15-400 Project Proposal: Using Just in Time Compilation to Implement an In-Memory Database Key Comparator

Abigale Kim

November 2020

1 Project Title

My project title is "Using Just in Time Compilation to Implement an In-Memory Database Key Comparator".

2 Project Web Page

The web page for this project can be located at https://abigalekim.github.io/15400-research/.

3 Project Description

I will be working with professor Andy Pavlo, who works on databases research, on this project. Associated PhD students I will also be working with PhD student Prashanth Menon, who recently wrote a paper on JIT compilation for query execution, and Masters student Abhijith Anilkumar, who is working on JIT compilation for database indexes.

The feature I am planning on implementing is an in-memory database key comparator with the just-in-time compilation technique. Key comparators in databases are important because in databases, we typically run queries on keys, and in order to run these queries, we need a way to compare keys. This is done with a key comparator. The current key comparator cases on the type of the key, and then returns the corresponding comparator. However, since it currently uses a switch statement, branch prediction assembly optimizations don't work particularly well, and this has been identified as a bottleneck in CMU DB Group's database. Although we have a fast comparator for integers, we don't have a fast comparator for varchars, or strings, and this is where the switch statement fails us.

A much faster method of generating the key comparator would be to generate the correct key comparator as the program was running. This method is known

as just-in-time compilation, and has been predicted to run much faster than the previously implemented switch statement, especially on strings.

The main goal for this project is to implement the key comparator and demonstrate that it increases performance significantly. However, there are a couple challenges. First, I have to learn code generation, concurrently while working on this project. I will be taking 15-745 next semester (Optimizing Compilers) to help me learn the necessary information I need, and I do have background databases knowledge, but it will be difficult to learn and apply very quickly. Additionally, I have to integrate my changes with the larger databases system. This is difficult since it is a large codebase and there are many people working on different features within it.

The project is important because it has very high potential to significantly increase the speed of running the database. The CMU DB group has shown that applying the just-in-time compilation technique to query execution leads to 2x better performance on benchmarks against other DBMSs. This project is also interesting because we are implementing a new application of code generation– we are choosing to apply it onto database key comparators, which is something that has never been done before.

4 Project Goals (75%, 100%, 125%)

I hope to correctly implement the key comparator using just in time compilation techniques, and show that it significantly improves the performance of the database. The main metrics that will be used to assess my project is the performance improvement and successful integration into NoisePage, CMU DB's current database.

4.1 75%

If I complete 75% of the project, I will have correctly implemented the key comparator using just in time compilation techniques, but will not have tested it sufficiently enough (maybe minor testing) to prove that it improves the overall performance of the database.

4.2 100%

If I complete 100% of the project, I will have correctly implemented the key comparator, tested it sufficiently enough to show correctness and significant performance improvements, and added my changes to the existing database management system.

4.3 125%

If I complete 125% of the project, I will have achieved all goals listed in the 100% milestone, along with adding additional testing and benchmarking for my feature.

5 Milestones

In this section, we list the milestones of this research project for 15-300 and 15-400.

5.1 1st Technical Milestone for 15-300

My first technical milestone before 15-300 ends is understanding the compilers background I need to before I start this project. This might include reading up on compilers or even asking other professors (e.g. Professor Todd Mowry, who teaches 15-745) on material to read in advance. I also plan to read Prashanth's research paper and understand the ideas and implementation strategies he used for code generation, along with the research papers for OLTP Indexes (Trie Data Structures) listed on 15-721's (Advanced Database Systems) website.

5.2 Bi-weekly Milestones for 15-400

Here we list the bi-weekly milestones for the 15-400 project. They are ordered by date.

5.2.1 February 15th

Assuming that I have a key of a given type, write out the key comparator in TPL. TPL is the language CMU DB Group uses for code generation, and it is a combination of a C style language and SQL. Make sure my key comparator is written correctly, and test it. If this works, write several key comparators for other keys of different types. This is primarily to increase my understanding of TPL before code generating it.

5.2.2 March 1st

For the first key comparator that I wrote, generate this comparator using the code generation interface. Make sure the generation is correct. This requires generating the comparator into an AST.

5.2.3 March 15th

Start implementing the key comparator generation code. The key comparator generation code should take in a key of any type, and return the code generated comparator, so that this comparator can be compiled on the spot and ran.

5.2.4 March 29th

Finish implementing the key comparator generation code, and write correctness tests that show that your code generator works for keys of different types, especially strings, where it will be most effective. At this point, the key comparator generation code is all separate from the index which needs the keys.

5.2.5 April 12th

The next step in implementing the key comparator is to integrate this new comparator with the existing index code. Since the database index will be needing the key comparator, I need to integrate the key comparator generation code with the database index generation code.

5.2.6 April 26th

By this week, I should have finished writing tests and performed benchmark testing. If performance is suboptimal, then I must debug my code and figure out either what went wrong or why the code generation did not cause much improvement.

5.2.7 May 10th

Integrate the new key comparator code into the existing database. Ensure that the code passes all build checks. If necessary, I can write a report on my findings.

6 Literature Search

Here is a selection of papers and resources I have collected to help me gain full understanding of this project:

- Permutable Compiled Queries: Dynamically Adapting Compiled Queries Without Recompiling by Prashanth Menon et al. This is Prashanth's paper on JIT compilation applied toward query execution.
- The papers below are useful for when I combine my code generation for key comparators with the existing index, or the integration part of my research project.
  - *A comparison of adaptive radix trees and hash tables* by V. Alvarez et al. [1]
  - *HOT: A Height Optimized Trie Index for Main-Memory Database Systems* by R. Binna et al. [2]
  - *The adaptive radixtree: Artful indexing for main-memory databases* by V. Leis et al. [3]
  - *The art of practical synchronization* by V. Leis et al. [4]
- Several code snippets from NoisePage, the CMU DB Group's DBMS, that help me learn how TPL and code generation works within NoisePage. These include:
  - The original switch statement used for selecting key comparators in NoisePage.
  - Simple (and more complicated) code snippets in TPL.

- NoisePage's code generation interface.
- Tests for code generation and for code in TPL.

One important knowledge set I am missing is the understanding of code generation. To resolve this, I will be taking 15-745, or Optimizing Compilers, next semester. This class will give me enough knowledge to complete my research project.

7 Resources Needed

I am set with software! Relevant software is readily available, and I already have a development environment with NoisePage, the CMU DB Group's DBMS, set up on my computer. I do not need any hardware or machines to run my project besides my own personal laptop.

References

[1] V. Alvarez, S. Richter, X. Chen, and J. Dittrich. A comparison of adaptive radix trees and hash tables. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1227–1238, 2015.

[2] Robert Binna, Eva Zangerle, Martin Pichl, Günther Specht, and Viktor Leis. Hot: A height optimized trie index for main-memory database systems. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 521–534, New York, NY, USA, 2018. Association for Computing Machinery.

[3] Viktor Leis, Alfons Kemper, and Thomas Neumann. The adaptive radix tree: Artful indexing for main-memory databases. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, page 38–49, USA, 2013. IEEE Computer Society.

[4] Viktor Leis, Florian Scheibner, Alfons Kemper, and Thomas Neumann. The art of practical synchronization. In *Proceedings of the 12th International Workshop on Data Management on New Hardware*, DaMoN '16, New York, NY, USA, 2016. Association for Computing Machinery.