NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Cloud Computing Systems

2024/2025

Project Assignment #1

Authors:

55806, Tomás Santos
72287, Eduardo Silveiro

Professores:

Sergio Duarte
Kevin Gallagher

November 10, 2024

Introduction

This year project for the "Cloud Computing Systems – 2024/25" course aimed to provide experience utilizing cloud computing services to develop applications that are scalable, fast, and highly available. The task consisted of porting an application, "TuKano," to the Microsoft Azure Cloud platform, using the best practices in cloud engineering.

The "TuKano" is a social network application inspired by video-sharing platforms, allowing users to upload, view, and interact with short video where the initial version operates with a single server, hosting services for managing user data, media metadata, and video storage. The project required some modifications to integrate Azure services, like Azure Blob Storage, Azure Cosmos DB, and Azure Cache for Redis.

This report is the way to evaluate our solution in terms of performance, analyzing throughput and latency for deployments.

Code Choices

During the development, we adopted strategies to simplify code management and allow flexibility. One of these decisions was using a Constants class to store important data, such as user names and specific configurations of our azure resources. When needed, we swapped between enum variables, avoiding to change the code directly in multiple places, which helped prevent hard-coded modifications. Additionally, in each endpoint method, we checked the type of the database flag and whether the cache was active or not, ensuring the application behaved correctly under different environment configurations.

Results Analysis

In this section, we present a comparison of the performance results between two database configurations: CosmosDB NoSQL and CosmosDB for PostgreSQL. Additionally, we analyze the impact of enabling caching on both configurations. These comparisons provide insights into how each setup influences key metrics, such as latency and throughput, and help evaluate the effectiveness of caching in enhancing overall application performance.

It is important to denote that this data was generally the “average” of the best results when running the tests and the endpoints working correctly were :

- **Register User:**
  - (POST /users/)
- **Get User's Shorts:**
  - (GET /shorts/{{ userId }}/shorts)
- **Upload Short:**
  - (POST /shorts/{{ userId }}?pwd={{ pwd }})
  - (POST /blobs/{{ blobUrl }})
- **Download Short:**
  - (GET /shorts/{{ shortId }})
  - (GET /blobs/{{ blobUrl }})

- **Like Short:**
    - (POST /shorts/{{ shortId }}/{{ userId }}/likes?pwd={{ pwd }})
- **Get Short Likes:**
    - (GET /shorts/{{ shortId }}/likes?pwd={{ pwd }})
- **Get User Follows:**
    - (GET /shorts/{{ userId }}/followers?pwd={{ pwd }})
- **Follow User:**
    - (POST /shorts/{{ userId1 }}/{{ userId2 }}/followers?pwd={{ pwd }})

SQL .vs NoSQL

1. Response Time Comparison

The response times for several key endpoints reflect improvements without caching in both database types NoSQL and SQL. Below is a comparison highlighting mean, P95, and P99 responses for this type of database..

Endpoint	SQL Mean (m/s)	No SQL Mean(m/s)	Improvement (%)	SQL (P95 ms)	No SQL (P95 ms)
Get User's Shorts	905	1266.4	-40%	1085.9	1408.4
Download Short	437	745.3	-70%	432.7	1130.2
Follow User	956	937	2%	963	944
Get Short Likes	1045	1462	-40%	104.3	1408.4
Register User	1452	882.6	26.7%	1456.5	882.6

Observations:

- SQL generally shows lower mean response times for most endpoints, particularly for Download Short and Get User's Shorts.

- No SQL demonstrated a slight improvement for the Follow User endpoint, achieving a 2% gain in mean response time.
- P95 values indicate higher variability in response times for No SQL compared to SQL, suggesting more consistent performance from SQL under these conditions.

2. Request Rate and Throughput Second

Scenario	Requests per Second
SQL	5/sec
No SQL	5/sec

Both scenarios have the same request rate, meaning both are processing requests at the same speed.

3.Downloaded Data Volume

No SQL downloaded significantly more data, which indicates a higher volume of interactions or the transfer of larger data .

Scenario	Requests per Second
SQL	343
No SQL	3719

4. HTTP Status Code Analysis

The success rate and error patterns shift positively with caching, with a higher proportion of 200 (success) responses and a noticeable reduction in server errors.

Status Code	SQL Count	No SQL Count	Change (%)
200	**26**	**32**	**+23%**
403	**12**	**15**	**-25%**
500	**3**	**1**	**-67%**

Summary of Performance Comparison:

- **HTTP 200 Requests:** No SQL processed far more successful requests.
- Response Time: SQL has faster and more consistent response times, while No SQL showed higher latency spikes with greater variation in response times (P95 and P99).
- **Request Rate:** Both systems had the same throughput (5 requests per second), indicating similar request processing capacity.
- **Data Volume:** No SQL downloaded much more data, suggesting heavier interactions and/or larger payloads.

Cache Analysis

1. Response Time Comparison

The response times for several key endpoints reflect improvements with caching. Below is a comparison highlighting mean, P95, and P99 response times before and after enabling caching.

Endpoint	Non-Cached (Mean ms)	Cached (Mean ms)	Improvement (%)	Non-Cached (P95 ms)	Cached (P95 ms)
Get User's Shorts	407	514.5	-26.4%	399.5	507.8
Download Short	513.7	504.6	1.8%	518.1	518.1
Follow User	1206	521.4	56.7%	1200.1	539.2
Get Short Likes	1408	505	64.1%	1130.2	507.8
Register User	1552	982.6	36.7%	1556.5	982.6

Observations:

- Follow User shows the most significant improvement with a 56.7% reduction in average response time.
- Register User and Get Short Likes also show marked reductions in average response times, suggesting caching is effectively lowering backend processing time for these data-intensive operations.
- Get User's Shorts shows a slight increase in response time, potentially due to cache write operations.

2. Request Rate and Throughput

The cached configuration achieves a higher request rate overall, indicating increased efficiency in handling requests. In the non-cached scenario, a maximum of 5 requests per second was recorded. With caching:

- **Request Rate:** The system reaches up to 10 requests per second, demonstrating that caching can nearly double the throughput by reducing database dependency.
- **Downloaded Data:** Cached scenarios downloaded less data, indicating that frequently accessed items were retrieved from the cache instead of querying the database.

Scenario	Request Rate (Requests/sec)	Downloaded Data Volume (Bytes)
Non-Cached	**5/sec**	**3300**
Cached	**10/sec**	**2116**

Observations:

The reduction in data volume and doubled request rate under caching supports that Azure Cache for Redis is effectively handling frequently accessed data, lowering the workload on the backend database.

3. HTTP Status Code Analysis

The success rate and error patterns shift positively with caching, with a higher proportion of 200 (success) responses and a noticeable reduction in server errors.

Status Code	Non-Cached Count	Cached Count	Change
200	26	39	+50%
403	12	9	-25%
500	3	0	-100%

Observations:

- The increase in 200 responses (from 26 to 39) indicates a higher success rate with caching.
- The reduction in 500 errors shows that caching decreases the load on the backend, resulting in fewer server-side errors.
- There is also a reduction in 403 errors, suggesting that caching might be reducing access restrictions .

4. Session Duration and Virtual User Completion

Caching positively impacts session length and user completion rates. The average session duration for virtual users is shorter in the cached scenario, reflecting faster task completion:

Metric	Non-Cached Avg (ms)	Cached Avg (ms)	Improvement (%)
Session Duration	1534.2	1118	27.1%
Completed VUs	21	30	+42.8%

Observations:

Shorter session durations and more completed user sessions with caching indicate enhanced performance .

Summary of Performance Comparison:

- Reduced response times for frequently accessed endpoints.
- Increased request rate and reduced downloaded data, indicating more efficient data retrieval.
- Higher success rate and fewer errors , demonstrating increased reliability.
- By adopting caching, the application enhances scalability and responsiveness, essential for a high-performance cloud application like "TuKano."
- This concludes the cache analysis; please let me know if further details on specific endpoints are needed!

Conclusion

In this project, we successfully ported the "TuKano" web application to run on a local (dockerized) Tomcat10, with remote Azure resources and to the Microsoft Azure Cloud, leveraging Azure Blob Storage, Azure Cosmos DB, and Azure Cache for Redis. This transition has allowed us to implement a scalable solution that follows best practices in cloud computing. However, several advanced features were not fully implemented, including Azure Functions, Geo-Replication support, Counting Views, and the "Tukano Recommends" system. Additionally, certain methods like feed, deleteAllShorts, and deleteAllBlobs functions, could not be thoroughly tested due to time and resource constraints.

Our ability to perform additional testing was also limited by the high cost of maintaining and running the Azure services required for this project, which impacted our student account budget. Despite encountering some 500 errors, when requests were tested more carefully without using Artillery, we consistently received success codes.