

# 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*



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 ...



## Περίληψη

Στην εργασία αυτή ...





## Acknowledgements

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Motivation . . . . .	4
1.2	Background . . . . .	4
1.3	Methodology . . . . .	4
1.4	Other section . . . . .	4
1.5	Related Work . . . . .	4
1.5.1	Professional Networks . . . . .	4
1.5.2	Memcached . . . . .	4
<b>2</b>	<b>Management of ...</b>	<b>5</b>
2.1	AA . . . . .	5
2.2	BB . . . . .	5
2.3	CC . . . . .	5
2.4	DD . . . . .	5
<b>3</b>	<b>Implementation</b>	<b>7</b>
3.1	Implementation of Social Network . . . . .	7
3.2	Memcache . . . . .	9
<b>4</b>	<b>Evaluation</b>	<b>11</b>
4.1	Improving Performance with memcached . . . . .	11
4.2	Improving Performance with engine . . . . .	11
<b>5</b>	<b>Comparison</b>	<b>13</b>
5.1	AA . . . . .	13
5.2	BB . . . . .	13
<b>6</b>	<b>Conclusions and Future Work</b>	<b>15</b>



# List of Figures

3.1	The overall architecture of Social Network. . . . .	8
3.2	Architecture of the Elgg Social Networking engine. . . . .	9



# List of Tables



# Chapter 1

## Introduction

In this work proposed the design and implementation of a social networking platform for designers of Multi-Cloud applications. In this target specific social networking platform, DevOps can benefit from other users experience and answer design questions such as which is the most cost-effectiveness deployment, which configuration fits their needs. This social networking platform joins together 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).

This repository means to several benefits, besides the ease of a unified user interface and a single sign-on. 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 and from code-centric to model-centric is expected to facilitate interaction between DevOps professionals. 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 Motivation

## 1.2 Background

## 1.3 Methodology

## 1.4 Other section

## 1.5 Related Work

This section describes related work for other professional networks and their system architecture.

### 1.5.1 Professional Networks

Copy from jisa and add more sites?

### 1.5.2 Memcached

Facebook serves billions of requests per second using memcached [1]. In this magnitude of scale Facebook has several pools of memcached servers (regional pools) along the globe. A single request for a page can produce hundred of requests to the back-end system. Memcached 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 server 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 [1] 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 in the same time. Stale sets resolved by a N-bit token, bound to specific key, sent from memcached to web server that want to update the key when cache miss occurs. If a delete request received then the request for updating this value from that client is rejected. The thundering herbs solved by configuring memcached servers return a N-bit token only once every ten seconds per key.

Describe linkedin caching systems?

## Chapter 2

# Management of ...

General discussion . . .

**2.1** AA

**2.2** BB

**2.3** CC

**2.4** DD



## Chapter 3

# 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 3.1: At the first layer lives (1) the Social Networking engine, which runs all PHP scripts and described in section 3.1. At the second layer lives (2) the Memcached caching system, which described in section 3.2. At the third layer lives (3) the Social Network MySQL database, and (4) the CDO server - client components and the CDO repository.

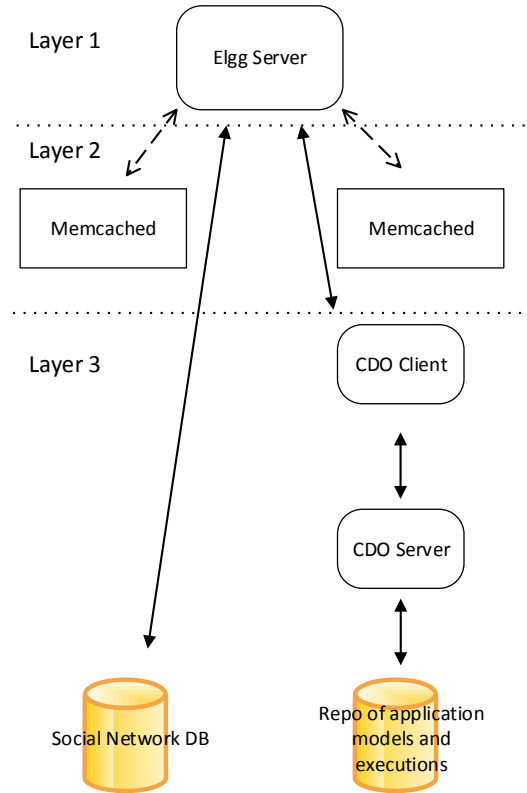
Achieving the scalability of the system, two system architectures are examined at two layers of the system: (1) We added more than one Social Network engine at the first layer of the system. In this implementation, in order to keep the file system in consistent mode we integrated Apache Zookeeper[2]. (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.

### 3.1 Implementation of Social Network

The social networking platform is implemented over the extensible Elgg social network framework[3]. Elgg is open source software written in PHP, uses MySQL for data persistence and supports jQuery [4] for client-side scripting. The Elgg framework is structured around the following key concepts:

- *Entities*, classes capturing social networking concepts: users, communities, application models, etc.
- *Metadata* describing and extending entities (e.g., a response to a question, a review of an application model, etc.).
- *Relationships* connecting two entities (e.g., user A is a friend of user B, user C is a contributor to an application model, etc.).

Figure 3.1: The overall architecture of Social Network.



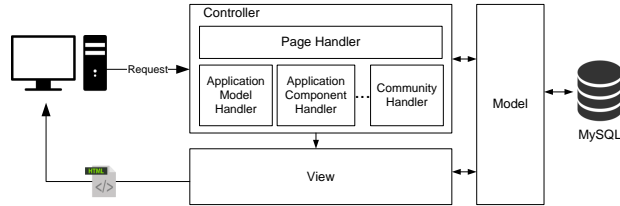
All Elgg objects inherit from `ElggEntity`, which provides the general attributes of an object. Elgg core comes with the following basic entities: `ElggObject`, `ElggUser`, `ElggGroup`, `ElggSite`, `ElggSession`, `ElggCache`, as well as other classes necessary for the operation of the engine.

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 (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.

Figure 3.2 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 3.2: Architecture of the Elgg Social Networking engine.



All plugins share a common structure of folders and php files. Folder *actions* includes the actions applied on application models (delete, save, or search). The *views* folder contains the *php* forms applied on application models, *river* events (Elgg terminology for live feeds), and the application model editor. *Pages* overrides elements of core Elgg pages. 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.

Social network relationships (friendship, group, ownership, etc.) are persisted in the Elgg back-end database. 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 CDO server is implemented over sockets.

## 3.2 Memcache

This section describes the experience gained by using memcached[5]. Memcached is an open source, high-performance, distributed memory object caching system. We choose memcached, because it is a generic simple in-memory key-value store. It has a powerful API available for PHP. After memcached integration the system increases the response time and performance.

Memcached stores all entities of Social Network, applications, components, users, group discussions and most important the executions of applications. 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[6] was used to measure the response time of the system and the sysstat tool[7] was used to measure the cpu usage. Section 4.1 shows the performance results of this implementation.



## Chapter 4

# Evaluation

This chapter describes the evaluation of the two different implementation of the system. (1) By introducing more than one memcached instances at layer 2 as figure 3.1 shows and (2) by introducing more than one Social Network engines at layer 1.

### 4.1 Improving Performance with memcached

### 4.2 Improving Performance with engine





## Chapter 5

# Comparison

Compare your work . . .

**5.1**    **AA**

**5.2**    **BB**



## Chapter 6

# Conclusions and Future Work



# Bibliography

- [1] 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.
- [2] A. Zookeeper, “Apache zookeeper,” <https://zookeeper.apache.org/>, 2015, [Online; accessed 20-May-2015].
- [3] E. S. N. Engine, “Elgg social networking engine,” <http://elgg.org/>, 2015, [Online; accessed 19-May-2015].
- [4] jQuery, “jquery,” <https://jquery.com/>, 2015, [Online; accessed 19-May-2015].
- [5] Memcache, “Memcache,” <http://memcached.org/>, 2015, [Online; accessed 18-May-2015].
- [6] A. jMeter, “Apache jmeter,” <http://jmeter.apache.org/>, 2015, [Online; accessed 19-May-2015].
- [7] sysstat, “sysstat,” <https://github.com/sysstat/sysstat>, 2015, [Online; accessed 19-May-2015].