DECENTRALIZED FAULT TOLERANT CACHING

WITH MEMCACHED

BIVAS DAS

Department of Computer Science

APPROVED:

Eric A. Freudenthal, Ph.D.

Luc Longpré, Ph.D.

Virgilio Gonzalez, Ph.D.

Patricia D. Witherspoon, Ph.D.
Dean of the Graduate School

to

MOTHER, FATHER

and

SHUBHRA

...

with love

DECENTRALIZED FAULT TOLERANT CACHING WITH MEMCACHED

by

BIVAS DAS, B.Tech.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

May 2011

Acknowledgement

First and foremost I offer my sincere gratitude to my supervisor, Dr Eric Freudenthal, who has supported me throughout my thesis with his patience while allowing me the room to work in my own way. I am greatly indebted to him for my Masters degree without whose encouragement and effort this theses would be incomplete.

I also thank the members of my graduate committee for their guidance and suggestions. Special thanks go to Dr. Luc Longpré, without his assistance this study would not have been successful. Also, thanks to Dr. Virgilio Gonzalez for helping me with this theses.

I am heartily thankful to my dear friends Manali Chakraborty and Amritam Sarcar. Without their persuasion and guidance, I would not have come to study abroad in the first place. They helped me in countless ways everytime I needed them. I greatly appreciate my good friends, Avranil Tah for all of his help and friendly support when I needed it the most.

I also want to thank my fiancé Shubhra Datta for her constant efforts and never ending patience for inspiring me to work harder.

Last but not the least, I thank my parents for supporting me throughout all my studies at UTEP, without their motivation and thoughtful decisions, I would not have been even what I am now.

And finally, I offer my thanks and regards to all of those who supported me in any respect during the completion of the thesis.

Bivas Das

NOTE: This thesis was submitted to my Supervising Committee on May 11, 2011.

Abstract

Recent changes in web trends not only have increased popularity of web services, but also have vastly increased usage. Popular techniques, such as web-caching, used by content and service providers to provide faster content delivery at user end, is not enough to keep up, due to diversity and dynamic nature of web-contents.

Emerging ideas to cache on the server side, for faster content generation and delivery, is currently in use by popular web-service providers. One example of such caching system is Memcached. However, memcached, a distributed high performance caching system, is ineffective in dynamic organization of itself and scaling when required, such as failure or adding more systems on demand.

In this theses I have designed and implemented a group membership protocol within a set of Memcached servers, used as a caching pool, that can make the pool aware of its participants. The protocol also helps to dynamically resize itself in case of change of members in the pool. This way in case of both failure and adding more nodes, the pool automatically adjusts itself and publishes the information to the pool clients when adapted to the protocol. I have also shown effectiveness of the protocol by running several test cases.

Table of Contents

Page

Acknowledgement v

Abstract vi

Table of Contents vii

List of Figures ix

Chapter

1 Introduction 1

1.1 Caching and Web Services 1

1.1.1 Advantages and Disadvantages of Caching 1

1.1.2 Evolution of Caching 2

1.2 Emergence of WEB 2.0 as a platform 2

1.2.1 Impact of WEB 2.0 4

1.2.2 Caching in WEB 2.0 5

1.3 Memory Based Object Caching With Memcached 6

1.3.1 Managing Memcached with Membership Protocol 6

1.4 Scope of this Thesis 7

2 Related Works 9

2.1 Paxos Group of Membership Protocols 9

2.2 Distributed Hash Table and Consistent Hashing 10

2.3 Caching Trend in Popular Web Services 11

3 Design and Implementation 14

3.1 Decentralized Leader Approach 14

3.1.1 Master Selection Within The Pool 14

3.1.2 Cache Reliability and Availability 15

3.1.3 Ring Formation and Keeping Track of Members 16

3.2 Message Passing Architecture . . . 17
3.2.1 Communication Messages . . . 17
3.2.2 Fault Tolerance due to Message Loss . . . 18
3.3 Server Side Protocol Description . . . 19
3.3.1 High-Level Protocol Description . . . 19
3.3.2 Detail Description of The protocol . . . 22
3.4 Client Side API Modification . . . 26
3.4.1 Client Side API . . . 27
3.5 Implementaion Details . . . 28
3.5.1 Server Side and Client Side . . . 28
3.5.2 Implementation . . . 28
4 Experiments and Results . . . 30
4.1 Experiment Setup . . . 30
4.1.1 Designing The Pool (of Servers) . . . 30
4.2 Results . . . 32
4.2.1 Responsiveness With Higher Interval . . . 33
4.2.2 Awareness With Dynamic Changes . . . 33
4.2.3 Responsiveness In Simple Failure . . . 35
4.2.4 Handling Complex Failure . . . 35
4.2.5 Interpreting The Results . . . 41
5 Future Works and Summary . . . 42
5.1 Future Works . . . 42
5.2 Summary . . . 43
References . . . 44
Appendix
Resources . . . 49
Curriculum Vitae . . . 50

List of Figures

1.1 Proxy Caching in Corporate (Firewalled) Network 2
1.2 Hierarchical Caching and Distributed Caching 3
1.3 Caching on the Server Side 5

3.1 Ring Formation Within The Pool 16
3.2 High Level Protocol Diagram 20
3.3 Initialization Stage 20
3.4 Pre-Grouping Stage 21
3.5 Pre-Master Stage 21
3.6 Master and Slave Protocol Diagram 22
3.7 Process State in Master and Slave 24
3.8 Recovery State Protocol Diagram. 25

4.1 System Responsiveness at Higher Interval 33
4.2 System Responsiveness at Lower Interval 34
4.3 Client without Protocol in Simple Failure 36
4.4 Client with Protocol in Simple Failure 37
4.5 Client without Protocol in Complex Failure 38
4.6 Client with Protocol in Complex Failure 39
4.7 Client with Protocol With Lower Interval 40

Chapter 1

Introduction

1.1 Caching and Web Services

World Wide Web has an exponential growth and increasing number of users, resulting in network congestion and server overloading. As a solution researches from early 90s suggested to cache popular objects closer to client location [44]. Such as a proxy server on corporate network (see Fig. 1.1) can be used to cache external resources. With time, this web-caching became a popular trend to serve users with faster content delivery [34].

1.1.1 Advantages and Disadvantages of Caching

Caching reduces the used bandwidth and latency by fetching a majority of the contents from local location. Therefore, the un-cached documents are fetched faster using the bandwidth gained in caching mechanism [28], [20]. Furthermore, it not only reduces the workload on the remote server, but also assures some availability when remote server is not available. Also, the service providers can organize the contents and their availability by studying the usage pattern of clients.

Cache miss can significantly increase access latency based the availability and locality of the contents. A naive cache-update procedure may present the client with stale and irrelevant data. A single proxy-cache-host can be a bottleneck, may degrade performance, and can be single point of failure. Also proxy-cache must be set to maximum allowed to clients for stable operation. Furthermore, content providers' fail to find true access of contents, as caching reduces access to their (remote) server, consequently failing to rank

them based on access rate.

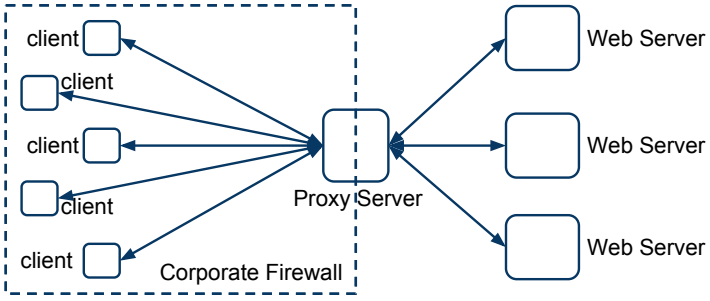

Figure 1.1: Proxy Caching in Corporate (Firewalled) Network

1.1.2 Evolution of Caching

Cache architecture (see Fig, 1.2), can be hierarchical [15] (contents are cached at multiple levels and queried in order until it is found) or can be distributed [39] (contents are placed at the same level and is shared without replication). It might also be a hybrid system between these two [18], optimizing replication and access latency. Cache usability determines which type of content will be cached by the system. Examples of cached contents are document, image, streaming content, results of computation, etc.

In prolonged operation, better availability can be assured by co-operation among caches, where neighbor responds to cache misses with appropriate content if available. Cache resolution and routing of the data efficiently assures smooth operation. Also, studying usage pattern and using heuristics to prefetch contents reduces latency significantly [38], [29], [19]. Caches placement and management also influences throughput and access latency [10].