UNIVERSITY OF WATERLOO
Faculty of Engineering

MEMCACHED VS MYSQL QUERY CACHE. AN ANALYSIS OF DATABASE CACHING SOLUTIONS FOR TIME SERIES ANALYSIS

CPP Investment Board
Toronto, ON

Prepared by
John Antony Green
ID #20235387
ja2green@engmail.uwaterloo.ca
3A Computer Engineering

May 11, 2009

John Antony Green
256 Phillip St. Unit 45.,
Kingston, ON N2L 6B6

May 11, 2009

Manoj Sachdev, Chair
E&CE Department,
University of Waterloo,
Waterloo, ON N2L 3G1

Dear Manoj Sachdev:

Re: Submission of my work term report.

This report, entitled Memcached vs MySQL Query Cache. An analysis of database caching solutions for time series analysis , was prepared as my 2B Work Term Report for the CPP Investment Board. This report is in fulfillment of the course WKRPT 300. The intent of this report is to analyze several data caching solutions, compare their performance and ultimately come up with a suitable recommendation.

The CPP Investment Board is an government independent organization created to monitor and invest the funds of the Canada Pension Plan. I was a member of the Research Team, within the Capital Markets Division, part of larger, encompassing Public Market Investments Department. Our research required us to store and maintain large amounts of fin

nancial data, which we would subsequently run analytics on. This report focuses on the some of the ways in which we data retrieval using caches.

I would like to thank Mr. Manjeet Ram for providing direction on this research endeavor. I hereby con

rm that I have received no further help other than what is mentioned above in writing this report. I also con

rm this report has not been previously submitted for academic credit at this or any other academic institution.

I hereby confirm that I have received no further help other than what is mentioned above in writing this report. I also confirm that this report has not been previously submitted for academic credit at this or any other academic institution.

Yours sincerely,

John Antony Green, 20235387

Encl.

Contributions

The CPP (Canada Pension Plan) Investment Board is a organization independent of government, created in 1997 to invest the funds held by the CPP. CPPIB is a crown corporation created by an Act of Parliament [1]. Prior to 1997, the CPP fund was fully invested in federal government bonds, but has since diversi

ed both by asset class and geographically. Now, it holds a diverse set of investments in equities,

xed income, and real estate. I worked within the Capital Markets Department, which focused mainly on trade execution in the public markets. This includes tradable instruments such as equities, bonds, foreign exchange currency pairs , and derivatives. The Capital Markets (CM) Department consisted of approximately thirty people. The research team, of which I was a member, is a relatively small subset of the CM department . Our team consisted of twelve people.

The research team was composed of software developers and mathematicians, focused on creating software that performed various analytics and computations on

financial data. The end result of these processes was frequently a report which was in turn used by members of other departments within Capital Markets, to aid them with financial analysis. The volume of data required for these processes was extremely large. We dealt mainly with time series data, a sequence of data points measured at at successive times. In our context, the sequence typically consisted of the prices of a financial instrument. The

software our team developed ran an array of analytics on the aforementioned time series data, ranging from simple to the computationally intensive. The overarching goal of our team was to provide an accurate, organized and useful analysis of data for the investment professionals within our department. This would, ideally, contribute to more informed decisions, and in turn help generate profit.

My primary duties were application development, database and system adminis- tration, and research into alternative data solutions. The bulk of my time was di- rected towards the development, maintenance and deployment of the aforementioned software. I typically worked on small applications where I was the sole developer. Since our team was relatively small, the development process was iterative in na- ture. There was no formal quality assurance department; we were required to test our own software. Maintenance and improvement of existing software was also one of my responsibilities. In addition to this I was researching ways to speed up fetching time series from disk using caches. I developed a project which used a distributed caching system to handle frequently used sets of financial data. Throughout the development of the project, I encountered various design decisions, which form the basis for this report.

The CM department wanted faster data access on time series queries and I suggested implementing a caching system. My manager asked me to complete a qualitative and quantitative analysis of possible solutions and then implement the solution which I thought was superior. Through this analysis, I was able to improve my ability in technical analysis and the associated presentation of ideas. The software I helped develop is being used to generate reports and

manage data used by our department. As mentioned above, this will ideally help our department and others make more informed decisions and ultimately contribute to the bottom line of the fund.

Summary

The purpose of this report is to analyze several data caching approaches in the context of time series data. The solutions which are being analyzed are Memcached, a standalone caching application and MySQL Query Cache, a feature within the MySQL database system. The intent of this report is to compare the performance of a standalone object caching system with a integrated query caching system and draw some conclusions as to their feasibility in the realm of time series data analysis. The reader should gain perspective and insight into the advantages of a cache over continually reading from disk as well as the possible implementations of such a caching system. This report is intended for anyone interested in open source, freely available caching techniques for financial data.

The caching systems will be first analyzed qualitatively, highlighting some of the architectural features of each, and the relative advantages and disadvantages of implementation. Quantitative analysis will follow, in the form of a benchmark for a large time series. This analysis should give the reader a good view into not only the relative performance of each solution, but also it's suitability for different contexts through an analysis of the architecture and implementation details.

The report concludes that Memcached provides optimal performance and greater flexibility and convenience in regards to object serialization and processing. MySQL Query Cache, on the other hand, provides slightly worse, yet similar performance, but does not require a standalone process in addition to the main database store. For raw performance and ease of implementation, Memcached is superior. Although, since the performance of each solution is

similar, the choice of a caching system can be largely be based on specific business need and situation.

Either of the caching systems analyzed in this report provide an huge performance advantage over continually fetching from disk. Since both of these solutions are free and the only cost of implementation is the server hardware involved,there is little reason to resist the implementation of such a system. My recommendation is that a memcahed based solution be implemented if optimal performance and greater flexibility is required, but if an exisiting MySQL server exists, query caching is a good option, with similar performance. In the context of my employer, I would recommend that MySQL query caching be enabled instead of the implementation of a memcached client/server, since all of our data is currently housed in a MySQL server. The benefit of a memcached based implementation would be minimal and would not offset the cost of such a system.

Table of Contents

Contributions . . . . . **iv**

Summary . . . . . **vii**

List of Figures . . . . . **x**

List of Tables . . . . . **xi**

1 Introduction **1**
1.1 Background 1

2 Overview of Products **3**
2.1 Memcahed 3
2.2 MySQL Query Cache 3

3 Architecture Comparison **4**
3.1 Memcached 4
3.2 MySQL Query Cache 4

4 Implementation Comparison **5**
4.1 Memcached 5
4.2 MySQL Query Cache 5

5 Performance Analysis **6**
5.1 Background 6
5.2 Results 6

References **7**

List of Figures

List of Tables

1 Introduction

The report begins with an overview of the caching solutions which will be subsequently analyzed. For each solution, there will be a brief background section, followed by an analysis of their architecture and functionality. A performance benchmark will follow. The majority of the background information was obtained from firsthand knowledge through test implementation of the solutions. Additional information is referenced accordingly.

1.1 Background

A cache is a temporary storage area where frequently accessed data can be stored for fast access. Once data is cached, it can be used in the future by quickly accessing the cached copy instead of the of re-fetching the original value. By taking advantage of the concept of temporal locality: the idea that data is frequently reused within relatively small time durations, caching has become an extremely effective tool in improving performance and scalability. The concept of caching has been used in many different areas of computer science. Perhaps the most famous example is the CPU cache, which is used by the CPU to reduce access time to memory. However, caching is not limited to applications within hardware design, as the same concept has been implemented within software. An area of application which is of specific interest is database caching.

The volume of financial data has grown considerably over the past decade, and the databases used to store this data must use increasingly sophisticated methods in order to maintain adequate performance for analysis. Database

caching techniques have been implemented as an aforementioned strategy. In the financial industry, time series data is usually accessed repeatedly, since computations are generally done on a single asset repetitively. The data access patterns are therefore perfectly exploitable by a caching system, translating into significant performance advantages. By placing a caching system in the middle tier, and caching the results of queries, or the objects obtained from the result set, performance can be significantly improved.

The main strategies involved in database caching consist of an integrated solution within the database system itself, which simply caches the results of queries

2 Overview of Products

2.1 Memcahed

Memcached is a high performance, distributed memory caching system. It was originally developed by Danga Interactive for LiveJournal, a website which was doing 20+ million dynamic page views per day. Memcached was developed to alleviate database load, and thus yield faster page loads from the user perspective. It is frequently used by large database driven websites to cache data and objects in memory, preventing unnecessary and costly database fetches. This caching system is used by many large well known sites, including Youtube, Wikipedia, Facebook, and Digg.

Memcached provides a large hash table distributed across multiple servers. It works by allocating a block of memory, of which the size is user defined. This block will contain key value pairs, mimicking the functionality of a hash table. However, memcached is unique in the fact that it spans transparently across servers, creating a massive hash table, spread over a network. Thus any computer with available memory can be used as an additional node, increasing the size of the cache.

Memcached is a relatively small 3000 line program written in C, which runs as a lightweight daemon on a server machine. Key value pairs are inserted programmatically using a client API. There is no restriction on the type of data which can be inserted into the cache. When the table becomes full, data is removed using the LRU replacement algorithm described above.

2.2 MySQL Query Cache

3 Architecture Comparison

3.1 Memcached

3.2 MySQL Query Cache

4 Implementation Comparison

4.1 Memcached

4.2 MySQL Query Cache

5 Performance Analysis

5.1 Background

5.2 Results

References

[1] T. Akenine-Moller and E. Haines, *Real Time Rendering.* Wellesley, MA: A K Peters, Ltd., second ed., 2002.