Institut für Informationssysteme
Abteilung für Verteilte Systeme
VU Advanced Internet Computing (184.269)

Project Topic 3 - Cloud Computing

1 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).

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 [1, 4].

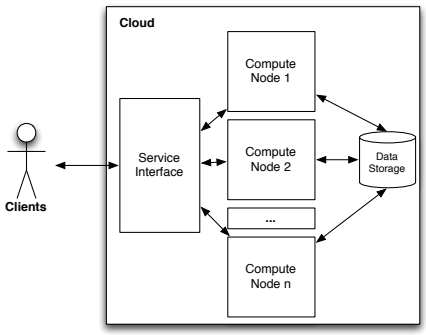

Figure 1: System Architecture Sketch

A very high-level sketch of the system architecture we propose is given in Figure 1. Essentially, clients communicate with your service through a service interface. The actual implementation of this service is parallelized in a "cluster" of n cloud virtual machines. The size of n should be dynamic and depend on the current load on the system. For practical reasons (see tasks below), your system should be implemented in Java.

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

Outcomes:

The expected outcomes of this project are three-fold: (1) the actual project solution, (2) a brief paper that analyses the scientific state of the art in elastic cloud applications, and (3) two presentations of these results (paper and tool).

Project Solution

The project should be hosted on a public git repository, whereas the URL to the repository has to be submitted. We recommend either Github[2] or Bitbucket[3]. Every member of your team should have a separate Github or Bitbucket account. It is required to provide an easy-to-follow README that details how to deploy, start and test the solution. Test whether it is actually possible to build and deploy your solution according to your instructions on a random Ubuntu Linux, Windows or Mac OS X machine. Make sure that it easily possible to test both, Stage 1 and Stage 2 (e.g., via separate Maven profiles).

Paper

The paper should provide an overview about the scientific state of the art in elastic cloud applications. Note that the paper is not the documentation of your tool – it should discuss scientific papers related to this topic in the style of a seminar paper. Good starting points for finding related scientific papers are the sources cited in this text, Google Scholar[4], IEEEXplorer[5] or the ACM Digital Library[6]. Use the ACM 'tight' conference style[7] (two columns), and keep it brief (3 pages). You do not necessarily need to install a LaTeX environment for this - you can use writeLaTeX[8], a collaborative paper writing tool as well.

Presentations

There are two presentations. The first presentation is during the mid-term meetings (Nov. 28th and Nov. 29th), and should cover at least the tasks of Stage 1 (see below), the second presentation is during the final meetings (Jan. 30th and Jan. 31st) 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 e.g., Slides part 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).

Grading

A maximum of 50 points are awarded in total for the project. Of this, up to 25 points are awarded for the tool, up to 10 points are awarded for the paper, and up to 15 points are awarded for the presentations. All gradings will necessarily be subjective (we judge the quality and creativity of solutions, presentations, and papers).

Deadline

The hard deadline for the project is **January 29th, 2014**. Please submit a link to your solution repository, deployment instructions, the paper, and the presentations as a single ZIP file via TUWEL. The submission system will close at 18:00 sharp. Late submissions will not be accepted.

[2] `https://github.com`
[3] `https://bitbucket.org`
[4] `http://scholar.google.at/`
[5] `http://ieeexplore.ieee.org/Xplore/home.jsp`
[6] `http://dl.acm.org/`
[7] `http://www.acm.org/sigs/publications/proceedings-templates`
[8] `https://www.writelatex.com`

Stage 1

In the first stage, this system should be implemented on top of a private Infrastructure-as-a-Service (IaaS) cloud. You will receive an account for Amazon Web Services[9] (AWS), which you can use to deploy and test your solution. To actually build your application on top of AWS, we ask you to use a research prototype of ours, CloudScale[10] (see also [3]). CloudScale allows you to build your elastic application like a local multi-threaded Java application, and handles all interaction with AWS for you. If you have problems getting started with CloudScale, we should be able to help. Should you discover any unexpected problems or bugs, or think that essential features are missing, please add them to our issue tracker[11].

Tasks:

1. Build the business logics of your service, i.e., implement the sentiment analysis. Note that this is required, but not the actual core of the project, so don't spend too much time on tuning your analysis. For implementing the actual sentiment analysis, you can use this Google Code project[12] (nevermind that the documentation is in Italien :) you should be able to just copy the code and execute it). You receive a Twitter data set from us to test your application with (see TUWEL). Alternatively, you can also use a library such as Twitter4J[13] to get some data directly from Twitter.

2. Add the Java annotations as required by CloudScale (see the documentation on Google Code for details), and test your application in the `local` environment.

3. Test your application in the actual AWS EC2 cloud by changing the environment to `ec2`.

4. Define a useful scaling policy for your application. This means you should decide when your application is overloaded, and how you find this out.

5. In this context, you should also think about a sensible strategy for scaling up and down. In the real world, instances can take minutes to actually become available, so it may make sense to keep instances in "reserve" despite not being used at the moment, to cover unexpected load spikes. On the other hand you should of course not just acquire as many instances as you can get from the cloud and be done with it (this would counter the entire point of an elastic application).

Stage 2

In the second stage, you should investigate the usage of a Platform-as-a-Service (PaaS) cloud instead of the IaaS and CloudScale. PaaS clouds promise to handle elasticity transparently for you as a developer. On the other hand, PaaS means giving up the freedom of having actual root access to the machines running your service, and having to program against the API defined by the PaaS provider. We will use Google AppEngine[14] in this stage. For your paper, you can also look at AppScale[15], which is an open source implementation of AppEngine (see also [2]).

Tasks:

1. Register an account with Google AppEngine.

[9] `http://aws.amazon.com`
[10] `https://code.google.com/p/cloudscale/`
[11] `https://code.google.com/p/cloudscale/issues/list`
[12] `https://code.google.com/p/twitter-sentiment-analysis/`
[13] `http://twitter4j.org/en/index.html`
[14] `http://cloud.google.com/appengine`
[15] `http://www.appscale.com/`

2. Use the PaaS services of AppEngine to deploy your sentiment analysis service there. Note that the middleware should now take over handling the elasticity of your application. Hence, a lot of code from Stage 1 will likely be redundant now and should be removed (including the code you added to use CloudScale).

3. Compare with the Stage 1 solution. What are the advantages and disadvantages of the PaaS cloud model, as compared to the CloudScale model? What did you like better and why?

References

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[2] Chandra Krintz. The appscale cloud platform: Enabling portable, scalable web application deployment. *Internet Computing, IEEE*, 17(2):72–75, 2013.

[3] Philipp Leitner, Zabolotnyi Rostyslav, Waldemar Hummer, Christian Inzinger, and Schahram Dustdar. Cloudscale: Efficiently implementing elastic applications for infrastructure-as-a-service clouds. Technical Report TUV-1841-2013-1, Vienna University of Technology, 2013. `http://stockholm.vitalab.tuwien.ac.at:8090/TechReportGenerator/reports/TUV-1841-2013-1.pdf`.

[4] Dinesh Rajan, Anthony Canino, Jesus A. Izaguirre, and Douglas Thain. Converting a high performance application to an elastic cloud application. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 383–390, Washington, DC, USA, 2011. IEEE Computer Society.