Cloud Computing Systems Project 1 - Report

AUTHORS

Bruno Melo - 60019
Frederico Silva - 60247

2024-11-10

Contents

1 Introduction **1**

2 Modifications Required **2**
2.1 Azure Deployment 2
2.2 Azure Blob Storage Integration 2
2.3 Cosmos Database Integration 2
2.4 Redis Cache Integration 2
2.5 PostgreSQL Integration Attempt 2
2.6 Geo-Replication Implementation 3
2.6.1 Cosmos DB Geo-Replication: 3
2.6.2 Blob Storage Geo-Replication: 3

3 Experimental Evaluation **4**

4 Analysis of Performance **5**
4.1 Hibernate-Based Operations 5
4.2 Cache Performance 5
4.3 Geo-Replication with Primary Region Selection 5

5 Conclusion **6**

References **7**

1 Introduction

The objective of this project was to understand how services available in cloud computing platforms can be used to enhance the scalability, speed, and availability of web applications [1]. For this end, we were tasked with porting an existing web application (Tukano) to the Microsoft Azure Cloud platform. The original base code that was provided needed to be modified to leverage the Azure Platform as a Service (PaaS) portfolio, in ways that agree with current cloud computing engineering best practices.

This report delves into the technical details of our solution, outlining the modifications made to the original base code to harness Azure's capabilities. Furthermore, we discuss the challenges encountered during the porting process and present a comprehensive experimental evaluation. By comparing the performance of the base solution against various cloud-based alternatives and add-ons, we aim to gain insights into the factors that significantly impact user experience and overall performance.

2 Modifications Required

A number of changes were made to the original program in order to fully utilize Azure's cloud capabilities.

2.1 Azure Deployment

The initial step involved deploying the application to Azure to ensure global accessibility. This was relatively straightforward, requiring only minor code adjustments in the TukanoRestServer class and the addition of necessary dependencies.

2.2 Azure Blob Storage Integration

To offload file storage from the client's local device, Azure Blob Storage [2] was integrated. This change was seamless, as the underlying structure facilitated a straightforward swap of the storage mechanism.

2.3 Cosmos Database Integration

The most significant modification involved migrating data related to users, shorts, likes, and followings to Azure Cosmos DB [3, 4]. This complex task required substantial code restructuring, including the creation of new classes to accommodate the transition from Hibernate to Cosmos DB while maintaining the original Hibernate functionality. Additionally, several issues with the original code's implementation were identified and resolved. This migration significantly enhanced the application's fault tolerance, reliability, and availability.

2.4 Redis Cache Integration

To improve response times, Redis Cache [5] was integrated to store frequently accessed data referring to users and shorts. This was a relatively simple implementation.

2.5 PostgreSQL Integration Attempt

An attempt was made to integrate PostgreSQL as an alternative database. This should have been a relatively simple procedure which only required to create the necessary resources in the Azure web platform and feed its parameters to the existing Hibernate functionality. However, despite efforts to configure Hibernate, the integration proved unsuccessful. Further investigation and troubleshooting are required to resolve this issue.

2.6 Geo-Replication Implementation

To ensure high availability and reliability, geo-replication was implemented for both Blob Storage and Cosmos DB.

2.6.1 Cosmos DB Geo-Replication:

Cosmos DB inherently supports geo-replication, and this feature was enabled within the Azure portal.

2.6.2 Blob Storage Geo-Replication:

Two approaches were considered for Blob Storage geo-replication [6]:

Read-Access Geo-Redundant Storage (RA-GRS): This approach allows for automatic fail over to secondary regions in case of primary region failures.

Custom Geo-Replication: This approach involves creating two storage accounts in different regions (North Europe and US North) and implementing a custom synchronization mechanism. A client can specify which of these regions he wishes to use as the primary. Write operations are performed synchronously for the primary region, after which thread is launched to perform the same operation asynchronously in the secondary region, before return the result to the client. Read operations are directed to the primary region or, in case of failures, to the secondary region. This approach provides stronger consistency guarantees.

After careful consideration, the second approach was chosen for its robustness and flexibility. Later we will discuss the performance and latency observed when choosing either region as the primary.

3 Experimental Evaluation

We developed custom YAML tests to assess the performance of our solution, evaluating all operations across each distinct module. Initially, we measured the performance of the base Hibernate solution. Next, we tested the system's performance with Cosmos DB and Azure Blob Storage, without cache. Following this, we assessed the system's performance with caching enabled. Finally, we analyzed the performance of the geo-replicated Azure Blob Storage, designating either the Europe or US region as the primary. All the results of our experimental evaluation can be consulted on our GitHub Repository, under directory **results** [7].

Below, we present the results for selected methods:

	Hibernate	**Cosmos DB/Blob Storage**	**Cache**
Post User	914.8	154.9	436.1
Get User ID	101.6	211.4	338.1
Get Pattern	124.1	151.3	170.1

Table 1: Mean HTTP response time for User related operations

	Hibernate	**Cosmos DB/Blob Storage**	**Cache**
Post Short	377.7	565.4	1076.0
Get Short ID	80.7	402.9	99.3
Get Shorts	130.1	411.2	227.4
Post Follow	123.3	312.6	268.6
Get Followers	78.2	263.0	1100.4
Post Like	113.3	572.6	171.0
Get Feed	108.6	274.5	310.5
Delete Shorts	534.3	1587.2	-

Table 2: Mean HTTP response time for Short related operations

	Hibernate	**North Europe Primary**	**US North Primary**
Upload Blob	73.9	460.4	1430.0
Download Blob	87.5	207.5	727.8
Delete All Blobs	69.9	378.1	873.7

Table 3: Mean HTTP response time for Blob related operations

4 Analysis of Performance

4.1 Hibernate-Based Operations

The data shows that, in general, operations using Hibernate are the fastest among the tested approaches. This result aligns with expectations, as Hibernate operates locally on the device, eliminating network latency. While there were some outliers, such as an unusually slow performance in the `Post User` operation, this deviation could be attributed to temporary device processing spikes. Importantly, this outlier does not detract from the overall conclusion that Hibernate provides the best performance.

4.2 Cache Performance

The caching approach presented inconsistent results across operations. We implemented caching only for users and shorts, as these were determined to be the most frequently accessed data, leaving out likes and followers to simplify the setup. However, we encountered issues with cache stability, including occasional cache crashes and broken connections. When the cache crashes, the application reverts to then non-cached execution, leading to significant variance in mean response times between the cached and non-cached scenarios. Despite these inconsistencies, caching demonstrated a clear performance benefit, particularly in retrieving shorts, which showed retrieval speeds 2-4 times faster with caching enabled.

4.3 Geo-Replication with Primary Region Selection

When examining geo-replication, we compared the impact of choosing different primary regions, Europe North versus US North. As expected, operations using Hibernate remained the fastest across all configurations. Among geo-replicated operations, selecting Europe North as the primary region resulted in noticeably faster response times than using US North, owing to our physical location in Europe during testing. This difference is evident in the data, with response times approximately three times slower when US North is the primary region. These results highlight the significant impact of geographic proximity on latency, consistent with expectations for geo-distributed systems.

5 Conclusion

In this project, we explored the impact and performance of cloud tools in deploying and managing web applications. Our study highlights the strengths of cloud-based solutions, which offer remarkable versatility and accessibility, enabling global application deployment with minimal upfront cost and operational risk. Notably, our own deployment and testing incurred a total expense of only 10 euros, even under varying workload conditions, which demonstrates the cost-efficiency of cloud platforms and it's tools.

A key advantage of cloud services lies in their reliability and fault tolerance. By allowing users to define data consistency levels and customize data replication across regions, cloud tools can support applications with diverse performance and availability requirements. While cloud-based solutions may introduce an increase in latency compared to local deployment, this latency remains within acceptable limits. Additionally, intelligent use of the existing tools, such as optimized use of caching and strategically located data centers, can mitigate latency impacts. For applications requiring even higher performance, options for increased throughput and processing capacity are readily available.

We would also like to acknowledge the use of AI tools, specifically Chat GPT for some code refactoring and troubleshooting.

Overall, our findings confirm that cloud tools are a powerful and adaptable choice for modern web application deployment. They empower developers to deploy resilient, globally accessible applications with flexibility, scalability, and cost-effectiveness.

References

[1] P. Borra, “Exploring microsoft azure’s cloud computing: A comprehensive assessment,” *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, 2022.

[2] Microsoft Azure, “Introduction to azure blob storage.” `https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction`. Accessed: 2024-11-10.

[3] Microsoft Azure, “A technical overview of azure cosmos db.” `https://azure.microsoft.com/en-us/blog/a-technical-overview-of-azure-cosmos-db/`. Accessed: 2024-11-10.

[4] Microsoft Azure, “Azure cosmos db whitepapers.” `https://learn.microsoft.com/en-us/azure/cosmos-db/whitepapers`. Accessed: 2024-11-10.

[5] Microsoft Azure, “Azure cache for redis: Overview.” `https://learn.microsoft.com/en-us/azure/azure-cache-for-redis/cache-overview`. Accessed: 2024-11-10.

[6] Microsoft Azure, “Azure storage redundancy options.” `https://learn.microsoft.com/en-us/azure/storage/common/storage-redundancy`. Accessed: 2024-11-10.

[7] Reidasfestas, “Scc project1: Tukano.” `https://github.com/Reidasfestas/SCC_Project1/tree/main/scc2425-tukano/results`, 2024. Accessed: 2024-11-10.