Advanced Software Engineering FS 2017
s.e.a.l. – software evolution & architecture lab
University of Zurich

Project Description

Introduction

Your task in this project is to develop a simple cloud-based service for Twitter-based sentiment analysis[1]. Sentiment analysis is, broadly, the process of finding out (typically automatically) what the general feeling ("sentiment") of one or more Web communities (for instance, the blogosphere or the Twitter community) about a company or product is. This sort of analysis has become an increasingly relevant marketing tool in recent years.

Your service should provide two basic functionalities:

1. Prospective customers can *register* for your service, that is, they provide their company name for sentiment monitoring. You do not need to bother with other boring details, like payment information, at this point.
2. Afterwards, registered customers can *query* the aggregated sentiment for a specified time period. For simplicity, the output of your service should be a simple numerical value between 0 (people with pitchforks have been sighted striving towards the company headquarters) and 1 (people buy whatever the company CEO tells them to buy).

[1]`http://www.computerworld.com/s/article/9209140/Sentiment_analysis_comes_of_age`

System Description

You are expecting that this service will pick up in popularity quickly, hence, you are designing and implementing your service based on an elastically scaling cloud computing infrastructure.

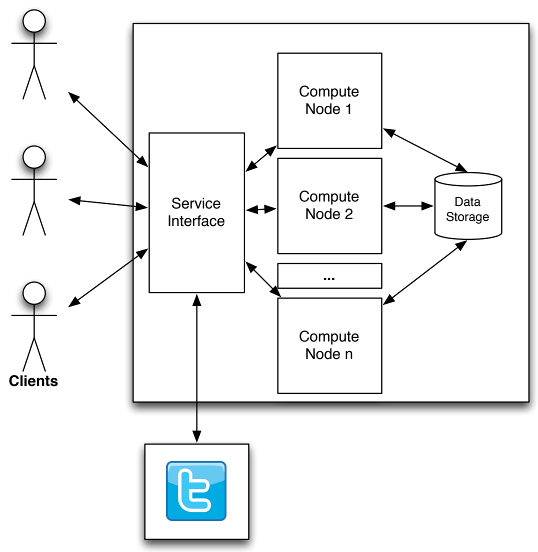

Figure 1: Very High-Level System Architecture Sketch

As a starting point, Figure 1 depicts a very high-level sketch of a possible system architecture. Essentially, clients communicate with your service through a service interface, which also acts as a load balancer. The actual implementation of this service is parallelized in a "cluster" of n compute resources. The size of n should be dynamic and depend on the current load on the system. For your project, you should further flesh out this architecture. For instance, you can split up your computing into more microservices or add additional databases. However, keep the core idea in tact that sentiment processing is handled in dedicated compute nodes whose number varies due to load.

For building the actual sentiment analysis part of the project, you can make use of an external library, for instance this Google Code project[2] if you use Java, or this GitHub project[3] for NodeJS. Do not spend too much time on tuning the actual sentiment analysis. For this project it is not central how well the analysis itself performs. To test your application you should retrieve live data directly from Twitter (i.e., via its API). Your service should implement the following core functionality:

1. It provides a simple way to register new terms, for which the service will then support sentiment analysis.
2. It allows to query the current sentiment value for previously registered terms. To this end, the service shall aggregate the (atomic) sentiments calculated for all tweets that contain the term in a meaningful way.
3. The service should monitor its performance (e.g., CPU load, time it takes to generate a sentiment value, time it takes to aggregate sentiment values on request, etc.) and acquire and release computing resources from the cloud depending on its current performance. If the load goes up, the service may acquire more resources from the cloud, if the load goes down, the service may release resources.

[2]`https://code.google.com/p/twitter-sentiment-analysis/`
[3]`https://github.com/thisandagain/sentiment`

4. A nice Web interface for your service. The web interface should act as both, an interface to register terms to monitor and show results, as well as a dashboard to show the state of the system (e.g., backend CPU load on each compute node).

Tip: in the past, it has proven useful to mine a database of historical tweets and use this as test data.

When building your solution, keep the concepts we discussed in the ASE lecture in mind. Think about the architectural styles and patterns used in your solution. Why did you go for this architecture? How does your architecture support scaling and elasticity? What are the drawbacks of your architecture? What functional and cross-cutting concerns are there in your solution? Do you see a point in using AOP to implement them? You should also consider splitting up your system into microservices, and assign the responsibility for a service to a specific person in the team. Comment on all of these aspects during the presentations. This will be part of the presentation grading.

Presentation

The outcomes of this project should be presented in two live demos. The first presentation is during the mid-term meeting, and should cover at least the tasks of Stage 1 (see below), the second presentation is during the final meetings and contains all your results. Every member of your team has to participate in either the first or the second presentation. Each presentation needs to consist of a regular presentation with slides and a demo of your tool. Carefully think about how you are actually going to demonstrate your solution, as this will be part of your grading. You have 20 minutes per presentation (strict). 10 minutes will be reserved for discussion and changeover.

Stage 1

1. Get started with your cloud technology. If necessary, register an account and deploy some example services.
2. Design your project. What is the architecture of your solution going to be? What rationale do you have for your architectural decisions?
3. Think about technology. What programming language(s) will you use? How will you devide your work?
4. Implement a proof-of-concept for your sentiment analysis algorithm. Figure out how you can get data from Twitter, and how you can analyze individual tweets.
5. Think about how to demo your thoughts for Stage 1. Maybe it is a good idea to already have a first version of the web interface ready.

Stage 2

1. Think about how to aggregate individual sentiment values. Do you want to consider all data, or just a relevant subset of tweets?
2. Implement your design and architecture. Build all the functional features mentioned above.
3. Think about performance monitoring and scalability. When should your solution scale up and scale down? How do you test this?
4. Create a polished demo scenario. Make sure that we actually see what is going on in the background.

Teams

Every team will consist of about 3 students. We will form teams in class during the kickoff meeting. Teams are expected to solve their project together.

Grading

A maximum of 40 points are awarded in total for the project. Of this, up to 25 points are awarded for the tool, and and up to 15 points are awarded for the presentations. All gradings will necessarily be subjective (we judge the quality and creativity of solutions, as well as the presentations). The hard deadline for the project is Friday before the final presentations. Please submit a ZIP file with your code, deployment instructions, and the presentations via email to leitner@ifi.uzh.ch. Late submissions will not be accepted.