Cloud Computing Systems Work Report

Bruno Cabrita[57833], Diogo Almeida[58369], and Diogo Fona[57940]

NOVA School of Science and Technology - MSc in Computer Engineering
{brm.cabrita,daro.almeida,d.fona}@campus.fct.unl.pt

1 Introduction

The goal of this work is to understand how services available in cloud computing platforms can be used for creating applications that are scalable fast, and highly available.

1.1 Context

This work consists in the design and implementation of the backend for an auction system like E-Bay and companion scripts for testing the system.

The system will manage auctions. Users can create auctions and bid on open auctions. User can also pose question about the product of an auction. A question can only be answered by the user that created the auction, and there can be only one answer for each question.

2 Design

2.1 Architecture

The system mantains the following information:

- **Users**: information about users, including the nickname, name, (hash of the) password, photo;
- **Media**: manages images and videos used in the system;
- **Auctions**: information about auctions, including for each auction a title, a description, an image, the owner (the user that created the auction), the end time of the auction (i.e. until when bid can be made), the minimum price, the winner bid for auctions that have been closed, the status of the auction (open, closed, deleted).
- **Bids**: Each bid includes the auction it belongs to, the user that made the bid and the value of the bid.
- **Questions**: auctions' questions and replies. Each question includes the auction it refers to, the user that posed the question and the text of the message.

To support the storage, querying, and processing of this information, we use the following cloud software and hardware components. In our work these are provided by Microsoft Azure.

App Service which clients interact directly with. Afterwards the service communicates with other components to return information to the clients. The software running in this service is implemented by us in Java.

Database that stores the information of the objects described above in a structured way and allows querying them. In this work we use CosmosDB as the database.

Blob Storage that stores binary data, images and videos of users and auctions in this context.

Cache which stores query and computation results, so later requests of the same information doesn't need to be queried or computed again. In this work we use Redis for caching.

App Functions that do additional computations that don't need to be constantly running in the App Service. We use Azure Functions to provide this service.

Search Service which provides search approaches based on AI, like searching for relevant information based on a text query. We use Azure Cognitive Search for the searching service.

2.2 Endpoints

The App Service provides the following REST endpoints for communication with clients. The descriptions encompass a number of endpoints, having the following as the base ones:

- **Users** (/rest/user): Create, authenticate, get, delete and update a user. List auctions a user has created and has bidded for;
- **Media** (/rest/media): Upload and download media;
- **Auctions** (/rest/auction): Create, get and update an auction. List auctions that were recently created, are closing soon, and popular ones. Query auctions.
- **Bids** (/rest/auction/{id}/bid): Bid for an auction or list its bids;
- **Questions** (/rest/auction/{id}/question): Post a question on an auction, reply to a question and list an auction's questions. Query questions from an auction.

3 Implementation

In this section we provide some details on some relevant highlighted aspects of the implementation.

3.1 Database

3.2 Caching

We store in Redis cache all objects (User, Auction, Bid and Question DAOs) until they are invalidated by either timeout, after 1 hour (or another configurable value), or are updated, in which they must be replaced in the cache, or deleted.

Lists of IDs of these objects are formed in cache with each write operation. These constitute the result of some operations such as listing 1. a user's auctions, 2. a user's followed auctions, 3. an auction's bids and 4. an auction's questions, and also for saving the winning bid of each auction. The objects can then be calculated from their IDs from cache.

The advantage of this approach is when an object is updated, since the lists are of immutable IDs we don't need to invalidate one which contained that object.

The disadvantage of this is we need to get each of the objects through their IDs from the cache, or database in the worst case. To amortize this (this isn't currently implemented), we could bundle all IDs and get their respective objects in one request.

Additionally we keep track of results of some listing operations which involve ordering. This is possible with Redis' sorted sets and incrementing scores. Those operations are the listing of the closing auctions (ordered by the closest auction's end time), of the recent auctions (ordered in list on insertion), and of popular auctions (ordered by number of bids on them, which are incremented with every added bid). The top 20 of each operation are returned to the application.

The listing of recent and popular auctions in particular need to have cache enabled to work, meaning these queries are not computed in the database.

Session Keeping Using Redis cache, we keep the session keys of users after they authenticate in the system. Each session key expires in 30 minutes, after which users need to authenticate again to use the system. Furthermore some subsystems could be implemented over this: the session key could be removed after the user "logs out"; after each operation the expire time is refreshed (idle logout system).

3.3 Azure Functions

Blob Replication For blob replication through different data centers, our approach was to use an Azure Function where, on Blob creation/update in one data center, it would upload that same blob to their respective blob storage center. This way, a client uploading a media resource in one data center, that same media resource will be saved in all other data centers.

3.4 Querying

To search for auctions based on the user's text query, we use Azure Cognitive Search where it searchs for an occurence in the documents (this case auctions) and returns those documents as a result.

The same logic is applied for querying questions from an auction: client provides a text query to which is used to look for an occurence in the auction's questions' descriptions and return the gathered documents as a result to the client.

4 Evaluation

In this section we evaluate our work, and demonstrate why the use of technics such as caching and replication are relevant.

5 Conclusions