Diversifying locks for effective synchronization in dynamic graphs

Ayush Pandey, Julien Sopena, Swan Dubois, Marc Shapiro

Diversifying locks for effective synchronization in dynamic graphs

Ayush Pandey, Julien Sopena, Swan Dubois, Marc Shapiro

LIP6/Sorbonne university, Paris, France

Abstract

A graph database contains complex data, which application transactions access concurrently. Maintaining correctness under concurrent updates requires some form of synchronization; locking is widely used for this purpose. However, the locking techniques used in graph databases suffer from some drawbacks. Locking techniques that utilize metadata to exploit the topology of the graph for locking have limited scalability over dynamic graphs owing to the cost of maintaining the metadata. Most locking techniques also use the classical read/write locks, which are not sufficient to facilitate maximum concurrency due to the complex nature of trivial graph operations. Our work aims to address these issues by proposing a new extensible labeling scheme called CALock for large, rooted graphs that allows efficient lock grain identification in dynamic graphs. We also aim to introduce new, richer and finer lock types for graph operations that allow for more concurrency while maintaining the same synchronization level. In this paper, we present the current state of our work, the preliminary results and the future work we intend to do.

1 Introduction

Directed graphs are widely used to represent connected data, where a vertex represents some object or fact, and an edge represents a relationship. Well-known examples include social media graphs like Twitter and Facebook; Communication network graphs like Wiki-Talk and Web graphs generated by scraping websites like Google. Applications navigate the graph, while concurrent transactions make updates. Queries in graph databases may involve a large number of vertices and hops [5]. Ensuring the safety of updates and the consistency of query results in concurrent transactions requires synchronization.

Locking methods for graphs generally associate a *read/write* lock l with every *target vertex* v. The more vertices a transaction accesses, the more locks it acquires. As such, any locking technique can be optimized in two ways. One, by reducing the number of locks acquired per operation and two, by reducing the number of conflicts between locks.

Multi-granularity locking (MGL) [2] techniques reduce the number of locks by allowing a single lock (*guard*) to cover multiple vertices (*grain*) [6]. This is called lock coarsening. Common MGL techniques like Intention Locks [2] and DomLock [4] use some metadata in the form of labels to identify the lock grain. Choosing grains becomes harder when structural modifications alter the graph's topology. This is due to the extra work needed to keep track of metadata, often using long traversals.

In our work towards lock diversification, we study techniques to minimize the cost of locking while allowing the maximum amount of concurrency in dynamic graphs. We identify two independent research directions for this. First, towards effective lock placement by optimizing lock grain identification and second, towards minimizing conflicts by making the lock types more specific than simple read/write locks.

2 Overview of the work

To improve lock placement, we propose *CALock*, a topology-based locking technique that implements an extensible labeling scheme for large rooted graphs that allows efficient lock grain identification without the need for graph traversals. CALock uses the following recursive labeling function.

$$L_v = \{v\} \cup \{\cap_{u \in parents(v)} L_u\} \tag{1}$$

We derive equation 1 using path definitions from graph theory and design a MGL technique that uses these labels. We also prove that CALock is safe and live (proof omitted for brevity). An example of labeling is shown in Figure 2.

Alongside this, in order to diversify the types of locks, we study several typical workloads and identify a set of queries that applications typically execute on graphs, for example, single vertex shortest

paths, vertex/edge label updates etc. Using this set, we identify the synchronization choke-points. These choke points then serve as a design guide for the new locks.

3 Preliminary results

As a first proof of viability, we implement the recursive function from equation 1 in STMBench [3] and evaluate its performance against common MGL techniques.

We find that in almost all workloads, CALock allows for more concurrency and outperforms other techniques when used for dynamic graphs. One of the results is shown in Figure 1. DomLock performs the worst because of its inextensible labelling scheme while coarse and medium grain locks do not scale well with the number of threads. In general, CALock improves throughput by 4.5× and response time by 1.5×, for workloads with heavy contention. Similarly, the latency of CALock is 4× lower than DomLock and 2× lower than coarse and medium grain locks.

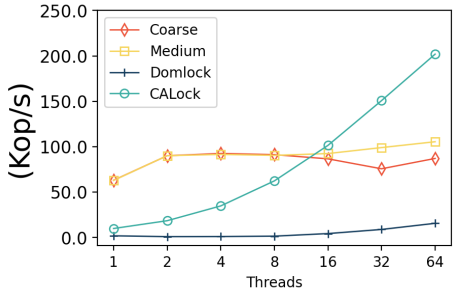

Figure 1: Throughput of Locking Techniques (R:90%,W:9.9%,SM:0.1%)

4 Future work

The function in equation 1 can be used to label generic graphs by labeling the paths from every root of the graph. However, whether the labeling is effective in identifying lock grains is yet to be studied. Since STMBench is a synthetic benchmark that simulates a real workload, it is only suitable for proofs of concept. We need to evaluate the performance of CALock on real-world graphs. To do this, we are implementing a new set of benchmarks on top of a graph database [1] that implements the CALock labeling and locking. To evaluate the performance, we intend to use real-world graphs from the SNAP datasets [5]. These benchmarks will better identify the bottlenecks in the CALock design.

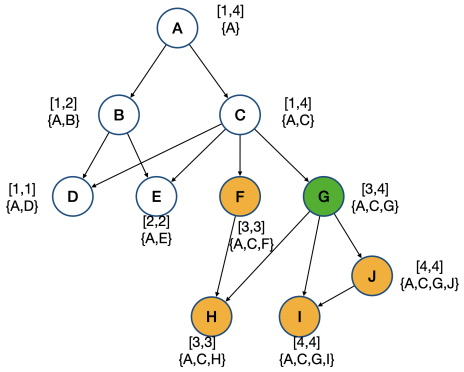

Figure 2: DomLock [x,y] and CALock {x,y} Labels Intervals with a lock on G

Another orthogonal problem we are studying is the diversification of lock types. The goal is for these locks to allow for more concurrency while maintaining the same synchronization level. As an example, the shortest path query is a read operation and an edge label update is a write operation. With only read/write locks, these queries cannot run concurrently on the same grain. But, changing the label on an edge doesn't affect the length of the path between two vertices that include that edge. As a result, it doesn't impact the outcome of a shortest-path query. Therefore, these two queries should be able to execute concurrently. To achieve this independence, we need locks that build on such trivial operation types and still achieve the same synchronization level as read/write locks.

5 Related Work

Several MGL techniques are used in practice. Some of them are following:

Intention lock [2] marks the vertices along all paths to the lock targets with intention locks (guards). Other threads traverse the graph and use intention tags to check for conflicts. If a thread encounters a tagged vertex with a conflicting mode, it waits until the tag is cleared or is not in conflict anymore.

DomLock [4] labels each vertex with an *interval* of the form $[min, max]$, assigned via a post-order traversal of the graph. Figure 2 shows an example of such labeling. The interval for an internal vertex is the minimum of the *min* and the maximum of the *max* values from the intervals of its children. Since the pre-order traversal of a graph changes with every structural modification, the intervals need to be recomputed. This makes DomLock inefficient for dynamic graphs.

References

[1] Apache Foundation. TinkerPop Documentation: Provider Reference. `https://tinkerpop.apache.org/docs/current/reference/#implementations`.

[2] Jim Gray, Raymond A. Lorie, Gianfranco R. Putzolu, and Irving L. Traiger. Granularity of locks in a large shared data base. In Douglas S. Kerr, editor, *Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA*, pages 428–451. ACM, 1975.

[3] Rachid Guerraoui, Michal Kapalka, and Jan Vitek. Stmbench7: a benchmark for software transactional memory. In Paulo Ferreira, Thomas R. Gross, and Luís Veiga, editors, *Proceedings of the 2007 EuroSys Conference, Lisbon, Portugal, March 21-23, 2007*, pages 315–324. ACM, 2007.

[4] Saurabh Kalikar and Rupesh Nasre. Domlock: a new multi-granularity locking technique for hierarchies. In Rafael Asenjo and Tim Harris, editors, *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2016, Barcelona, Spain, March 12-16, 2016*, pages 23:1–23:12. ACM, 2016.

[5] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[6] Daniel R. Ries and Michael Stonebraker. Effects of locking granularity in a database management system. *ACM Trans. Database Syst.*, 2(3):233–246, 1977.