Design and implementation of a social network infrastructure for designers of Multi-Cloud applications

Christos Papoulas

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Knossou Av., P.O. Box 2208, Heraklion, GR-71409, Greece

Thesis Advisors: Prof. *Kostas*, Dr. *Magoutis*

Optionally: This work was partially supported by **fill in sponsor name**

University of Crete
Computer Science Department

Your Title

Thesis submitted by
Author Name
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author:	Author Name
Committee approvals:	Name of first member Assistant Professor, Thesis Supervisor
	Name of second member Associate Professor, Committee Member
	Name of third member Professor, Committee Member
Departmental approval:	Name of Director of Graduate Studies Professor, Director of Graduate Studies

Heraklion, July 2015

Abstract

In this work we propose a Social Network Platform, both front-end and back-end technology, for model-driven software design and deployment of applications in Multi-cloud environments. Nowadays, DevOps users, especially cloud deployment specialists, wander around the web looking for automated tools like Chef supermarket, IBM bluemix and other deployment tools where they almost manually configure and deploy their applications without any community-sourced information repository. The users, inside of the proposed social network, can create their own Cloud Application Modelling and Execution Language models or benefit from automatically generated models. We provide a Platform with an integrated community helping the DevOps users to design and deploy their applications and a repository of execution histories of applications. We incentivize user to stay inside the Platform instead of roaming around the web looking for other Q&A sites like StackOverflow to find their answers. The runtime executions of applications can be uploaded and analysed by the Platform presenting cost effectiveness analysis. In addition, Natural language Processing tools are integrated with the platform in order to identify the context of users questions and provide automated answers. A scalable front-end is implemented using multiple front-end nodes and a powerful back-end system is implemented using caching technologies.

Περίληψη

Στην εργασία αυτή . . .

Acknowledgements

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this work, the design and implementation of a social networking platform for designers of Multi-Cloud applications is presented. In this targeted social networking platform, DevOps [1] and particularly cloud deployment specialists can benefit from other users' experience and answer design questions such as which is the most cost-effective deployment and which configuration fits their needs. DevOps consists of several types of specialists such as programmers, testers, QA staff and specialists from Operations who all use tools for various purposes like building applications, testing, deploying routines, configuration, automation of utilities, tracking and versioning in systems. Cloud Deployment Specialists are responsible for migration and configuration of hosted solutions for applications. Furthermore, they have deep industry knowledge about security, auto scaling, storage, load balancers, CDN and everything related to your application secured hosting and scalability. They talk to each other and exchange opinions on cloud deployment of applications, but the lack of a repository where they can find the configuration of an application that fits their needs.

This social network targets the cloud deployment specialists and binds all social networking concepts such as personal messaging, groups, new feeds with modelling-driven concepts of application composition and deployment, integrating a repository of cloud applications and infrastructure description based on Cloud Application Modelling and Execution Language (CAMEL) [2].

This CAMEL repository means to several benefits to the DevOps community. Among the range of possible DevOps tasks, the Social Network focuses on selecting the most appropriate deployment configuration for an application. This is especially challenging in a multi-cloud setting due to the large diversity of deployment possibilities and tradeoffs. Currently DevOps users work with a small set of well-understood deployment options, missing opportunities for improving performance, reliability and/or lower cost. Investigating new options involves timeconsuming testing over new infrastructures. Discussing with the community in online social or technical forums may provide insight over deployment options; however the answer to a hard question often needs to be backed by experimental data that is not readily available.

An integrated environment can enrich user interactions with structured references to applications and their components, execution data, and mined knowledge from real deployments. Mined knowledge can be combined with user activity and profiles to provide personalized suggestions and hints. An improved mode of user interaction is expected to result to stronger incentives for DevOps users to contribute information to the underlying repositories. More content should lead to better quality of mined knowledge, benefiting the DevOps community and providing further incentive for contributions. The social networking platform designed to be closely integrated with a set of information repositories satisfying the following requirements: (R1) handle entire applications rather than just software components; (R2) abstract application structure through software modeling; (R3) capture and analyze application runtime performance. Raising the level of abstraction from components to applications. The analysis of application execution data can provide answers to many interesting questions of the community and support discussions and arguments with hard data. These requirements can provide software developers with strong incentives to contribute, leading to the sustainability and growth of information and derived knowledge in the repository.

1.1 Background

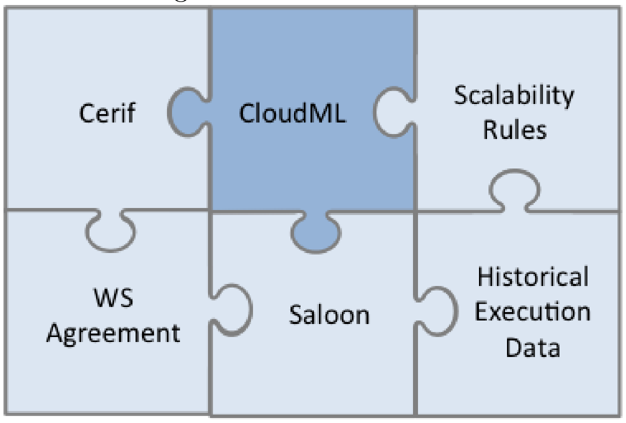
Application models inside social network platform are described in CAMEL. CAMEL integrates various domain-specific languages (DSLs). These DSLs cover a wealth of aspects of specification and execution of multi-cloud applications like CloudML, Scalability rules, WS Agreement, Saloon and Historical Execution Data. CAMEL is using the Eclipse Modelling Framework (EMF) on top of the Connected Data Objects (CDO). Applications is persisted on CDO repository. CloudML [3] is a recent approach that focuses on the provisioning and deployment of multi-cloud applications, is built upon MDE techniques and methods, and provides a models@run-time environment for enacting the provisioning. WS Agreement [4] is a Web Services protocol for establishing agreement between two parties, such as between a service provider and customer. Saloon [5] is an approach that uses models to represent clouds variability, as well as ontologies to describe the heterogeneous aspect of the cloud ecosystem. The CAMEL model couple together aAll those DSLs as shown in figure 1.1.

Chef . . .

1.2 Related Work

This section describes related work for other professional networks and their caching architecture. Additional an overview of and related work of Natural Language Processing is presented.

Figure 1.1: CAMEL DSLs.



1.2.1 Caching Application Data

Facebook, the largest social network, serves billions of requests per second using memcached [6]. In this magnitude of scale, Facebook has several pools of memcached servers (regional pools) around the globe. A request for a single page can produce hundred of requests to the back-end system. Memcached is used to store not only key-value from MySQL queries but also pre-computed results from sophisticated algorithms. In order to achieve a near real time communication experience to the end user, memcached servers have to be efficient, reducing latency.

The research question in such systems is when a particular key will be invalidated. This problem occurs according to [6] in two cases: (1) *stale sets* and (2) *thundering herds*. A stale set occurs when a web server sets a value to the memcached that does not reflect the real value of the database. Thundering herds occur when a specific key has a heavy read and write activity at the same time. Stale sets are resolved by an N-bit token, that is bound to a specific key and sent from the memcached to the web server that wants to update the key when a cache miss occurs. If a delete request is received, the request for updating this value from the client is rejected. The thundering herbs are solved by configuring the memcached servers to return an N-bit token only once every ten seconds per key.

Linkedin, the largest professional network, stores hundreds of terabytes of data to Project Voldemort [7], a key-value store, inspired by Amazon Dynamo [8]. Linkedin stores to Voldemort pre-computed offline data. For example, the results of data mining applications, such as features like “People You May Know”, that are running on hundreds of terabytes to make an estimation and are using Hadoop as the computational component of those estimations. Voldemort and Dynamo have the same following requirements: (1) a simple *get/put* application interface (2) A *replication* factor, the number of replicas for each key-value tuble, implemented using vector clock, (3) a *required read* factor to succeed a get request and (4) a *required write* factor to succeed a put request.

1.2.2 Professional Networks

IT professionals use a variety of online sources as aids in their daily tasks. Developers typically prefer community-moderated forums over vendor-moderated sites. Social networks focusing on software technology in particular provide developers with the opportunity to leverage the knowledge and expertise of their peers.

One of the most popular such platforms is GitHub [9], a collaborative revision control platform for developers launched in April 2008, and arguably the largest code-hosting site in the world. GitHub provides social networking functionality such as feeds, followers, wikis and a social network graph that captures how developers work on versions of their repositories, which version is newest, etc. Gitter [10] is a related service that facilitates discussions between members of GitHub communities by providing a long-term chat integrated with code and issues. Sourceforge [11] was the first code-hosting platform offered to open-source projects. It was launched in 1999 and offered IT professionals the ability to develop, download, review, and publish open-source software. Sourceforge is similar to GitHub in its support for social features. Other similar code-hosting platforms are Google Code [12] and Microsoft CodePlex [13]. None of those platforms collect, analyze, or use information from executions of application deployments to improve the level of technical discussion between users or abstract code structure through modelling or enhance user interactions through the use of analytics over application execution histories.

StackOverflow [14] advances on earlier Q&A sites in which users ask and answer questions. Users can vote up or down questions and answers and earn *reputation points* and *badges* in return for their active participation. Although StackOverflow and GitHub address different aspects of software development (StackOverflow is not a code-hosting platform) there is a synergy and correlation between the two [15]. The proposed social network platform extends StackOverflow through the use of social networking features that enable users interested in reasoning about application deployments to use and share knowledge drawn from analyses of information repositories.

IBM's BlueMix [16] is a development and support platform for communities of DevOps users wishing to compose distributed applications out of components drawn from libraries and deploy them at IBM-provided and supported cloud infrastructure. BlueMix is a key component of IBM's DevOps best practices [17] for achieving rapid prototyping, automated deployment, and continuous testing of software. BlueMix encourages its users to ask their questions to StackOverflow but also includes a community forum [16] with rating of answers contributes to eventually building a basic knowledge base, similar to traditional approaches such as StackOverflow. The proposed social network platform system differs from BlueMix in its support for expressing applications as models (CloudML, CAMEL) and its use of two information repositories, the PaaSage repository of models and execution histories and Chef supermarket, and the use of analytics over past executions to enable users to reason about application deployments. A common feature between the proposed social network platform and BlueMix is support for deployment of distributed applications.

LinkedIn widely adopted across a range of professional communities due to its robust set of social features (and to some extent due to its use of extensive analytics over collected information [18]), LinkedIn provides no specific support for software engineering activities and thus more closely resembles traditional social networking platforms such as Facebook.

The lack of Social Networking features of github came to fill the Geeklist platform [19], where developers and IT companies can discover and share the work they have done, connect with other companies in a social network manner or join development communities. Another code hosting platform is Snipplr [20], where developers can upload short code snippets but not full programs, in order to keep all of their frequently used code in one place that is accessible from any computer and any user. Masterbranch [21] is a new under development platform that allows collating and sharing of projects within a user's profile. This profile works similarly to LinkedIn and has an incentivisation scheme called DevScore, coupled with unlockable achievements that add a gamification element. Dzone [22] is essentially a link repository for developers allowing link sharing and incentivisation based on voting for the popular links. The Code project [23] website and forum allow code-specific discussion and share relevant articles and news, contains blogs, newsletter and a questions and answers section.

The above systems can be further classified based on whether they use a repository to store software-related information (code, models, configuration, or execution histories) and whether this information is shared and raised through crowd sourcing [24]. GitHub, GoogleCode, CodePlex, SourceForge, BlueMix, Chef Supermarket, and our platform store at least one type of software-related information and all systems but BlueMix are raising shared content in their software-related repositories via crowd sourcing. Our professional network is the only solution that analyzes information in its software-related repositories to assist users with suggestions and hints.

1.2.3 Natural Language Processing

The research questions are:

- The platform identities the similarity of a given question with the others questions already posted in the system.
- The platform maps a question with a relevant query to the repository providing to the one who asks with an appropriate response.
- The platform can paraphrase the queries according to the user's arbitrary input in order to meet the previous objective.

. . .

Table 1.1: Feature comparison.

	User Interaction				Repository						
	Social features [a]	Groups	Q & A	Personal messaging	Software code	Software models	Software config	Execution histories	Crowd sourced	Repo assisted hints [b]	Application deployment
GitHub	✓	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗
Sourceforge	✓	✗	✓	✓	✓	✗	✗	✗	✓	✗	✗
GoogleCode	✗	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗
CodePlex	✓	✗	✓	✓	✓	✗	✗	✗	✓	✗	✗
StackOverflow	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
BlueMix	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✓
Chef Supermarket	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗
LinkedIn	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
geeklist	✓	✓	✗	✗	✗	✗	✗	✗		✗	✗
Snipplr	✗	✗	✗	✗	✓	✗	✗	✗		✗	✗
Masterbranch	✗	✗	✗	✗	✓	✗	✗	✗		✗	✗
Dzone	✓	✗	✗	✗	✓	✗	✗	✗		✗	✗
codeproject	✓	✗	✓	✗	✓	✗	✗	✗		✗	✗
PaaSage SN	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓

[a] Features: follow and news feed

[b] User assistance based on data analysis of the repository

Chapter 2

Implementation

This section describes the implementation of social network site and how the system scales.

The system is composed by the following components, as shown in figure 2.1: At the first layer lives (1) the Social Networking engine, which runs all PHP scripts and described in section 2.1. At the second layer lives (2) the Memcached caching system, which described in section 2.3. At the third layer lives (3) the Social Network MySQL database, and (4) the CDO server - client components and the CDO repository. The Social Networking Engine at layer 1 is considered as the front end system and the CDO Client, the memcached nodes, the CDO Server and the repositories are considered as the back end system.

Achieving the scalability of the system, two system architectures are examined at two layers of the system: (1) We added more than one Social Network engine at the first layer of the system. In this implementation, in order to keep the file system in consistent mode we integraded Apache Zookeeper[25]. (2) We added more than one memcached machines at the second layer in order to add more cpu capacity and improve the system response time.

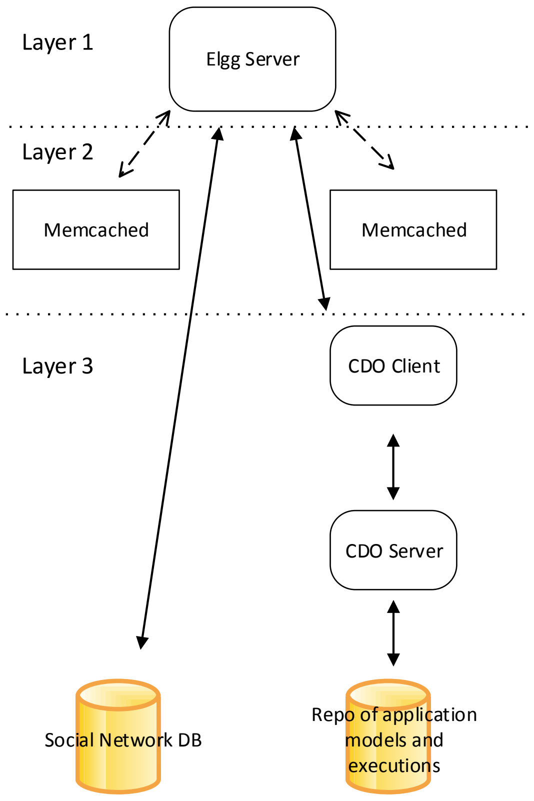
2.1 Implementation of Social Network

The social networking platform is implemented over the extensible Elgg social network framework[26]. Elgg is open source software written in PHP, uses MySQL for data persistence and supports jQuery [27] for client-side scripting. The architecture of Elgg Social Network shown in figure 2.3. The Model of the framework is structured around the following key concepts as shown in figure 2.2:

- *Entities*, classes capturing social networking concepts: users, communities, application models. Elgg Core comes with four basic objects: ElggObject, ElggUser, ElggGroup, ElggSite, ElggSession, ElggCache and a lot of other classes necessary for the proper engine operation.

- *Metadata* describing and extending entities (e.g., a response to a question, a

Figure 2.1: The overall architecture of Social Network.



review of an application model, etc.).

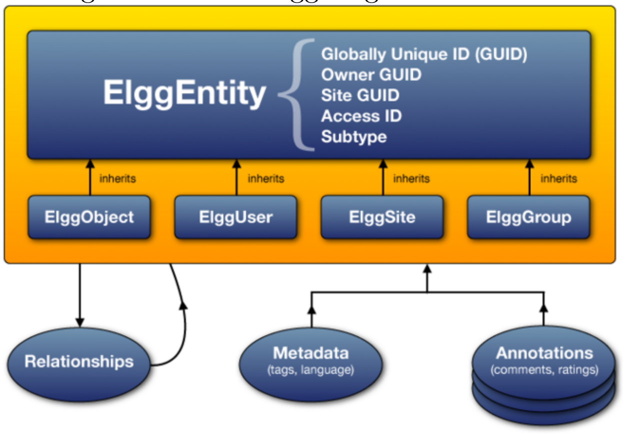
- *Relationships* connect two entities (e.g., user A is a friend of user B, user C is a contributor to an application model, etc.) and are persisted in the Social Network DB.

- *Annotations* are pieces of simple data attached to an entity that allow users to leave ratings, or other relevant feedback.

All Elgg objects inherit from ElggEntity, which provides the general attributes of an object. Elgg core comes with the following basic entities: ElggObject, ElggUser, ElggGroup, ElggSite, ElggSession, ElggCache, as well as other classes necessary for the operation of the engine.

Elgg comprises a core system that can be extended through plugins (examples are the Cart system or the handling of Application Models). Plugins add new functionality, can customize aspects of the Elgg engine, or change the representation of pages. A plugin can create new objects (e.g., ApplicationObject) characterized

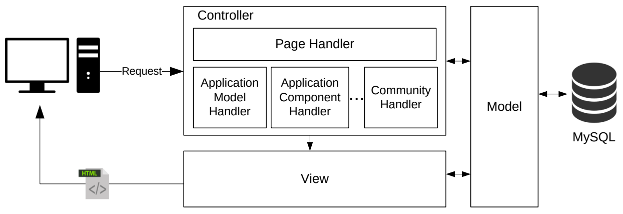
Figure 2.2: The Elgg Engine Data model.



(through inheritance of ElggEntity) by a numeric globally unique identifier (GUID), owner GUID, Access ID. Access ID encodes permissions ensuring that when a page requests data it does not touch data the current user does not have permissions on.

Figure2.3 shows the model, view, and control parts of Elgg's architecture. In a typical scenario, a web client requests an HTML page (e.g., the description of an application model). The request arrives at the *Controller*, which confirms that the application exists and instructs *Model* to increase the view counter on the application model object. The controller dispatches the request to the appropriate handler (e.g., application model, component handler, community handler) which then turns the request to the view system. View pulls the information about the application model and creates the HTML page returned to the web client.

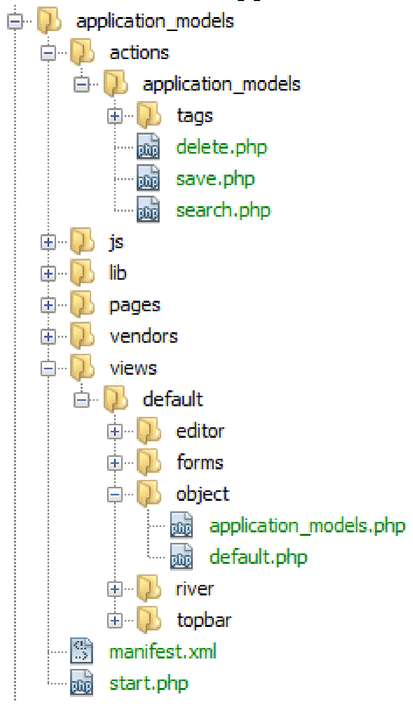
Figure 2.3: Architecture of the Elgg Social Networking engine.



The extensibility of Elgg can be established not by modifying the core system but by introducing new plug-ins which follow the MVC model. A new plug-in can create a new entity. Thus, each entity is characterized by a numeric Globally Unique Identifier and Access ID. The Access ID determines the permissions that other users have. Thus, when a page requests data, it never touches those data that the current user does not have permission to see. All plug-ins share a common structure of folders and PHP files, following the MVC model of figure 2.3.

The hierarchy of a plug-in is shown in figure 2.4. Folder *actions* includes the actions applied on application models. Every active participation by the user is performed via an action. Logging in, creating, updating or deleting content are all generic categories of actions. The *views* folder contains the *php* forms applied on application models, *river* events (Elgg terminology for live feeds). Viewss are responsible for creating the output for the client browser. Generally, this will be HTML, but it can be also JSON or other format. *Pages* overrides elements of core Elgg pages and can be from chunks of presentation output (like sidebars) down to individual html code. The *js* and *lib* folder provides javascript and *php* library functions. Finally, the *vendors* folders include third-party frameworks such as Twitter's bootstrap front-end [28]. The most important file of a plug-in is the *start.php* script, which contains the *page handler*. Page handler is a function manages the plug-in pages enabling custom url redirect to a specific page. The plug-in initialization is also defined in the start.php and registers actions, events and determines the views.

Figure 2.4: The structure of the application description plug-in.



```
application_models
    actions
        application_models
            tags
            delete.php
            save.php
            search.php
    js
    lib
    pages
    vendors
    views
        default
            editor
            forms
            object
                application_models.php
                default.php
            river
            topbar
    manifest.xml
    start.php
```

The execution history of deployments of application models and the description of those models is stored in the CAMEL information repository, which is implemented as an Eclipse CDO server. The exchange of information between Elgg and the CAMEL information repository is going through CDO Client who retrieves the information from CDO server and sent it to the Elgg over sockets.

2.1.1 Scaling Social Network Engine

This subsection describes how the horizontal scale of Social Network engine achieved. At the layer 1 of figure 2.3 lives the Apache2 server which as the stress test of the system shows in chapter 3, the Apache2 server takes a heavy load on CPU utilization. Therefore, we introduce more than one Apache2 Servers running the Social Networking Engine of Elgg Framework.

The Social Networking Engine keeps some information in the file system instead of the Social Network DB. This information includes the profile photos of users and any other photo such as photos that users add to the community groups. Furthermore, the initial configuration of Social Networking Engine keeps in the file system some caching files representing some views which are independent from specific users and do not change among all users. This file system caching feature removed from the Social Network Engine because is more efficient to use memcached for the caching instead of the slow file system.

The Network File System (NFS) is used in order to all SN Engines have access to the same file system store. An NFS server installed in one of the SN Engines and all the other SN Engines have an NFS client accessing the remote file system.

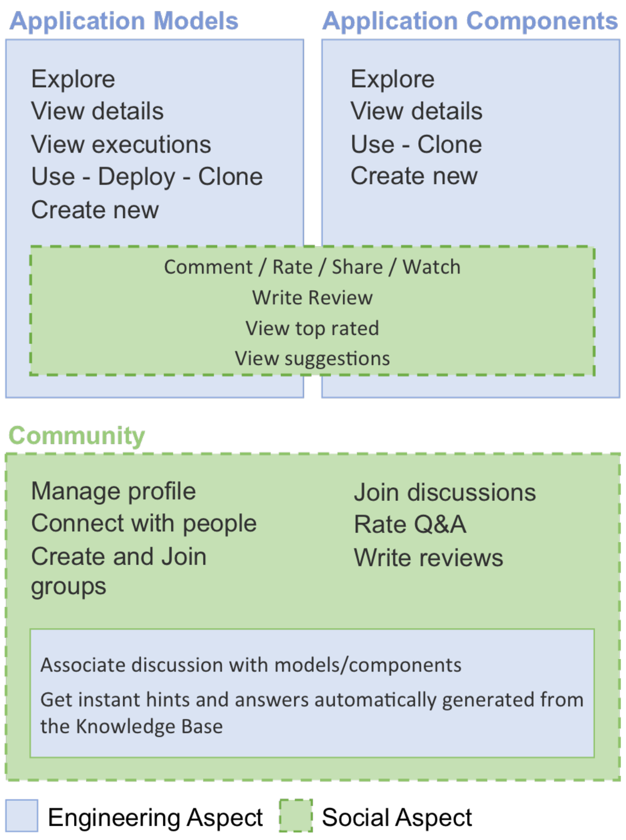
Distributing Social Network Engine was not an easy problem, so Apache ZooKeeper [25] is used to enable highly reliable distributed coordination among access to the file system storing by Social Networking Engine. Apache ZooKeeper provides a tree abstraction where every node in that tree (or znode) is a file on which a variety of simple operations can be performed. ZooKeeper orders operations on znodes so that they occur atomically. Therefore there is no need to use complex locking protocols to ensure that only one process can access a znode at a time. The tree represents a hierarchical namespace, so that many distinct distributed systems can use a single ZooKeeper instance without worrying about their files having the same name.

Social Networking Engine uses Apache ZooKeeper in order to keep consistent the file system in rear but possible scenarios such as two users try to upload in the same time a file to the same group. When a SN Engine wants to write a file in file system first locks the specific path and after finish the write operation releases the lock.

2.2 User interface

In this chapter described the User Interface of Social Network implementated based on 104 mock-ups created by HCI expert team. The key design objective of the social network platform is to create a strong bond between (i) software engineering services for managing and deploying cloud-targeted application models; and (ii) community-oriented facilities for communication and collaboration between users. The interconnections between the two in the design of the user interface are depicted in Figure 2.5. The prototype implementation is publicly accessible on-line at http://socialnetwork.paasage.eu.

Figure 2.5: The engineering & social activities are seamlessly within the Platform.



2.2.1 User Interface Design

The discrete entities, which bound together the Social Networking with model aspects of Platform are:

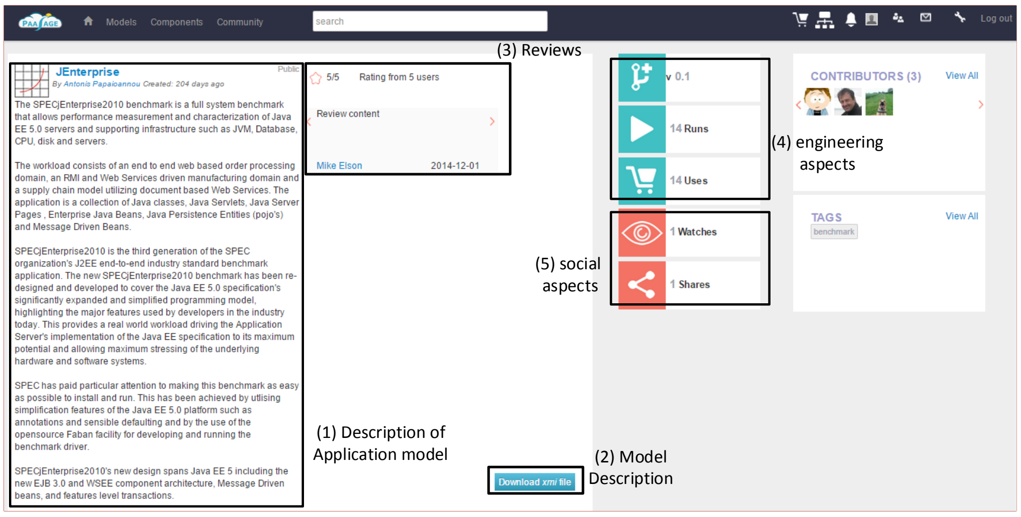
- *Application Models*. An example is shown if figure 2.6, consisting of a human friendly description (label 1 in fig.2.6), the Camel Description of the model (label 2 in fig.2.6), reviews about the model (label 3 in fig.2.6). An overview of engineering aspects such as version and runs (label 4 in fig.2.6) and an overview of social aspects such as share and watches (label 5 in fig.2.6)

- *Components*. We have integrated the Chef supermarket components into Social Network Platform. The components help the DevOps users to generate their application models.

- *Groups*.

- *Users*.

Figure 2.6: The application model home page



The SPECjEnterprise2010 benchmark is a full system benchmark that allows performance measurement and characterization of Java EE 5.0 servers and supporting infrastructure such as JVM, Database, CPU, disk and servers.

The workload consists of an end to end web based order processing domain, an RMI and Web Services driven manufacturing domain and a supply chain model utilizing document based Web Services. The application is a collection of Java classes, Java Servlets, Java Server Pages , Enterprise Java Beans, Java Persistence Entities (pojo's) and Message Driven Beans.

SPECjEnterprise2010 is the third generation of the SPEC organization's J2EE end-to-end industry standard benchmark application. The new SPECjEnterprise2010 benchmark has been re-designed and developed to cover the Java EE 5.0 specification's significantly expanded and simplified programming model, highlighting the major features used by developers in the industry today. This provides a real world workload driving the Application Server's implementation of the Java EE specification to its maximum potential and allowing maximum stressing of the underlying hardware and software systems.

SPEC has paid particular attention to making this benchmark as easy as possible to install and run. This has been achieved by utlising simplification features of the Java EE 5.0 platform such as annotations and sensible defaulting and by the use of the opensource Faban facility for developing and running the benchmark driver.

SPECjEnterprise2010's new design spans Java EE 5 including the new EJB 3.0 and WSEE component architecture, Message Driven beans, and features level transactions.

Gamification

Following recent trends in social networks design and with the aim to motivate users active and regular participation in the professional network, the design employs gamification features, namely use of video game elements to improve user experience and user engagement in non-game services and applications [29]. One gamification feature in the Social Network design is the reward system for active community members. As users contribute content (models, components, ratings, reviews, questions, or answers) they receive experience points leading to special badges visible to all community members. Other features are the Profile completeness bar with suggestions on how to increase it. Finally, the concept of Model badges awarded to application and component models in case of excelling performance. Badges can serve among others as goal-setting devices, status symbols, and indications of reputation assessment procedures [30].

In order to support those look & feel and the functionality of those mock-ups 25K lines of php, js and css code is written.

2.3 Memcache

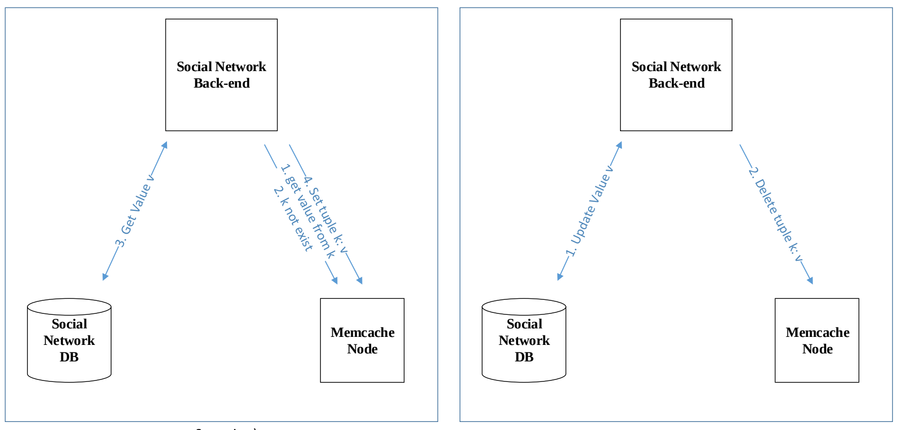
This section describes the experience gained by using memcached[31]. Memcached is an open source, high-performance, distributed memory object caching system. We choose memcached, because is a generic simple in-memory key-value store. It has a powerful API available for PHP. After memcached integration the system increase the response time and performance.

Memcached added in layer 2 of system architecture and used for storing the following key-value tuples: (1) values from Social Network Database such as entities of Social Network, applications, components, users, group discussions, (2) evaluated javascript code results and (3) executions histories from repository of application models. Storing the executions of applications at Memcached the response time of the system increased because the PHP modules do not need to go through the heavy CDO client but get directly the executions of applications from Memcached.

The apache jmeter[32] was used to measure the responce time of the system and the sysstat tool[33] was used to measure the cpu usage. Section 3.1 shows the performance results of this implementation.

All tuples at Memcached are inserted with maximum key expiration time of thirty days. When a value in social network is updated, the memcached key will deleted 2.7.

Figure 2.7: The scenario a depicts a request from memcached when the key does not exist and scenario b depicts a updated operation of a value)



Scenario a)
(1) Request a key k from Memcached node, (2) the k not found in memcached and the SN back-end (3) requests the value v from SN DB. When the SN back-end receive the v, (4) stores it to memcached node

Scenario b)
When (1) a value updated to SN DB, (2)the corresponding memcached key will invalidated.

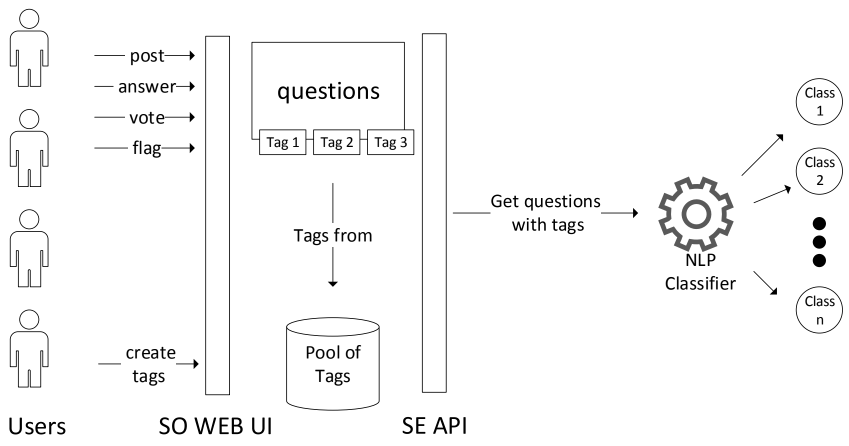
2.4 Natural Language Processing and classification

Natural Language Processing(NLP) [34] is a feature added to the Social Network. NLP is used in the interactions between the users and the SN and in the way the platform can understand and determine the type of user input. Particularly, Natural Language Understanding using Naivy Bayes algorithm is added to SN. The Natural framework [35] implemented with node.js. In general, Machine Learning algorithms such as Naivy Bayes, want an input data set of training data. This data is pulled from StackOverflow (SO) questions. Those questions are an excellent repository to train the NB algorithm.

In general, the main actions of SO members is shown in figure 2.8. When users ask questions in SO, they must specify some tags describing their questions. A tag is a keyword or label that categorizes their question with other, similar questions. The users of SO sometimes, try to add as much tags as possible in order to make their questions popular and get them answered. SO restricts the users to add up to five tags in each of their questions. After a question is posed, the community of SO can vote up or down the question or privileged users can flagged the question as *duplicate*, *off-topic*, *unclear*, *too board*, *primarily opinion-based*. So low quality questions will can be removed from the site and keep the questions repository clear and helpful to other potential DevOps.

For the training sets, which is the data to train the NB algorithm, the most voted question of the SO community is used. Those questions have emerged as the good questions in their fields and surely, we avoid the case to use a training set with miss-tagged questions which would result a miss-guided NLP classification. The NLP training set is retrieved from SO site using the stack exchange(SE) API [36]. The SE API is a powerful API, which allows to take the questions, the answers, the users and all the information that exists in SO site through a programming interface.

Figure 2.8: The main StackOverflow users' actions and NLP Classifier.



The first training set of our Natural Processing Tool consisted of five tags, relative to our platform. Those tags were: *scalability*, *reliability*, *design*, *performance* and *optimization* with thirty questions per tag. For each of the tags, the thirty most voted questions from StackOverflow were retrieved and classified to each specific tag. Those exactly tags, after classification, are transformed in classes in NLP classification, as shows the figure 2.8

Every time a user asks a question to a platform's community, the classifier determines the class of the question. Then, if the platform is able to determine a heuristic answer, it will post it to the user's question. All the users of the platform can vote up or down this answer, depending on its accuracy, or provide their own answers.

The second training set of Natural Processing Tool was retrieved by automatically discovering tags. NB is trained with 10K questions from StackOverflow Q&A site, fifty questions per tag. The general algorithm is shown below. Firstly, the algorithm starts with a tag which is relevant to the Social Network Platform such as *scalability* at line 01. Afterwards, using the SE API the algorithm gets the 5 most voted questions tagged with *scalability*. For each question (line 04), the populateClassifier clears the body from any html tags inserted by the StackOverflow users to beatify their questions (line 05) and classifies this question's body with each tag. Automatically, the populateClassifier proceeds to the next tag of this question. When the populateClassifier is finished the NB is trained. It should be noted that a question may have more than one tag, so a question can be classified up to five tags / classes. Changing the threshold parameter at the following algorithm, the *populateClassifier* can classified with an arbitrary number of tags. At the following section the process to Bayes classification is described in more details.

```
01:var tags = ['scalability']
02:populateClassifier(0)
03:function populateClassifier(index) {
04:   var questions = stackexchange.api.getQuestionsByTag(tags[index])
05:   foreach(questions as q)
06:      body = clear(q.body)
07:      foreach(q.tags as t)
08:         classify(body, t)
09:         if(not tags.exist(t) and tags.length() < threshold)
10:            tags.push(t)
11:            populateClassifier(tags.indexOf(t))
12:}
```

Bayes Classification Algorithm

In this section the Bayes classification algorithm is being described. As the above code snippet show at line 08, the algorithm classifies a document named *body* into the class *t*. Diving in this function, the *body* transformed to lower case and the Porter Stemming Algorithm [37] is used for suffix stripping, so the plural part of the words and the *-ing* and *s* parts of verbs are removed. Thus the following words: *connected*, *connection*, *connections*, *connecting* will all be transformed to the single word "connect". The Porter Stemming Algorithm is not using any dictionary but a simple list of suffixes which makes the algorithm fast (10.000 different words in 8.1 seconds). After this process the table of words of this *body* is kept.

After the populateClassifier has finished, the *trainClassifier* is called (for simplicity not shown in the above code snippet). The work of *trainClassifier* is to make the Bayes Classification ready for process and determine the class of other future documents. So, *trainClassifier* counts for each word the number of occurrences for each class.

Since the Classifier is ready, when a future request for classification comes, the Classifier returns for each class the probability to be part of this class. The probability of each class is calculated by following formula:

$$prob(d/c) = log\left(\frac{countedTerms(d,c)}{totalsTerms(c)}\right)$$

Where the probability of a document d to be a class c is the logarithmic value of the division of the words(terms) of d found in class c by the total number of terms in c.

Chapter 3

Evaluation

This chapter describes the evaluation of the two different implementations of the system architecture. In the first architecture, more than one memcached instances at layer 2 were introduced, as figure 2.1 shows. In the second, more than one Social Network engines were introduced at layer 1.

In order to measure the response time (RT) the Apache JMeter application [32] is used. The Apache JMeter is an open source benchmark designed to test functional behaviour and measure performance, targeting web applications. Notably, the RT measured by JMeter may not be the real one, because the JMeter measures the elapsed time from just before sending the request to just after the last response from the server has been received. As a result, the time to render the web page to the client web browser and the execution time of JavaScript code is not measured. Because those two time intervals are client limited and depend on client performance and on which web browser is used, they are excluded from the following performance test benches. For the next experiments, a specific web page will be used. This page does not use any AJAX call, in order to not misguide the results. Therefore, the RT measures the time from just before JMeter sends the request to just after the last response is received. During this measured time interval, the Social Network engine performs the following actions:

I The Social Network engine sends a request to CDO Client for the application execution model.

II The CDO Client forwards this request to CDO Server.

III Afterwards CDO Server queries the mysql repository of application models and executions, and finally gets the executions results.

IV CDO Server forwards the results through the CDO Client to the Social Network Engine.

V Finally, the Social Network engine sends queries to the Social Network DB in order to get all the necessary Social features for this application page

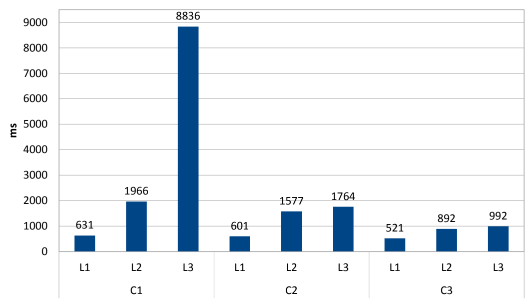
3.1 Improving Performance with memcached

By adding a memcached node at the system architecture, the Social Network Engine first asks the memcached node if it has the tuples that the SN Engine needs. So the steps(I to V), mentioned previously, are not necessary if the memcached node has cached the values that the Social Network Engine needs. The loop through CDO CIient - CDO Server and the repositories is bypassed.

For the following experiments all the memcached nodes are warm up and have cached all the needed CDO and Social entities information. Furthermore the CDO server has been warm up after a fresh restart. As the table 3.1 shows the starting process of the CDO server produces 1938 queries to MySQL database. The *fresh query* for an application model (both social information and executions) produces 15182 queries to MySQL database. The CDO server caches the results, so a second query for this application model produces 251 queries, which the most of them are the queries for the social information of the application. Introducing memcached, if the request for the application model is cached, the queries to database are lowering to 147.

The test performed with the following loads: (L1) ten users requests *two* applications, (L2) ten users requests *four* applications and (L3) ten users requests *eight* applications. All three Loads run consecutively one hundred times each. Those Loads request applications, which have ten execution rows pulled from the repository of applications models and executions, and about one hundred queries to the Social Network DB. In this experiment we kept constant the following components of the system: the Elgg front-end Apache2 server, the Social network BD, and the CDO server - client communication but increased the number of memcached nodes. The figure 3.1 shows the average, minimum and maximum response time (RT) in milliseconds with the following system configuration: (C1) no memcached node, (C2) one memcached node and (C3) two memcached nodes.

Figure 3.1: The average response time for all configurations.



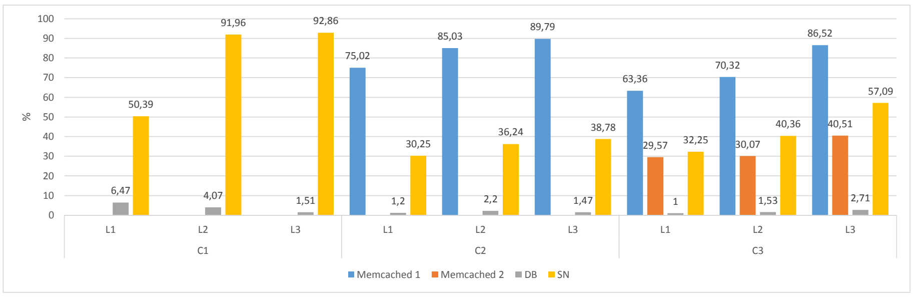
As we going from C1 to C3 and specifically for L(oad) 3, the RT is reduced by

Table 3.1: Number of Queries to Social Network and CDO server Databases.

State	# of Queries
Fresh start	1938
Fresh Query	15182
Cached Query from CDO	251
Cached Query from Memcached	147

80,4% at C2 and by 88,78% at C3. As the figure 3.1 shows, at the first configuration C1, the L3 takes 8836 ms, an RT which is definitely prohibitive for web applications. Introducing more memcached nodes at C2 and C3 the RT is decreased dramatically at 1764 ms at L2 and at 992 ms at L3. Going from C1 to C2, the 80,4% reduction of RT is due to the introduction of memcached node and bypassed the steps I - V. Going from C2 to C3, the 43,77% reduction of RT is due to adding more memcached nodes, resulting to more cpu cores introduced to the architecture.

Figure 3.2: The average CPU utilization for all components.



Furthermore, the CPU utilization is measured using the sysstat tool [33]. We measured the CPU utilization for all the VMs running the experiment. The information about the VM resources is listed in the table 3.1. The Social Network Engine and the CDO Client were running at t1.micro instance. The mysql (repositories) and the CDO Server were running at m1.xlarge. The average CPU utilization is shown in the figure 3.2. At the simple configuration C1, even in small loads such as L1, the SN Engine reached 50,39% CPU utilization. In the medium load L2 and big load L3 the SN Engine is kneeled down to 91,96% and 92,86%. This big consumption of CPU was due to all the initialization that Elgg Social Network Engine has to do for each request and due to CDO Server queries.

Moving from configuration C1 to C2, the CPU consumption went to memcached node. Thus, the Social Network engine was de-congested and the RT improved.

Table 3.2: VM resources

Component	VM type
SN engine, CDO client	t1.micro
memcached	t1.micro
repositories, CDO Server	m1.xlarge
jmeter	m1.large

However, for the big load L3 the memcached node reached 89,79%. To solve memcached CPU overhead, one more memcached node was added at configuration C3. This second memcached node shared the CPU overhead with the first memcached node and the RT improved furthermore. For all three loads at C3, the first memcached node has more CPU utilization from the second by an approximately factor of 2,2. This difference between the two memcached nodes appeared due to the first node storing more popular key-value pairs than the other.

3.2 Improving Performance with engine

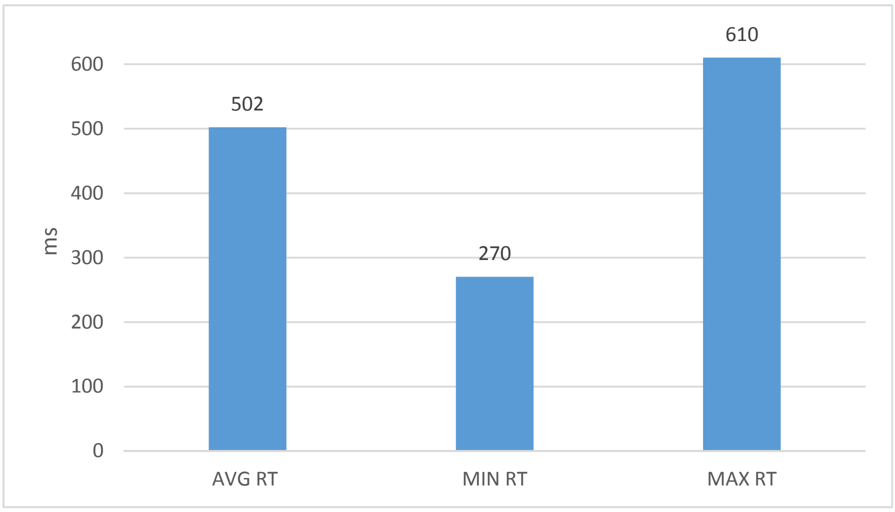
This section evaluates the horizontal scale of Social Network engine as described in 2.1.1. A memcached node was living between the Social Networking Engines and the back end system. The VM resources are kept the same in the previous experiment, shown in the table 3.1. One more Social Networking Engine instance was added with the same type as former SN Engine. The load from one SN Engine is now distributed to two SN Engines instances. So, the response time improved, as shown in the figure 3.3 for the Load 3 compared to the previous test-bench.

The CPU utilization to SN Engines decreased as shown in the figure 3.4. This reduction is due to the requests being distributed to two instances instead of only one. The CPU utilization of the memcached node increased, but this can be solved by introducing more memcached nodes as the previous section describes.

3.3 Evaluation of NLP classification

This section evaluates the Natural Language Processing Tool as described in section 2.4 for the first training set. This set had five different classes or, according to StackOverflow dialect, five tags, as described in 2.4 and shown in table 3.3. For the testing set, thirty questions from StackOverflow were used per class. Those questions were different from the training set and had the highest activity during that specific time period. Each row at table 3.3 shows a class as classified by StackOverflow users, and each column shows how our classifier classifies the question. For example, twenty one questions about *reliability* were classified correctly, but eight were wrongly classified as *optimization* and one was wrongly classified as *performance*. The misclassification was not an error of our classifier but some questions from the testing set were either wrongly classified by the StackOverflow

Figure 3.3: The Response time for two Social Network Engines.



users or had more than one tags. For example, one question that was wrongly classified as *performance* instead of *reliability* was: “can somebody explain me how to handle errors. my code is: ...”. The above question is out of the scope of reliability and even though the user tagged it as a reliability question, it was downvoted and marked as “very low quality” from the StackOverflow community.

Another example showing that the classifier is not misclassifying is the following question: “I want to perform some data manipulation tasks and analysis in spark and want to **optimize** the run times. here is the problem: ...”. This question was marked by the user with both scalability and optimization tags. Even though this question is retrieved with the scalability tag and in the table it is shown as a wrong classification, after examining this question, one can realize that the *scalability* tag was wrongly added by the user and only the *optimization* tag should be placed.

This shows that our tool can further be used by StackOverflow to mark new questions that are wrongly tagged or misguided. There is a trend among StackOverflow users to add as many tags as they can in order to attract the attention of other users, increase the views of their question and finally get their answers.

Figure 3.4: The CPU utilization for two Social Network Engines.

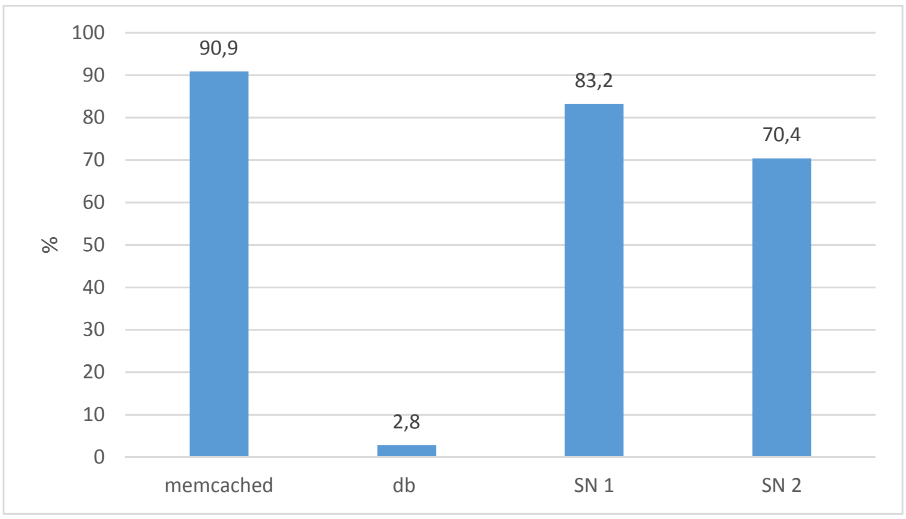


Table 3.3: NLP Evaluation of Classification

class / class	reliability	design	optimazation	performance	scalability
reliability	**21**	0	8	1	0
design	2	**15**	8	3	2
optimazation	2	3	**15**	9	1
performance	2	1	17	**9**	1
scalability	10	6	3	3	**8**

Chapter 4

Conclusions and Future Work

In this thesis a scalable Social Network for DevOps users is presented. The scalining of the system presented at the front end layer by introducing more than one Social Networking Engine instances, or at the back end of the system by introducing more than one memcached nodes. The DevOps users of the Social Network can benefit from the ...

Bibliography

[1] M. Loukides, *What is DevOps?* " O'Reilly Media, Inc.", 2012.

[2] P. D. 2.1.2, "Model Based Cloud Platform Upperware," http://www.paasage.eu/images/documents/paasage_d2.1.2_final.pdf, 2014.

[3] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Proceedings of CLOUD 2013: 6th IEEE International Conference on Cloud Computing*, L. O'Conner, Ed. IEEE Computer Society, 2013, pp. 887–894.

[4] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification (ws-agreement)," in *Open Grid Forum*, vol. 128, 2007, p. 216.

[5] C. Quinton, N. Haderer, R. Rouvoy, and L. Duchien, "Towards multi-cloud configurations using feature models and ontologies," in *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds.* ACM, 2013, pp. 21–26.

[6] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab *et al.*, "Scaling memcache at facebook." in *nsdi*, vol. 13, 2013, pp. 385–398.

[7] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah, "Serving large-scale batch computed data with project voldemort," in *Proceedings of the 10th USENIX conference on File and Storage Technologies.* USENIX Association, 2012, pp. 18–18.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[9] "Github," http://github.com/, [Online; accessed 24-May-2015].

[10] "Gitter: The chat for github," http://gitter.im/, [Online; accessed 24-May-2015].

[11] "Sourceforge," http://sourceforge.net, [Online; accessed 24-May-2015].

[12] "Google code," https://code.google.com, [Online; accessed 24-May-2015].

[13] "Codeplex: Project hosting for open source software," https://www.codeplex.com, [Online; accessed 24-May-2015].

[14] "Stackoverflow," http://stackoverflow.com/, [Online; accessed 24-May-2015].

[15] B. Vasilescu, V. Filkov, and A. Serebrenik, "Stackoverflow and github: Associations between software development and crowdsourced knowledge," in *Social Computing (SocialCom), 2013 International Conference on*, Sept 2013, pp. 188–195.

[16] "Ibm blue mix for developers," http://https://developer.ibm.com/bluemix/, [Online; accessed 24-May-2015].

[17] "Ibm devops best practices," http://www.ibm.com/developerworks/devops/practices.html, [Online; accessed 24-May-2015].

[18] R. Sumbaly, J. Kreps, and S. Shah, "The big data ecosystem at linkedin," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 1125–1134.

[19] "Geeklist," https://geekli.st/, [Online; accessed 10-June-2015].

[20] "Snipplr," http://snipplr.com/, [Online; accessed 10-June-2015].

[21] "Masterbranch," https://masterbranch.com/, [Online; accessed 10-June-2015].

[22] "Dzone," http://www.dzone.com/, [Online; accessed 10-June-2015].

[23] "The code project," http://www.codeproject.com/, [Online; accessed 10-June-2015].

[24] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.

[25] A. Zookeeper, "Apache zookeeper," https://zookeeper.apache.org/, 2015, [Online; accessed 20-May-2015].

[26] E. S. N. Engine, "Elgg social networking engine," http://elgg.org/, 2015, [Online; accessed 19-May-2015].

[27] jQuery, "jquery," https://jquery.com/, 2015, [Online; accessed 19-May-2015].

[28] "Bootstrap front-end framework," http://getbootstrap.com/2.3.2/, [Online; accessed 24-May-2015].

[29] S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon, "Gamification. using game-design elements in non-gaming contexts," in *CHI'11 Extended Abstracts on Human Factors in Computing Systems.* ACM, 2011, pp. 2425–2428.

[30] J. Antin and E. F. Churchill, "Badges in social media: A social psychological perspective," in *CHI 2011 Gamification Workshop Proceedings (Vancouver, BC, Canada, 2011)*, 2011.

[31] Memcache, "Memcache," http://memcached.org/, 2015, [Online; accessed 18-May-2015].

[32] A. jMeter, "Apache jmeter," http://jmeter.apache.org/, 2015, [Online; accessed 19-May-2015].

[33] sysstat, "Performance monitoring tools for linux," https://github.com/sysstat/sysstat, 2015, [Online; accessed 19-May-2015].

[34] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing.* MIT press, 1999.

[35] "Natural language processing with node.js," https://github.com/NaturalNode/natural/, [Online; accessed 1-July-2015].

[36] "Stack exchange application programming interface," https://api.stackexchange.com/, [Online; accessed 14-July-2015].

[37] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.