Making Cheetah Faster

Alan Ng, Gary Chaw, Dustin Kut Moy Cheung

Department of Electrical and Computer Engineering
University of Toronto
{alan.ng, gary.chaw, dustin.kutmoycheung}@mail.utoronto.ca

Abstract—In this report, we describe how we modified Cheetah, an in-memory database, to speed up queries and insertions using threads. The queries and inserts are transformed into tasks in a queue and the threads in a threadpool pull tasks to execute. Our results show a speedup in query time as the number of threads increase when using the column store. However the inserts suffer from the global lock imposed on the store which results in poor performance. Our results show that it is possible to improve the performance of Cheetah by adding more threads and exploiting the different cores in a multi-core processor. However we were unable to explore the impact of cache locality in our design to determine a better algorithm due to tools limited in the Java ecosystem to run these kinds of analysis.

1 Introduction

The need for in-memory databases has exploded in recent years with the requirement for faster data access to provide near real-time data to users. These types of databases are challenging the dominance of traditional relational databases in an era where multi-core processors and main memory are becoming cheaper.

Unlike traditional databases, in-memory databases do not require optimization of hardware for disk storage, or tweaks to the underlying operating system, allowing for an extremely high throughput rate for writes and access using default configurations.

Moreover, in-memory databases do not suffer from unpredictable performance experienced by traditional databases as the seek time required to read data from a hard-drive is eliminated.

Typically in-memory databases are employed when huge amounts of data needs to be queried and processed in real-time. Examples might include having a real-time feed of tweets having a specific hashtag for users of the Twitter platform.

This demo file is intended to serve as a "starter file" for IEEE Computer Society journal papers produced under LaTeX using IEEEtran.cls version 1.7 and later. I wish you the best of success.

1.1 Subsection Heading Here

Subsection text here.

2 Background

Cheetah inserts data into its store by reading JSON data saved in a file. The JSON data consists of arrays of objects, where each object consists of many keys. Each of these keys might contain values of type: boolean, arrays, number, strings, and objects. The keys for each object are often different from each other, making them unsuitable to be stored in a traditional table because of the sparseness of the data.

In a traditional database, data are organized into tables, where each table consists of columns. Each column has a particular type associated with it. The global configuration of tables and the column types is known as the schema of the database.

If one would like to store all of the objects in the JSON data in a traditional database, one of the strategies would be to save the objects into one table, where each column will represent the keys that all the objects have. However, this causes huge gaps in our table since most of the time only a few keys are used in an object, causing a waste of space and processing time.

Once those JSON data are parsed, the data can be stored into the main memory using different configurations.

2.1 Column Store

The column store creates a new table for each key in the object to store. The table name is the key whose

value needs to be saved. To know which value is linked to each object, every object being saved is assigned a unique object id, and each value is then mapped to that object id. As such, the column table has two columns, one for the object id and one for the value.

Those columns are implemented internally by using arrays. Other data structures are explored later in this report.

The column store performs better memory-wise compared to the other stores if the objects being stored do not specify a value to all the keys allowed for that object, i.e sparse data. However, searching data requires more time using that layout because we have to scan through all the tables in the store to see if the key table has data for a particular object id.

3 Conclusion

The conclusion goes here.

Acknowledgments

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.