Lazy Schema Changes
15-300, Fall 2018

Wan Shen Lim

November 1, 2018

1 Project Info

- **Title:** Lazy Schema Changes
- **URL:** wanshenl.me/15-400/

1.1 Description

- **Who?** I am advised by Professor Andy Pavlo in the Carnegie Mellon Database Group. I will be working on this system alongside other students in the group.
- **What?** Much of the value in a database comes from imposing structure on its underlying data. This structure is described through a database schema. A schema is a blueprint describing the logical layout of the database, for example, it describes the name and number of attributes being stored inside the database. In today's database systems, however, schema changes are expensive and slow. This problem is aggravated by complex schema changes that may affect multiple groups of data ("tables") at once. Fast schema changes are desirable because schema changes delay queries and operations on the database, which are otherwise known as transactions. Transactions are the means through which users get databases to produce useful information, and are often executed millions at a time. When a pending schema change blocks transactions, time and processing power are wasted. We propose to implement fast schema changes in the in-memory Terrier database system, a new in-house database system being built here at Carnegie Mellon. Our goal is developing a proof-of-concept fast schema change system that supports fast schema changes through lazy propagation.

  The challenge is in identifying, implementing and cleanly categorizing the various types of schema changes possible. Benchmarking will also be necessary to determine when it makes sense to coalesce multiple schema changes into one.
- **So what?** All the current database vendors out there have "lazy schema changes", which are extremely limited subsets of the capability defined above. For example, they may only support sequentially adding a single non-null table column lazily.

Providing a well-researched and open-source implementation of lazy schema changes, alongside benchmarks, would allow all the databases of today to advance their own schema change procedures. This would greatly reduce database downtimes.

1.2 Goals

- 75%: supporting fast schema changes
- 100%: classifying schema changes
- 125%: ability to compile and coalescse related schema changes

1.3 Milestones

- **15-300:** I intend to familiarize myself with the rewritten Terrier system by contributing code to it. As a starting point, I am currently porting over the old Peloton parser. This should be done by the end of 15-300.
- **15-400:**
  - Feb 1: schema support
  - Feb 15: multi-version schema
  - Mar 1: schema change support
  - Mar 22: fast schema changes
  - Apr 5: cataloguing schema change types
  - Apr 19: benchmarking
  - May 3: flex time

2 Literature Search:

As part of my summer research, we examined the schema change strategies of many database management systems today. They can be broadly categorized into either query rewriting or copying systems.

For query rewriting, instead of immediately applying the schema change, the system keeps track of all the changes to be made. The old transactions are then automatically rewritten to fit the new transaction's format. Example implementations are PRISM [2], PRIMA [5], and InVerDa [4]. This approach is reasonable for basic schema changes as described in [7]. However, for complex schema changes, it may be quicker to perform the complex change immediately instead of paying the rewrite price on every pending transaction.

Copying may be better known as pointer-swinging. A new empty database table is created and modified as desired. Existing data is copied to the new table in small chunks; old transactions use the old table until copying is done. Once the copying is complete, we switch to the new database table and new transactions will operate on that instead. Example implementations are

pt-online-schema-change [3], Online Schema Change [1], and Oracle [6]. This approach is simple to implement, but requires twice as much space and is performance and memory intensive.

Schema changes are recently becoming more important, e.g. with the latest version of PostgreSQL strengthening the types of lazy schema change supported. In general, however, the problem is neither well nor systematically studied, with every vendor calling their own limited implementation "fast schema changes".

3 Resources Needed:

I already have a development environment set up for the project. I am able to contribute code to the repository and have done so. I may eventually need benchmarking machines, which we have somewhere.

References

[1] Mark Callaghan. Online schema change for mysql, 2010.

[2] Carlo Curino, Hyun Moon, and Carlo Zaniolo. Graceful database schema evolution: The prism workbench. 1:761–772, 08 2008.

[3] Baron Schwartz Daniel Nichter. pt-online-schema-change, 2018.

[4] Kai Herrmann, Hannes Voigt, Andreas Behrend, Jonas Rausch, and Wolfgang Lehner. Living in parallel realities: Co-existing schema versions with a bidirectional database evolution language. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1101–1116, New York, NY, USA, 2017. ACM.

[5] Hyun J. Moon, Carlo A. Curino, Alin Deutsch, Chien-Yi Hou, and Carlo Zaniolo. Managing and querying transaction-time databases under schema evolution. *Proc. VLDB Endow.*, 1(1):882–895, August 2008.

[6] Oracle. Redefining tables online, 2018.

[7] Lesley Wevers, Matthijs Hofstra, Menno Tammens, Marieke Huisman, and Maurice van Keulen. A benchmark for online non-blocking schema transformations. In *Proceedings of 4th International Conference on Data Management Technologies and Applications*, pages 288–298, 2015.