Resolving Concurrent Graph Mutation Conflicts

Bliss
Team Member(s):
Aashish Dhakal
Eric Lin
Samantha S. Khairunnesa

Department of Computer Science
Iowa State University

COMS 641: Data Intensive Languages and Systems - Design and Semantics

- Problem
  - Conflicts when performing graph mutation operations in parallel, large data graph processing frameworks
- Background
  - Pregel as an example of graph processing framework
  - Graph mutation in these frameworks
  - Current solution in these frameworks
- Solution
  - Verification of reordering of mutation operations
  - Static detection of mutation conflicts
  - Operation abstraction approach to eliminate conflicts
- Evaluation
  - Graphs modeling and formal proofs using proof assistants
- Related works
- Other problems

- Many interesting problems require processing graph data for real world applications
  - Data Mining
  - Page Rank
  - Social Networking
  - Online Machine Learning
  - Business Intelligence
  - Analytics

- The scale of these graph, in some case contains billions of vertices, trillions of edges - poses a challenge to efficient processing
- Numerous frameworks have been developed to process these graphs efficiently (e.g. Pregel, GraphX, Giraph, Mizan, GraphLab, GPS, etc)

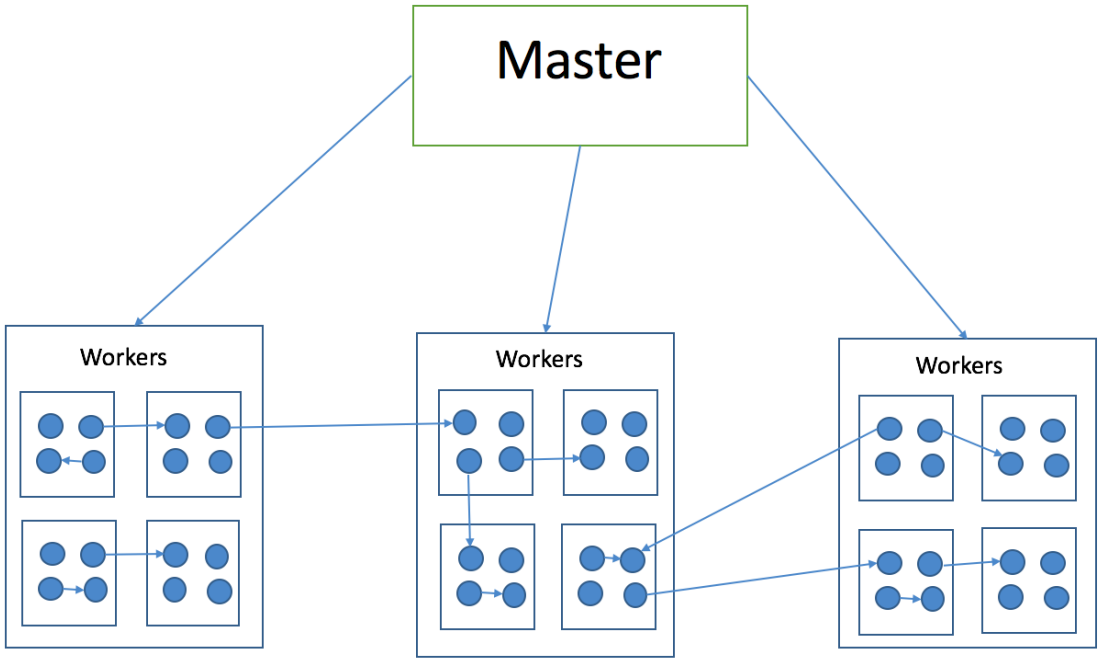

Figure: Execution model of Pregel

- Some graph algorithms requires a combination of vertex-centric (parallel) and global (sequential) computations
  - An example could be k-means-like graph clustering algorithm
- This type of algorithm requires a means of graph mutation operations
- These graph processing frameworks provide the means of graph mutation
  - e.g.: Pregel, Giraph, Mizan, GPS, Graph Lab (supports partial mutation)

- However, topology mutation operations performed concurrently at different vertices can conflict with each other
- All these graph processing frameworks resolve the mutation conflicts by using:
  - Partial ordering : edge remove $\rightarrow$ vertex remove $\rightarrow$ vertex add $\rightarrow$ edge add
  - User defined handlers
- Conflicts not solvable by the reordering are expected to be solved by the programmer

```
1  public interface Computation<I, V, E, M>{
2    void compute(Vertex<I,V,E> vertex, Iterable<M> messages);
3     // Must be defined by user to do computation on a single Vertex.

5    void  addEdgeRequest(I sourceVertexId, Edge<I,E> edge);
6    void  addVertexRequest(I id, V value);
7    void  removeEdgesRequest(I sourceVertexId, I targetVertexId);
8    void  removeVertexRequest(I vertexId);
9    long  getSuperstep();
10   long  getTotalNumEdges();
11   long  getTotalNumVertices();
12   void  postSuperstep();
13   void  preSuperstep();
14   void  sendMessage(I id, M message);
15 }
```

Essential operations of a *Computation* in Giraph

[giraph.apache.org/apidocs/]

```
1  Public class SimpleMutateGraphComputation extends BasicComputation<
2    LongWritable, DoubleWritable, FloatWritable, DoubleWritable> {

4    public void compute(Vertex<LongWritable, DoubleWritable, FloatWritable>
5      vertex, Iterable<DoubleWritable> messages){
6      if (getSuperstep() == 0) {
7        addVertexRequest(getVertexCount(), new
8        DoubleWritable(vertex.getValue()));
9      } else if (getSuperstep() == 1) {
10       vertex.voteToHalt();
11   }
12 }
```

Example of a program that mutates a graph

There are two kinds of conflicts:

- Conflicts such as the ones in the previous example still yields a well-formed graph after mutations, however the resulting graph is non deterministic
- Another possibility for mutations is to produce an erroneous graph. e.g. one vertex adding an edge, and one vertex removing the vertex of the edge being added, resulting in a dangling edge

- The first type of conflict still produces a valid output, but may not be correct according the requirement of the program
- The second type of conflict can happen because of the programmer's unawareness of the reordering, or algorithmic errors

Our focus is to further solve these conflicts in 2 ways:

1. Analysis to identify the conflicts

2. Propose an abstraction for graph mutation that can avoid having nondeterministic mutations from happening

1. Analysis to detect conflicts
   - Possible combinations introducing conflicts

	remove.e	remove.v	add.v	add.e
remove.e				
remove.v				X
add.v			X	
add.e				X

Figure: Combination that creates conflicts

- Refinement of the analysis
  - Look at the compute() of active vertices in a Superstep
  - Collect topology mutation operations
  - Prune the list if the operation in consideration does not fall into any categories that create conflicts
  - Take control and data flow into consideration
  - Formally model the graph mutation and use a proof assistant to prove the correctness of the process

```
1  Public class SimpleMutateGraphComputation extends BasicComputation<
2    LongWritable, DoubleWritable, FloatWritable, DoubleWritable> {

4    public void compute(Vertex<LongWritable, DoubleWritable, FloatWritable>
5      vertex, Iterable<DoubleWritable> messages){
6      if (getSuperstep() == 0) {
7        addVertexRequest(getVertexCount(), new
8        DoubleWritable(vertex.getValue()));
9      } else if (getSuperstep() == 1) {
10       vertex.voteToHalt();
11   }
12 }
```

Example of a program that mutates a graph

2. Handle conflicts by abstraction

- There are some conflicts that can not be solved by partial ordering
- User defined handlers are suggested to resolve such issues.
- Current mechanism is writing user defined handlers to randomly select one of the operation that introduces conflicts.
- A better abstraction can add priority to handlers to help decide the operation to be performed if there is a conflict.

Given a set of graph mutation, our goal is to prove:

- The soundness of our analysis to detect conflicting mutation
- The completeness of resolving conflicting mutation with the existence of required handlers.
- We want to model graph model graphs and graph transformation in a proof assistant and extend it further to prove the completeness of the 1st and 2nd solutions.

To evaluate the our anlaysis

- We formalize the analysis to evaluate a sequence of mutation and return the existence of conflicts.
- Formalize the semantic of newly defined mutation operations
- Show that analysis reports true if conflicts exists
- Show that conflicts doesn't exist with our proposed operations

One main contribution of FlumeJava is the optimization of execution plan by reordering the data operations. For example, ParallelDo operations are fused, related GroupByKey operations are combined by analyzing the data flow.

- Correctness of the optimization is not described
- Same for many optimization techniques that is used in many frameworks seen in the course
- Also similar to the problem we picked, but more specific to single techniques in specific framework.

A problem mentioned in the FlumeJava paper is that the abstraction of operations make it easier for programmers to write MapReduce programs, but the abstraction also causes programers to write program that is logical to them but is inefficient. The inefficiency is caused by the lack of understanding of the underlying execution.

- This can be optimized by having a better analysis to remove redundant operations
- Very specific analysis for a specific framework and difficult to apply to other works.

- The picked problem explores concurrent graph mutation in a concurrent context, which can be applied to following works on concurrent graph processing
- Graph processing have always been an important class of problems that can be used applied to many real world problems
- Work on concurrent graph processing can be more broadly applied, and may be more interesting to broader audience.

- Pregel provides the topology mutation feature to facilitate graph algorithms such as clustering algorithm, minimum spanning tree.
- Although it provides partial ordering to achieve determinism, yet forces the programmer to understand the reordering to close the gap between the logic of the programmer and what actually happens to effectively resolve conflicts that are not solved by partial ordering.
- Pregel-like frameworks such as Giraph, GPS, Mizan, and GraphLab supports graph mutations, yet do not address the conflicts arising due to concurrent topology mutation.

[Minyang et. el., "An Experimental Comparison of Pregel-like Graph Processing Systems", VLDB, Vol. 7 Issue 12, 2014]

- Martin et. al. formalized the graph transformation based on set-theoretic model.
- In the paper, it is mentioned that the graph transformation should specify the condition under which it will be applicable and the action needs to be taken at a position in a source graph to obtain target graph.
- They do not map graphs into graphs, as in traditional graph rewriting, rather verify that the applicability condition of a transformation rule is true

---

[Martin Strecker., "Modeling and Verifying Graph Transformations in Proof Assistants", ENTCS, Vol. 203 Issue 1, 2008]

- They have also claimed application of well-formed graph transformations to well-formed graphs yields again well-formed graphs.
- This kind of formalization has not been yet utilized to address the correctness of partial ordering.

[Martin Strecker., "Modeling and Verifying Graph Transformations in Proof Assistants", ENTCS, Vol. 203 Issue 1, 2008]

- Serafettin Tasci and Murat Demirbas, "Giraphx: Parallel Yet Serializable Large-Scale Graph Processing", EuroPar, 2013.
- Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, Carlos Guestrin, "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs", OSDI, 2012.
- Zuhair Khayyat, Karim Awara, Amani Alonazi, Hani Jamjoom, Dan Williams, and Panos Kalnis, "Mizan: A System for Dynamic Load Balancing in Large-scale Graph Processing" ACM EuroSys, 2013. Zuhair Mizan

- Semih Salihoglu and Jennifer Widom, "GPS: A Graph Processing System", Proc. International Conference on Scientific and Statistical Database Management (SSDBM), 2013.
- Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein, "GraphLab: A New Parallel Framework for Machine Learning, In Uncertainty in Artificial Intelligence, 2010.
- U Kang, Charalampos E. Tsourakakis, and Christos Faloutsos, PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations, IEEE International Conference on Data Mining (ICDM), 2009.

- Zechao Shang, and Jeffrey Xu Yu, "Catch the Wind: Graph workload balancing on cloud Data Engineering", IEEE 29th International Conference in Data Engineering, 2013.