Distributed Systems: Google App Engine session 2

Fatih Gey, Stefan Walraven, Wouter De Borger and Wouter Joosen

November 24 and December 1, 2015

Overview

There will be 2 exercise sessions on Google App Engine:

- 17/11/15 in pc lab:
  - Introduction to Google App Engine (GAE) and GAE datastore.
- 24/11/15 and 01/12/15 in pc lab:
  - Extend Google App Engine application with task queues and worker processes.
  - **Submit** the final version of the code on Toledo **before Friday, December 4, 19:00**.

In total, **each** student must submit 1 assignment to Toledo.

1 Introduction

Goal: In the previous session, we started with deploying our car rental application on top of Google App Engine. As a first step, we adapted the data entities in order to make them compliant with the JPA API provided by Google App Engine (GAE) [1]. In this session, we focus on more cloud-related requirements: high availability and performance scalability. By using indirect communication techniques offerend by GAE, we will redesign the application to be more loosely-coupled in order to address these requirements.

Approach: This session must be carried out in groups of *two* people. You will have to team up with the same person for each of the sessions in this course.

Submitting: On Friday, **each** student must submit his or her results to the Toledo website. To do so, close your Eclipse, enter the base directory of your project and zip your solution using the following command:

```
ant -Dzip.filename=firstname.lastname.zip
```

Please do not use a regular zip application to pack your entire project folder, as this would include the entire Google App Engine SDKs (about 40 MB). *Make sure that your report (PDF), which should be placed at the base directory of your project, and addition libraries you solution made use of are also included in the zip file.* Submit the zip file to the Toledo website (under Assignments) before Friday, 19:00.

Important:
- When leaving, make sure your server is no longer running.
- Retain a copy of your work for yourself, so you can review it before the exam.

2 Extra Requirements for Cloud Services

Cloud computing is an increasingly popular paradigm for software developers and various types of software users because of the *economies of scale*. Simply said, this means the more users a software application serves, the lower are its relative costs. This also provides new challenges: The parallel use of a cloud application by many people at once may render the direct use of transactions (or transactional "all-or-nothing" behaviour) impractical, e.g. imagine thousands of people attempting to rent a car of the same type. All transactions would be queued, making the user wait and letting the system appear very slow (poor user experience).

Using Indirect Communication. An approach to avoid such a delay is to decouple the part of the application that communicates to the client, i.e. the front end, from the part that is involved in the bottleneck activity of that application, i.e. the back end. Thereby, availability and performance requirements become valid for the front end only, allowing the back end to introduce additional delay for its processing.

For example, in the car rental application, we could divide the application into a part that only serves the purpose to interact with the customer, and a second part that handles all the back-office work, i.e. making the actual reservations, handing-out and receiving the cars as well as doing the paper work.

3 Assignment

First, we setup and test an extended version of the car rental application. If your previous assignment is properly implemented, this should work out of the box. Next, we design and implement a loosely-coupled version of the application, taking the additional requirements for cloud environments into account.

3.1 Preliminaries

Extend your car rental application from last session to setup the full-fledged version of the car rental company:

1. Download `GAE2_src.zip` from Toledo. In this ZIP file, we provide additional Servlets and JSPs to enable car renters to create quotes in the different car rental companies and to confirm them. These additional files use the `ds.gae.CarRentalModel` class that you implemented during the previous session.

2. Unzip `GAE2_src.zip` in your project in Eclipse and (thereafter!) open it. Ensure that the function `confirmQuotes(List<Quote>)` in `ds.gae.CarRentalModel` confirms either all quotes or none (when one of the car types is not available any more). You do not have to provide strong consistency guarantees, and may assume the absence of failures.

3. Run the application by deploying your GAE project and calling `http://localhost:8888` in your browser. **Please make sure that the application works properly before going on to the next task**, e.g. create at least a quote on each car rental company and request to confirm all quotes at once. The tab `Bookings` shows all reservations that succeeded. The last tab corresponds to the persistence test that was used during the previous session. To check its reservations, login as "TestUser".

3.2 Loosely-coupled Back and Front End

Recall from the previous GAE session that we had a reservation system that was based on the Java EE implementation. We will now adapt this system step-by-step to better utilize the scalability opportunities offered by the underpinning cloud environment.

Design: Make a design of the distributed car rental agency that uses the Task Queues provided by the App Engine SDK [2]. Therefore, divide your application into a front- and a back-end part in order to loosely couple these parts using indirect communication (as explained above), and submit it as a design report. **The design report consists of 2 diagrams and maximally 2 sections with design decisions in English, combined into 1 PDF. Be concise and to the point!** Don't forget to put your names on the front page of your report and to store it in the base directory of your solution.

First, create a first section called `GAE Exercise 3.2` in your design report, and briefly discuss your answers to the following questions (**in English**). Please try to limit the content to 150 words.

- At which step of the workflow for booking a car reservation (create quote, collect quotes, confirm) would the indirect communication between objects or components kick in?
- Which kind of data is passed between both sides? Does it make sense to persist data and only pass references to that data?

Further, you will have to make the following design artifacts: a *deployment diagram* which shows which processes are executed on which system[1]. Also create a *sequence diagram* showing a scenario (conform your design!) that uses Task Queues, including the back-channel (see below).

Implementation: Implement your design from Section 3.2. Please use **Push Queues** (no Pull Queues) and **Instance-Workers** (no Backend processes) or `DeferredTasks` [5]. For the implementation of your worker, we provided an additional but empty Servlet (`ds.gae.Worker`) that is bound to the URL `http://localhost:8888/worker`.

Important: Think about how to realize a back-channel to the caller, i.e. how to inform the front end whether a background task has executed sucessfully or not. The latter case may be even more important to report back, as an omission of a positive feedback can be ambiguous and may also depict that the task at hand has not been processed yet. You may use any (free) service provided by the GAE SDK to realize this. Depending on your solution, this requires to add or modify Java Servlets or JSPs. In case you want to return some information to the car renter on a web page, you may use the given JSP `confirmQuotesReply.jsp` (optional). For your solution, please bear in mind that backend-processing (and thereby the delay of the feedback) may be of long duration, e.g. up to hours and days.

3.3 Challenging Parallel-Processing Scenario

With the coupling of front- and back-end parts of our application through an indirect communication channel, we enabled back-end tasks to be processed independently. In addition, workers in GAE's Task Queues are by default set to run in parallel. While parallelism is usually a desirable property of a cloud service, as it enables scalability and thus faster overall processing, it may also endanger the application's state consistency.

Create a second section called `GAE Exercise 3.3` in your design report, and answer the following questions (**in English**). Please try to limit the content to 150 words.

Assume a scenario in which two different clients try to confirm a couple of tentative reservations, i.e. their quotes are queued to be processed by the back end. Both include a tentative reservation to the last available car of a certain car type, so that, assuming correct behaviour of the car rental application, it should fail to confirm the quotes to one of them.

- Is there a scenario in which the code to confirm the quotes is executed multiple times in parallel, resulting in a positive confirmation to both clients' quotes?
- If so, can you name and illustrate one (or more) possibilities to prevent this bogus behaviour?

[1]This diagram has to be documented as it is deployed on the actual App Engine platform, not the lab deployment where everything runs on the same machine.

- In case your solution to the previous question limits parallelism, would a different design of the indirect communication channels help to increase parallelism? For this question, you may assume that a client will have quotes belonging to one car rental company only.

4 Practical Information

- Data can be passed from a Servlet to a JSP by using a HTTP session: attributes that are set by the Servlet, can be read by the JSP. For further information, we refer to the previous session's assignment.
- To inspect the data in the datastore or the tasks that are currently waiting in the Task Queue, start the application and go to `http://localhost:8888/_ah/admin`.
- To restart your application with an empty development datastore, remove the files under `war/WEB-INF/appengine-generated/`. Refresh your project first!
- For the final design artifacts you can use Visual Paradigm:

```
/localhost/packages/visual_paradigm/bin/Visual_Paradigm
```

Important: In case of problems, try the following things:

- If the console gives a warning about classes that are not enhanced, refresh your project (F5) and try again. Otherwise, make a small modification in an entity class and save it.
- Clean your project: `Project ▷ Clean...` (in menu bar).

5 Summary

After both GAE exercise sessions, you should be able to answer the following questions:

- What is the benefit of running an application or a service in the cloud?
- Is it possible to migrate a Java EE application to a cloud platform such as Google App Engine? While doing so, what are the hot spots of an application to be taken care of?
- Why is indirect communication handy when it comes to services running in the cloud? What are the pitfalls when changing from synchronous to asynchronous communication?

References

[1] Google, Inc. *Google App Engine.* `https://cloud.google.com/appengine/docs`.

[2] Google Inc. *Task Queue Java API.* `https://cloud.google.com/appengine/docs/java/taskqueue/`.

[3] Google Inc. *Using Push Queues.* `https://cloud.google.com/appengine/docs/java/taskqueue/overview-push`.

[4] Google Inc. *Java Task Queue Configuration.* `https://cloud.google.com/appengine/docs/java/config/queue`.

[5] Google Inc. *DeferredTask Java Interface (JavaDoc)* . `https://cloud.google.com/appengine/docs/java/config/queue`.

Good luck!