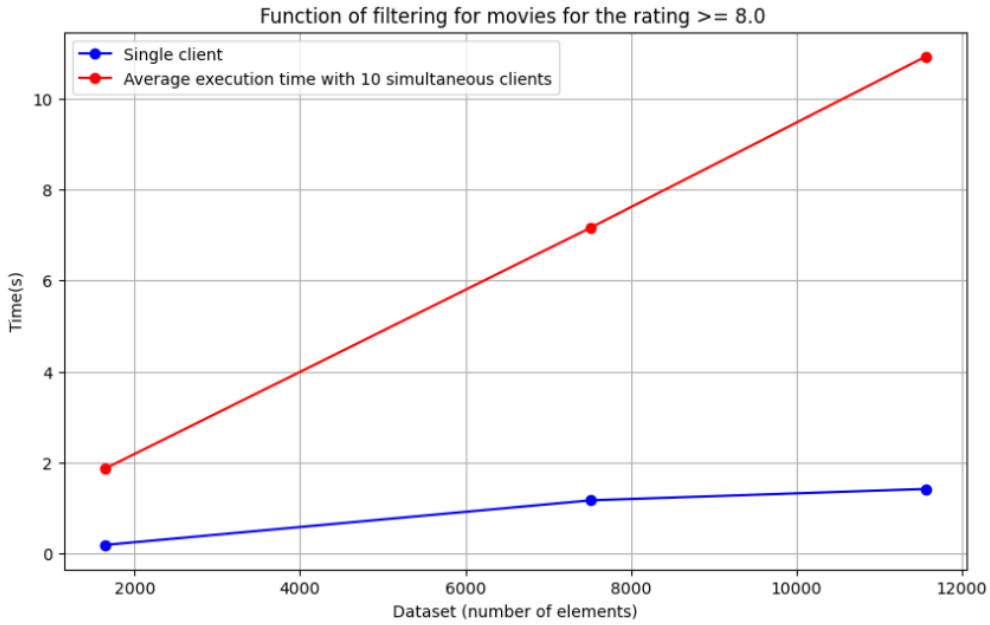
Below are 3 graphs showing the execution time for different functions (both filtering and aggregation) on a single server with varying dataset sizes. As shown in the plots, the relationship between dataset size (x-axis) and execution time of the function (y-axis) appears to be linear. This indicates that the filtering and aggregation time increases proportionally with the number of records processed.



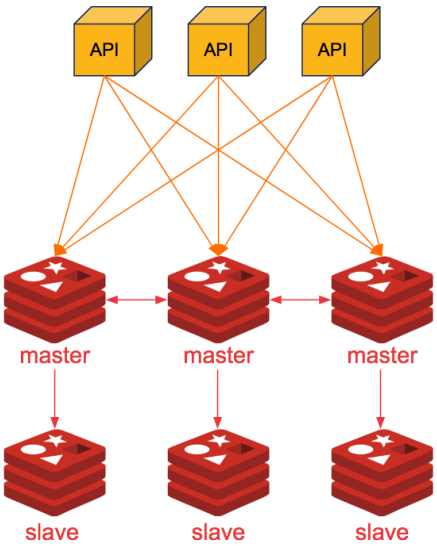




The image below shows a line graph comparing the performance of a single-client request and multiple concurrent client requests on a single server database for a filtering function (e.g., filtering_rating) that returns all movies with a rating above a specific threshold. The x-axis represents the size of the dataset, while the y-axis represents the query execution time. The blue line represents the single-client query, while the red line represents the average response time for all 10 clients running the same queries concurrently. The graph demonstrates that the single-client query is consistently faster than the multiple-client query, regardless of the dataset size. This is because the database server can dedicate all its resources to processing the single query.



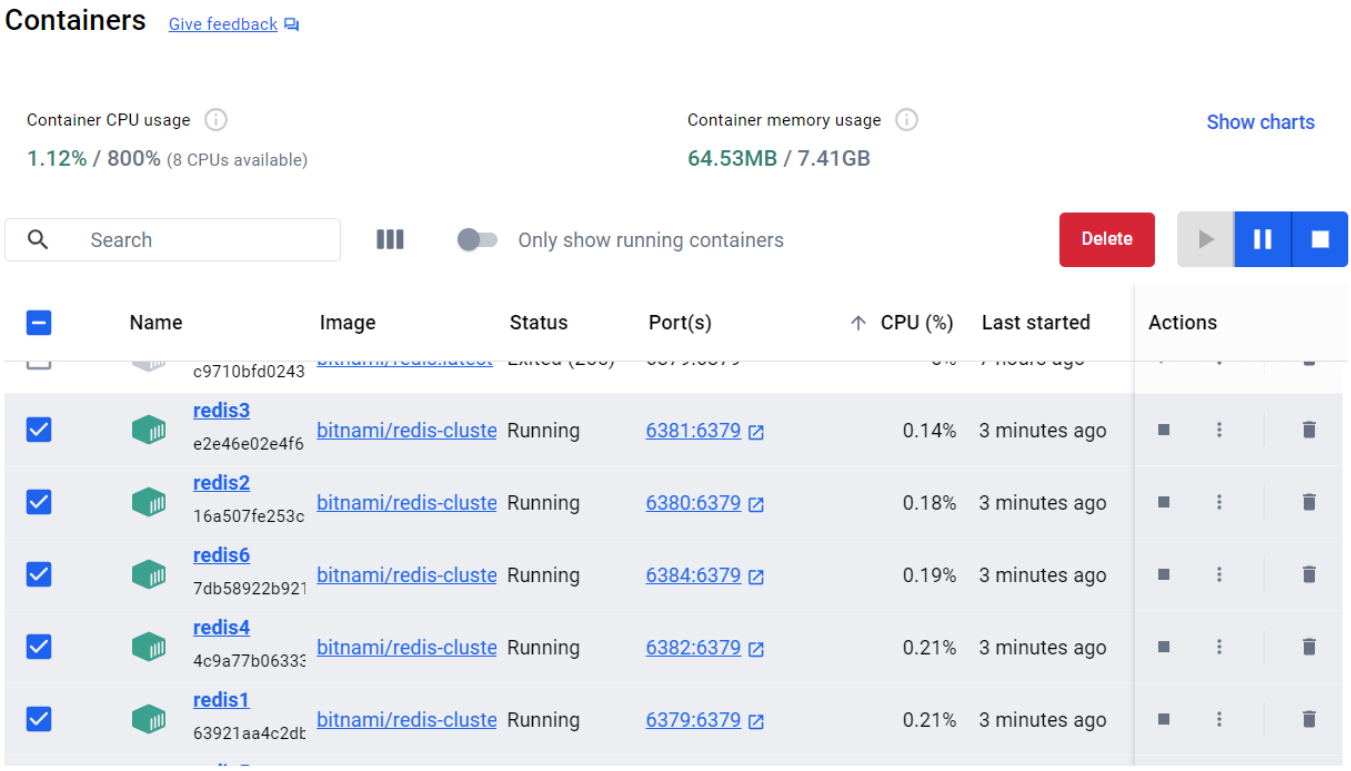
The same simulation was performed on a cluster to test the scalability. The cluster is composed by 6 nodes, by 3 masters and 3 slaves. Each master node serves as a primary point of access for data read and write operations, while the slave nodes replicate data from the master nodes to ensure data redundancy and availability. This setup enables parallel processing, effectively distributing the workload. Additionally, in the event of a master node failure, the slave nodes can seamlessly take over, ensuring uninterrupted service and minimizing downtime. Through this distributed architecture, the cluster achieves high availability and scalability, making it well-suited for handling large-scale data processing and high-throughput applications.



This code establishes a connection to a Redis cluster using the RedisCluster library. The variable startup_nodes contain a list of dictionaries, each representing a node within the cluster. Each dictionary has two keys: "host" and "port," indicating the IP address and port on which the Redis service is listening for each node. The option decode_responses indicates that responses received from the cluster will be automatically decoded into Unicode strings instead of being returned as bytes.

```
startup_nodes = [
    {"host": "redis1", "port": "6379"},
    {"host": "redis2", "port": "6379"},
    {"host": "redis3", "port": "6379"},
    {"host": "redis4", "port": "6379"},
    {"host": "redis5", "port": "6379"},
    {"host": "redis6", "port": "6379"}
]

rc = RedisCluster(startup_nodes=startup_nodes, decode_responses=True)
```

The Redis cluster nodes are being simulated using Docker containers. Docker is a platform that allows developers to package and distribute applications, along with all their dependencies, as lightweight containers.
Each node in the Redis cluster is instantiated as a separate Docker container. These containers encapsulate the Redis server along with its configuration and data, effectively simulating individual instances of Redis within a distributed environment. Each container runs a separate Redis server process, and these containers can communicate with each other to form a clustered Redis setup. By using Docker containers to represent individual nodes within a Redis cluster, it is possible simulate the scalability of the cluster.



The functions implemented for the cluster are the same seen before for the single server.

- Cluster with a dataset of 1646 elements

```
Number of elements in the database: {'172.21.0.6:6379': 553, '172.21.0.5:6379': 553,
'172.21.0.3:6379': 549, '172.21.0.4:6379': 549, '172.21.0.2:6379': 544,
'172.21.0.7:6379': 544}
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.22724556922912598 seconds
Average rating  of the film: 8.255882352941176
Execution Time for the calculate the average  is : 0.00460362434387207 seconds
Films with IMDb rating>= 4.0  are  1603
Execution Time for the function is : 0.22927141189575195 seconds
Average rating  of the film: 6.295196506550214
Execution Time for the calculate the average  is : 0.2266545295715332 seconds
Films with IMDb rating>= 6.0  are  977
Execution Time for the function is : 0.27622509002685547 seconds
Average rating  of the film: 6.876970317297851
Execution Time for the calculate the average  is : 0.2589290142059326 seconds
Research of the movies with writer :  Stephen King
Movie details: [{'name': 'The Running Man', 'rating': 'R', 'genre': 'Action', 'year':
'1987', 'released': 'November 13, 1987 (United States)', 'score': '6.7', 'votes':
'144000.0', 'director': 'Paul Michael Glaser', 'writer': 'Stephen King', 'star':
'Arnold Schwarzenegger', 'country': 'United States', 'budget': '27000000.0', 'gross':
'38122105.0', 'company': 'TAFT Entertainment Pictures', 'runtime': '101.0'}, ..]
Execution Time for the function is: 0.6027004718780518 seconds
Research of the movies with writer :  Dan Aykroyd
Movie details: [{'name': 'The Blues Brothers', 'rating': 'R', 'genre': 'Action',
'year': '1980', 'released': 'June 20, 1980 (United States)', 'score': '7.9', 'votes':
'188000.0', 'director': 'John Landis', 'writer': 'Dan Aykroyd', 'star': 'John Belushi',
'country': 'United States', 'budget': '27000000.0', 'gross': '115229890.0', 'company':
'Universal Pictures', 'runtime': '133.0'}, ..]
Execution Time for the function is: 0.6095025539398193 seconds
Research of the movies with writer :  Stephen King  and director : Stanley Kubrick
Movie details: {'name': 'The Shining', 'rating': 'R', 'genre': 'Drama', 'year': '1980',
'released': 'June 13, 1980 (United States)', 'score': '8.4', 'votes': '927000.0',
'director': 'Stanley Kubrick', 'writer': 'Stephen King', 'star': 'Jack Nicholson',
'country': 'United Kingdom', 'budget': '19000000.0', 'gross': '46998772.0', 'company':
'Warner Bros.', 'runtime': '146.0'}
Execution Time for the function is: 0.12003397941589355 seconds
```

- Cluster server with a dataset of 1646 elements, simulation with 10 simultaneous clients for the function of filtering

```
Client 1:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.23886752128601074 seconds
Client 1 completed in 0.24 seconds
Client 2:
Films with IMDb rating>= 4.0  are  1603
Execution Time for the function is : 0.22754192352294922 seconds
Client 2 completed in 0.47 seconds
Client 4:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.23993825912475586 seconds
Client 4 completed in 0.70 seconds
Client 6:
Films with IMDb rating>= 6.0  are  977
Execution Time for the function is : 0.2402641773223877 seconds
Client 6 completed in 0.94 seconds
Client 8:
Films with IMDb rating>= 4.0  are  1603
```

```
Execution Time for the function is : 0.5629396438598633 seconds
Client 8 completed in 1.50 seconds
Client 3:
Films with IMDb rating>= 6.0  are  977
Execution Time for the function is : 0.729588508605957 seconds
Client 5:
Films with IMDb rating>= 4.0  are  1603
Execution Time for the function is : 0.6369900703430176 seconds
Client 5 completed in 2.87 seconds
Client 7:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.5193264484405518 seconds
Client 7 completed in 3.39 seconds
Client 3 completed in 2.24 seconds
Client 10:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.5413229465484619 seconds
Client 10 completed in 3.93 seconds
Client 9:
Films with IMDb rating>= 6.0  are  977
Execution Time for the function is : 0.5086469650268555 seconds
Client 9 completed in 4.44 seconds
```

- Cluster with a dataset of 7512 elements

```
New database, Number of elements in the database: {'172.21.0.6:6379': 2472,
'172.21.0.5:6379': 2472, '172.21.0.3:6379': 2530, '172.21.0.4:6379': 2530,
'172.21.0.2:6379': 2510, '172.21.0.7:6379': 2510}
Films with IMDb rating>= 8.0  are  191
Execution Time for the function is : 2.027883529663086 seconds
Average rating  of the film: 8.280104712041881
Execution Time for the calculate the average  is : 0.028390169143676758 seconds
Films with IMDb rating>= 4.0  are  7366
Execution Time for the function is : 1.9673349857330322 seconds
Average rating  of the film: 6.448859625305447
Execution Time for the calculate the average  is : 1.9122040271759033 seconds
Films with IMDb rating>= 6.0  are  5053
Execution Time for the function is : 1.4387836456298828 seconds
Average rating  of the film: 6.916722738966924
Execution Time for the calculate the average  is : 0.8245503902435303 seconds
Reserch of the movies with writer :  Stephen King
Movie details: [{'name': 'The Running Man', 'rating': 'R', 'genre': 'Action', 'year':
'1987', 'released': 'November 13, 1987 (United States)', 'score': '6.7', 'votes':
'144000.0', 'director': 'Paul Michael Glaser', 'writer': 'Stephen King', 'star':
'Arnold Schwarzenegger', 'country': 'United States', 'budget': '27000000.0', 'gross':
'38122105.0', 'company': 'TAFT Entertainment Pictures', 'runtime': '101.0'}, ..]
Execution Time for the function is: 2.5347414016723633 seconds
Reserch of the movies with writer :  Dan Aykroyd
Movie details: [{'name': 'The Blues Brothers', 'rating': 'R', 'genre': 'Action',
'year': '1980', 'released': 'June 20, 1980 (United States)', 'score': '7.9', 'votes':
'188000.0', 'director': 'John Landis', 'writer': 'Dan Aykroyd', 'star': 'John Belushi',
'country': 'United States', 'budget': '27000000.0', 'gross': '115229890.0', 'company':
'Universal Pictures', 'runtime': '133.0'}, ..]
Execution Time for the function is: 2.2847695350646973 seconds
Reserch of the movies with writer :  Stephen King  and director : Stanley Kubrick
Movie details: {'name': 'The Shining', 'rating': 'R', 'genre': 'Drama', 'year': '1980',
'released': 'June 13, 1980 (United States)', 'score': '8.4', 'votes': '927000.0',
'director': 'Stanley Kubrick', 'writer': 'Stephen King', 'star': 'Jack Nicholson',
'country': 'United Kingdom', 'budget': '19000000.0', 'gross': '46998772.0', 'company':
'Warner Bros.', 'runtime': '146.0'}
Execution Time for the function is: 2.3597283363342285 seconds
```

- Cluster server with a dataset of 7512 elements, simulation with 10 simultaneous clients for the function of filtering

```
Client 1:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.23886752128601074 seconds
Client 1 completed in 0.24 seconds
Client 2:
Films with IMDb rating>= 4.0  are  1603
Execution Time for the function is : 0.22754192352294922 seconds
Client 2 completed in 0.47 seconds
Client 4:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.23993825912475586 seconds
Client 4 completed in 0.70 seconds
Client 6:
Films with IMDb rating>= 6.0  are  977
Execution Time for the function is : 0.2402641773223877 seconds
Client 6 completed in 0.94 seconds
Client 8:
Films with IMDb rating>= 4.0  are  1603
Execution Time for the function is : 0.5629396438598633 seconds
Client 8 completed in 1.50 seconds
Client 3:
Films with IMDb rating>= 6.0  are  977
Execution Time for the function is : 0.729588508605957 seconds
Client 5:
Films with IMDb rating>= 4.0  are  1603
Execution Time for the function is : 0.6369900703430176 seconds
Client 5 completed in 2.87 seconds
Client 7:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.5193264484405518 seconds
Client 7 completed in 3.39 seconds
Client 3 completed in 2.24 seconds
Client 10:
Films with IMDb rating>= 8.0  are  34
Execution Time for the function is : 0.5413229465484619 seconds
Client 10 completed in 3.93 seconds
Client 9:
Films with IMDb rating>= 6.0  are  977
Execution Time for the function is : 0.5086469650268555 seconds
Client 9 completed in 4.44 seconds
```

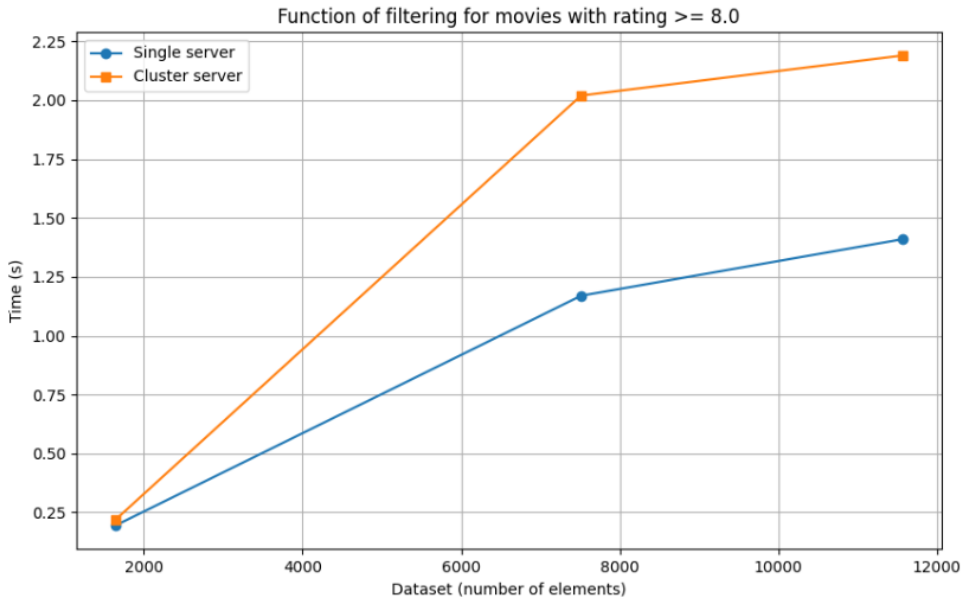
- Cluster with a dataset of 11563 elements.

```
New database, Number of elements in the database: {'172.21.0.6:6379': 3919,
'172.21.0.5:6379': 3919, '172.21.0.3:6379': 3809, '172.21.0.4:6379': 3809,
'172.21.0.2:6379': 3835, '172.21.0.7:6379': 3835}
Films with IMDb rating>= 80.0  are  928
Execution Time for the function is : 2.198124885559082 seconds
Average rating  of film: 85.62068965517241
Execution Time for the function is : 0.14643597602844238 seconds
Films with IMDb rating>= 40.0  are  4511
Execution Time for the function is : 2.4622507095336914 seconds
Average rating  of film: 69.75105298160054
Execution Time for the function is : 0.884711742401123 seconds
Films with IMDb rating>= 60.0  are  3461
Execution Time for the function is : 2.3615658283233643 seconds
```

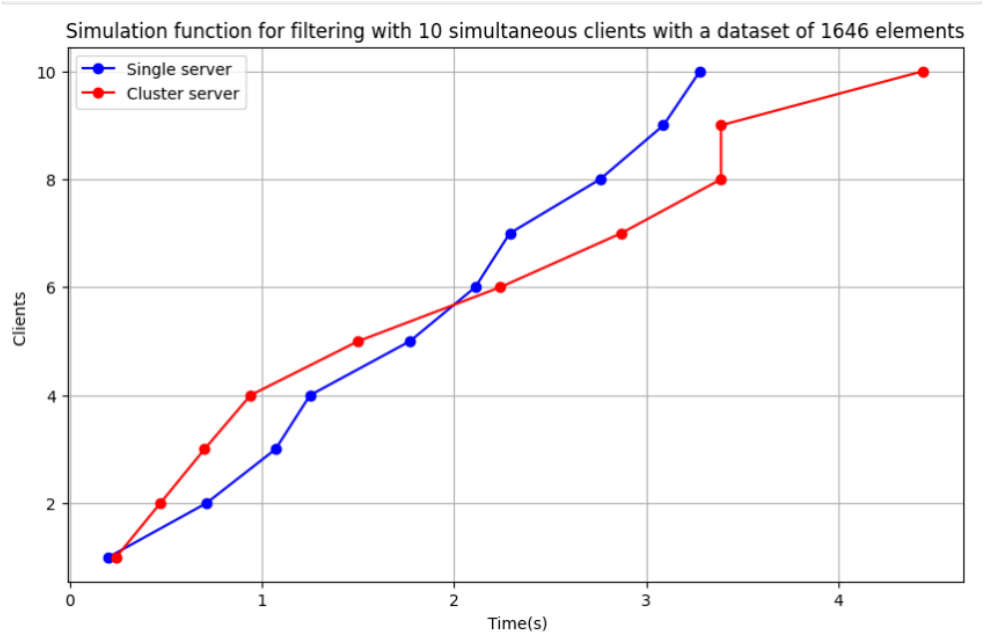
- Cluster server with a dataset of 11563 elements, simulation with 10 simultaneous clients for the function of filtering

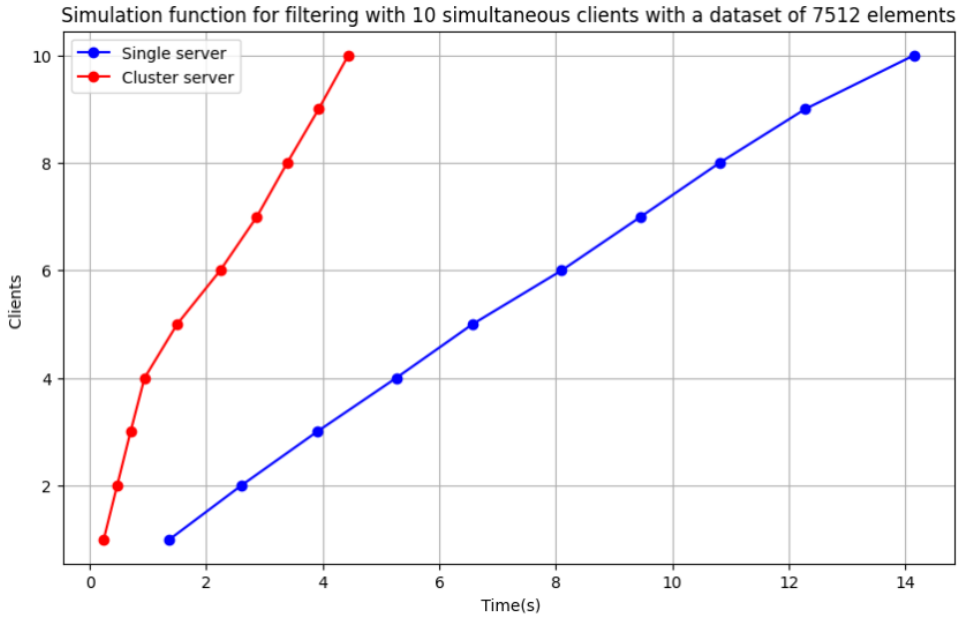
```
Client 1:
Films with IMDb rating>= 80.0  are  928
Execution Time for the function is : 4.101705312728882 seconds
Client 1 completed in 4.10 seconds
Client 2:
Films with IMDb rating>= 40.0  are  4511
Execution Time for the function is : 4.3308329582214355 seconds
Client 2 completed in 8.43 seconds
Client 4:
Films with IMDb rating>= 80.0  are  928
Execution Time for the function is : 4.108964204788208 seconds
Client 6:
Films with IMDb rating>= 60.0  are  3461
Execution Time for the function is : 4.03736138343811 seconds
Client 6 completed in 16.57 seconds
Client 7:
Films with IMDb rating>= 80.0  are  928
Execution Time for the function is : 3.590791702270508 seconds
Client 7 completed in 20.16 seconds
Client 10:
Films with IMDb rating>= 80.0  are  928
Execution Time for the function is : 3.942018508911133 seconds
Client 10 completed in 24.09 seconds
Client 3:
Films with IMDb rating>= 60.0  are  3461
Execution Time for the function is : 4.437882661819458 seconds
Client 3 completed in 28.55 seconds
Client 4 completed in 12.54 seconds
Client 9:
Films with IMDb rating>= 60.0  are  3461
Execution Time for the function is : 3.878084421157837 seconds
Client 9 completed in 32.41 seconds
Client 5:
Films with IMDb rating>= 40.0  are  4511
Execution Time for the function is : 3.93357515335083 seconds
Client 5 completed in 36.36 seconds
Client 8:
Films with IMDb rating>= 40.0  are  4511
Execution Time for the function is : 3.943467140197754 seconds
Client 8 completed in 40.28 seconds
```

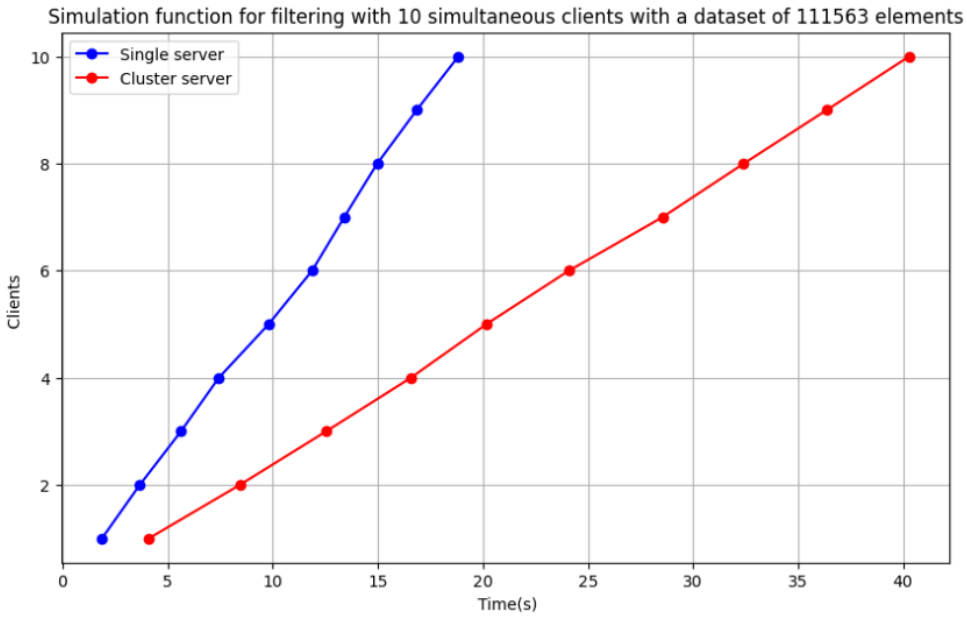
The graph shows the filtering times for movies with a rating greater than or equal to 8.0. The x axis represents the time in seconds, while the y axis represents the size of the dataset. The blue line represents the results obtained interacting with a single server, while the orange line represents the results with the cluster server. It is evident that the cluster server takes longer than the single server to complete the filtering as the dataset size increases. In cluster systems, there is an overhead associated with communication between nodes. This overhead can increase with the size of the dataset, making the overall process slower, also because the cluster requires the nodes to synchronize and coordinate to process the data. This can add an additional workload that is not present in a single server.



The 3 graphs below show the function of filtering based on the ratings, simulating 10 concurrent clients using threads. The x axis represents the time in seconds, while the y axis represents the number of the client. The blue line represents the time execution for the function using a single server, while the red line using the cluster. The charts demonstrates that the filtering function's efficiency decreases as the dataset size increases. Additionally, the average customer wait time is significantly higher when using a cluster compared to a single server, with the number of waiting customers increasing. This disparity is most pronounced for the largest dataset, reaching a maximum with the 10th customers within a cluster.







The tests conducted on the project provided valuable insights into the performance and scalability of Redis. In particular, we compared the execution times of filtering and aggregation operations on a single Redis server versus a Redis cluster. The results showed a linear relationship between dataset size and execution time, confirming Redis's efficiency in handling large volumes of data. However, we observed that execution times in the cluster increased due to communication overhead between nodes, especially with large datasets. Furthermore, by simulating concurrent requests from multiple clients using threads, we verified how server resources are contested, leading to higher response times compared to single requests. These tests confirmed that, despite the challenges of node synchronization and coordination, Redis remains an excellent choice for applications requiring high-speed data access and operational robustness.

Codes : https://github.com/ClaudiaEspo/DSE.git