Advanced Internet Computing
Project Topic 3 - Cloud Computing

Thomas Preißler, Tatiana Rybnikova, Simone Vecerka

29th January 2014

This paper describes the use of AWS EC2, a well-known Infrastructure as a Service provider, to handle the work of a sentimental classification. An algorithm for distribution the workload to various instances is developed as well as a decision algorithm to start and stop instances. Furthermore it is considered to move over the project to a Platform as a Service provider, namely Google App Engine.

Keywords: Cloud Computing, Infrastructure as a Service, Platform as a Service, Load Balancing

Introduction

This paper introduces a project which uses cloud computing techniques to do a sentimental classification of tweets. The focus lies on the work distribution of the classification and not on the classification itself.

The first part of the project uses various number of instances from Amazon EC2, a well-known Infrastructure as a Service provider, to distribute the classification work. The difficulty is to develop an algorithm which distributes the work load in a handy way with respect to the costs.

Secondly the classification system should be moved over to Google's App Engine, a Platform as a Service provider.

Cloud Computing

There are many definitions describing the term "Cloud Computing". The American National Institute of Standards and Technology provide their own, accepted definition:

"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." (Mell & Grance 2009)

As described by Vaquero et al. (2009) Cloud Computing can be distinguished into three different categories of services. Two services are used in this project and described in the following.

Infrastructure as a Service

A huge range of resources like storing and processing capacity is controlled by providers. By the use of virtualisation they distribute their resources to customers and can change the assigned resources quickly. Customers can use these resources very flexible, they usually use their own software stack.

Platform as a Service

Provider can create an additional layer upon their infrastructure. They use a small software stack and an execution environment to provide a software platform. Hardware resources are transparent for the customer and should be adjusted for the current resource consumption.

Employment of AWS EC2

As mentioned in the previous chapter, the full flexibility is available, which means full access to the operationAL?! system is available and there can be installed and used almost every software if desired. Therefore the disadvantage of the comprehensive flexibility is, to manage the whole system and software by everyone itself. There exists no automatic measurements or load balancing systems provided by the IaaS provider, it has to be implemented a load distribution system by itself.

In further steps it has to be defined, what is the reaction if one server cannot handle anymore the work which is intended for a particular server. There has to be an ongoing monitoring in place which checks various key indicators regularly and reports its measurements back to a central system which therefore decides if more or less servers are necessary than currently used.

Workload distribution

In the case of the tweet classification the focus of the decision algorithm has to lie on the cpu utilisation, because the bottleneck of the classification algorithm is the available CPU. But just using the current CPU utilisation of each instance as a decision basis for request distribution seems too easy. Every time a particular instance receives a new request the CPU utilisation raises up very quickly to about 100 percent and after finished the tweet classification it gets back to idling. The distribution algorithm may forward request to an instance only if it is in idling state. Decisions based on this mercurial indicator may result in a system which waits to finish for classification until new requests are forwarded to an instance. This behaviour results to instances which are idling sometimes and are just waiting for new requests, also when the overall system is under heavy load.

The algorithm may be improved if it assumes that each classification request fully utilises the CPU on the instance for a specific amount of time. After finishing the classification the utilisation goes down and a new request can be classified. The efficiency of the overall system can be improved if each instance is responsible for two requests at a time. The instance can immediately switch over to the second request after the first classification is finished.

This consideration results in the decision algorithm that requests are forwarded to instances depending on the currently executed classification of each instance. Instances with no classification job right now should be preferred when distributing a new request. Those instances are idling and wasting money. If there are no idling instances available new classification jobs should be processed by instances which have just one classification running right now. They may finish their currently running classification very soon and may drop back to idling if they don't receive a new request. But there should be no more than two classifications distributed to a particular instance at a time. This only results in longer time for finishing the classifications. To come up with short request peaks it may be necessary to forward more than two requests to a particular instance at a time. Instances with less concurrent requests should be always preferred. If instances receive more than two requests at a time for a longer period of time it is necessary to start new instances.

Decision to start and stop instances

As described in the previous section no instance should be responsible for more than two requests at a time. If there is a need to distribute three requests to at least one instance at a time it may indicate that the current amount of instances are not sufficient anymore to come by with the current request rate.

It looks obvious to start new instances if the requests increases the two-for-each-instance rate and stop an instance if it is idling. But this approach only works quite well if instances can be started immediately and are accounted per second. Unfortunately, both assumptions are not the case. Instances needs to start up to a couple of minutes and are accounted on an hourly basis. In conclusion there is a need for a different decision algorithm to start and stop instances.

As stated by Walker (2006) Unix and Linux systems report values, the so-called load average. This load average reports the number of processes which demands for the CPU. Due to not only provide and instantaneous snapshot the actual CPU utilisation the load average gives a trend of the prospective CPU requirement. This computation consists of three values, the 1-minute, 5-minute and 15-minute load average.

The load average value reports the number 0.00 for an idling system and 1.00 for a busy system. There are no limits for this numbers, but a load of 1.00 can be assumed as busy for the system.

To employ the load average value for the decision algorithm it has to be clearly defined when the overall system needs more or less instances than currently started. The load of the overall system can be calculated by the average of the load of each instance. As already mentioned a load of 1.00 indicates that a system is busy, therefore this number should be avoided for the overall system. As Brebner (2012) suggests, it seems quite ok to start new instances if the overall system reaches the load of 0.8. In contrast to stop instances the load should be decreased quite more than to 0.8 to overcome fluctuation request

rates and therefore load averages. Furthermore the stop of an instance should be as lazy as necessary to avoid the need to start a new instance quite short after the stop. It's necessary for future research about the best value, but for now the results of Brebner (2012) are considered, therefore instances will be stopped at the load of the overall system of 0.3.

Furthermore there are three values of the load average reported, one for the 1-minute, 5-minute and 15-minute time period. To add some additional laziness to the decision algorithm not the most fluctuating value, the 1-minute load average, is chosen, but the 5-minute load average. However some additional research in place is required to verify this decision.

In further consideration it may happen that there is a need for starting or stopping more than just one instance at a time. If there is a very fast increasing request rate there may be a need to start two or more instances simultaneously. To come by to this problem a simple load forecast can be employed. Each starting instance can be added to the load average with a load of 0.00 because this value is read quite short after the instance is ready for processing requests. If the load with this forecasted value in place is still more than the limit of 0.9 it is necessary to start additional instances.

A quite similar assumption can be done for stopping instances, but there is a load of 1.0 assumed for every stopping instance. This forecast meets the issue that as soon as the instance is stopped the load of the overall system increases again beyond its limit and a new instance is started quite short after another instance was stopped.

Migration to Google App Engine

The migration to a PaaS platform needs to be considered carefully. It has to be carefully checked, that the decreased flexibility of the new platform fits the requirements. But there may be also some advantages, such as less complexity of the application, because it is not longer necessary to take care about the load distribution and server starting and stopping anymore. Usually the PaaS provider offers some additional features such as a load-balanced database or services for sending huge amount of bulk emails.

For this project the migration to GAE was like pain in the ass. The Google App Engine requires some customisations for Applications. The most dramatic missed feature was the lack of full JPA support. Google App Engine only provides some very bascic JPA functions, but Joins are not supported. Furthermore it is required to create for all attributes of all order by clauses to create an index.

The aim to import the provided set of about 2.7 million tweets failed because of the very costly write operations to the data store.

In contrast to the deployment to AWS it is not necessary to implement any load balancing algorithms. The Google App Engine cares about the startup and shutdown of instances and forwards incoming request to the less demanded instance. Furthermore a task queue is offered which enables to easily store the classification task for a delayed execution. Backend instances pick up those tasks and are also started and stopped depending on the number of requested classifications.

Discussion / Conclusion

When it comes to a comparison between the Amazon EC2 and the Google App Engine it has to be noted that they offer different services. The Google App Engine as a Platform as a Service provider requires some customisations to fit an Application to its infrastructure, but offers a very fast initial deployment. There is no huge own knowledge required to deploy an application.

In contrast to the AWS EC2: It offers much more flexibility, because it doesn't offer any services and therefore it doesn't require customisations. But the customer has to handle many technologies by himself.

To say for this project Amazon EC2 fits better to the requirements than Google App Engine. The most crucial point for this decision is the very expensive write operations to the data store as well as the lack of many JPA features.

References

Brebner, P. (2012), 'Is your Cloud Elastic Enough? Performance Modeling the Elasticity of Infrastructure as a Service (IaaS) Cloud Applications', *ICPE'12* .

Mell, P. & Grance, T. (2009), 'The NIST Definition of Cloud Computing'.

Vaquero, L., Rodero-Merino, L., Caceres, J. & Lindner, M. (2009), 'A Break in the Clouds: Towards a Cloud Definition', *ACM SIGCOMM Computer Communication Review* .

Walker, R. (2006), 'Examining Load Average', *Linux Journal* .