EagerDB:
A Predictive Cache Warming Tool
6.885 Final Paper

John J. Wang
wangjohn@mit.edu

December 11, 2013

Abstract

Many applications increase database performance using some form of caching. The majority of cache implementations, however, only increase performance for queries that have been already seen. EagerDB takes this a step farther and provides a predictive cache. It predicts queries that will occur in the future with high probability and automatically loads them into the cache. EagerDB allows the user to manually specify query preloads or automatically parse historical database logs, estimate a Markov model, and preload queries which have high transition probabilities.

1 Introduction

Most web applications improve database performance by caching – storing data has has previously been used. Using a cache can dramatically improve database performance when temporal locality is present (i.e. when previously requested data is requested multiple times). Caching has become one of the most fundamental concepts in computer science, and in turn, almost all large technology companies use caches in some form.

However, caches assume that historical data repeats. This assumption does not capture all of the potential performance gains that are possible. In fact, there is quite a large amount of data that exists which can be used to improve performance even further. Since there workflows of users tend to be methodical, the queries that get sent to a database can be predicted. One can then estimate the probability of some query X happening given the current state of the database, and "preload" X when it has a high probability of happening.

EagerDB provides a framework for preloading such queries. Developers can manually specify queries to preload or they can use a prediction engine for discovering queries to preload. EagerDB is built as a Ruby gem and can be easily included in a Ruby on Rails web application for easy improvements in database performance. This paper shall outline the design of EagerDB, the reasons for making particular design decisions, and the results of benchmark tests.

2 Background

There have been a number of papers which have had similar ideas.

3 Manual Query Syntax

EagerDB has its own easy to use syntax for manually specifying preloads. Behind this manual query syntax is the idea that any query Y can be used as an indicator for a later query X. Thus, if the probability of X occurring is high given that Y has occurred, then one should preload X. We call Y the *match statement*, and X the *preload statement*.

3.1 Basic Definitions

In order to explain match and preload statements more, we need to make a distinction between non-binded and binded SQL queries. A *binded SQL query* is just any instance of a valid SQL statement which contains table, column, and attribute values. An example of a binded SQL query would be:

```
SELECT * FROM distributors WHERE name = 'walmart' AND
  town = 'Chicago' AND state = 'IL'
```

A *non-binded SQL query* is just an instance of a valid SQL statement which contains table and column values, but does not contain attribute values. For example, the non-binded version of the previous SQL query would be:

```
SELECT * FROM distributors WHERE name = ? AND
  town = ? AND state = ?
```

Non-binded SQL queries can be converted into binded SQL queries by providing *bind values*, an array of values which would be inserted into each bind location (denoted by a ? in the non-binded SQL query). To convert the previous non-binded SQL statement into a binded SQL statement, one would need to pass in the following array of bind values:

```
['walmart', 'IL']
```

3.2 Match Preload Files

Match and preload statements are non-binded SQL queries. Match statements allow EagerDB to match on a particular SQL statement structure, and preload statements allow us to preload a specific SQL statement structure when match statements occur. In this vein, one could manually specify match and preload statements. This is done by first specifying a single match statement, then any number of preload statements to load into the cache whenever that match statement is seen.

In a *match preload file* (MP file), a developer using EagerDB can specify a list of match statements and their corresponding preload statements. EagerDB can read over an MP file and incorporate any match and preload statements that have been made inside the file. A developer can use the MP file as a stand-alone product without running the Markov model so that the preload statements in the MP file will be the

only statements preloaded by EagerDB, or incorporate the MP file as an addition on top of the Markov model's statements. The syntax for an MP file is straightforward:

- Match statements are preceded by a dash "-".
- Preload statements are preceded by a rocket "=>". Preload statements will be paired with the last match statement seen in the MP file.
- SQL statements are encapsulated by quotation marks.
- Bind values, separated by commas, follow the SQL statement. The i^{th} bind value in the SQL statement corresponds to the i^{th} comma separated value after the SQL statement.

In addition to these syntax rules, there are methods provided for bind values which allow making general preload statements much easier. We provide the `match_result` and `match_bind_value` keywords.

The `match_result` keyword provides access to the result of the match statement of a preload. One can access the value of a column in the result by asking for the column name like so: `match_result.column_name`. For example, if a developer wanted to access the id of the result from his match statement, he could write a preload statement like so:

```
=> "SELECT * FROM things WHERE product_id = ?", match_result.id
```

The `match_bind_value` keyword is a similar construction, but provides access to the bind value of the binded instance of the match statement. One specifies an index i after the `match_bind_value` keyword, which gives access to the ith bind value from the match statement. For example, one could write:

```
- "SELECT * FROM products WHERE name = ?"
  => "SELECT * FROM things WHERE product_name = ?", match_bind_value(0)
```

In the above, whenever of a SQL query of the form `SELECT * FROM products WHERE name = ?` is seen, EagerDB will take the zeroth bind value (bind values are indexed by 0) from the statement and use it as the first bind value in the preload statement `SELECT * FROM things WHERE product_name = ?`.

```
- "SELECT * FROM users WHERE name = ?"
  => "SELECT * FROM products WHERE owner_id = ?",
       match_result.id

- "SELECT * FROM pinterest WHERE pin = ? AND interest = ?"
  => "SELECT * FROM tables WHERE pin = ? AND interest = ?",
       match_bind_value(0), match_bind_value(1)
  => "SELECT * FROM interests WHERE interest = ? AND
       pinterest_id = ?", match_bind_value(1), match_result.id
```

Figure 1: A snippet from an MP file

Figure 1 provides an example instance of an MP file. In this file, there are two match statements. The first match statement has a single preload, while the second

match statement has two preloads. EagerDB will listen to the stream of SQL queries coming from the database, and whenever a match statement matches the structure of a query, the preload statements associated with that match statement will be brought into the database's cache.

For example, using the MP file from figure 1, suppose the following sequence of queries arrived at the database (ordered chronologically):

```
SELECT * FROM products WHERE name = 'table' AND state = 'IL'
SELECT * FROM users WHERE name = 'john'
SELECT * FROM laptops WHERE BRAND = 'lenovo' AND serial_number = '5'
```

Then, after the second query (`SELECT * FROM users WHERE name = 'john'`) was run, its corresponding preload statement would be brought into the cache. For example, if the second query returned a result with an id of 52, then the following query would be brought into the database's cache: `SELECT * FROM products WHERE owner_id = 52`. The other queries in the sequence did not match the structure of any of the match statements, so EagerDB would not bring anything into the cache after their execution.