A Revised Implementation of the GRAPH/Z Graph Processing System

Ballmer, Alexander
alexandersballmer@gmail.com

Moudgalya, Shreyas
smoudgal@hawk.iit.edu

June 7, 2016

Abstract

GRAPH/Z is a distributed parallel graph processing system running on top of ZHT, a zero hop distributed hash table. It uses an iterative vertex-centric model to store a large graph and run a variety of algorithms over it. GRAPH/Z has some performance issues, and cannot scale the same as Graphlab, a similar commercial framework. We hope to rewrite GRAPH/Z to be competitive with Graphlab on a single node with multiple threads.

1 Background

GRAPH/Z was based off of a graph processing paradigm called Pregel. Pregel uses a model centered around the vertexes of the graph. [3] Each vertex has an update function that is run in the vertex's context in the graph. The update function allows the vertex to modify its edges, perform calculations, and send messages to other vertexes. [3] The computation occurs in iterations called supersteps, with each iteration calling every vertex's update function theoretically in parallel. Even if there are fewer threads than vertexes, all of the update functions behave as if they are called in parallel. [3]
Messages sent by vertexes are used to communicate between vertexes, sending data to the next iteration. At each iteration, vertexes can vote to halt and disable themselves. A halted vertex can be re-enabled by receiving a message. If all the vertexes are disabled at the start of an iteration, the entire system halts and returns.
GRAPH/Z adapts the Pregel model, but adds the concept of a distributed hash table. The hash table stores both the graph's edges and vertexes, and also provides a platform for running the distributed message queue. The hash table serves as the only means of communication between nodes. The hash table used is ZHT, a DHT implementation that is fault-tolerant and can scale to 32000 cores. [4] ZHT abstracts away the physical hardware, exposing only a key-value store.

2 Problem

GRAPH/Z has experienced problems in terms of scaling competitively with commercial software like Graphlab. The exact cause of the problem is still unknown, but seems to be related to the way that ZHT deals with data locality between nodes. ZHT's hashing function does not distinguish between local data on one node and remote data on a network node in the cluster. [1] This could lead to data being inefficiently stored, needing high latency network access to retrieve it from a remote node. The hashing function also may result in some computational overhead.

3 Related Work

The closes related system to GRAPH/Z is Pregel, which it was inspired by. In most of our work, however, we compare GRAPH/Z to Graphlab, which is another high performance graph processing framework. Graphlab uses a similar paradigm to GRAPH/Z, but allows a vertex to access data that is not in a message to the vertex. [5]
Another less similar but still relevant work is Hadoop, which follows the MapReduce paradigm. Hadoop is frequently used to process large graphs. In fact, GRAPH/Z and Pregel computations can be expressed as a series of chained MapReduce functions. [5] Hadoop has largely been replaced by Apache Spark, which is faster in some cases.

4 Proposed Solution

Our main work to solve our problems of scalability is to backtrack and try to achieve good scaling on a single node, instead of strong scaling across multiple nodes. We believe that the problem with GRAPH/Z lies within the 'crosstalk' when restoring key/value pairs stored on a remote node, as ZHT has bad data locality. Our main goal is to achieve some level of competitiveness against GraphLab on a single node, and to lay down a framework to expand to multi-node scaling through ZHT or another distributed datastore. We will base performance off of runtime, weak scaling with larger datasets, and profiling tools such as valgrind/callgrind.

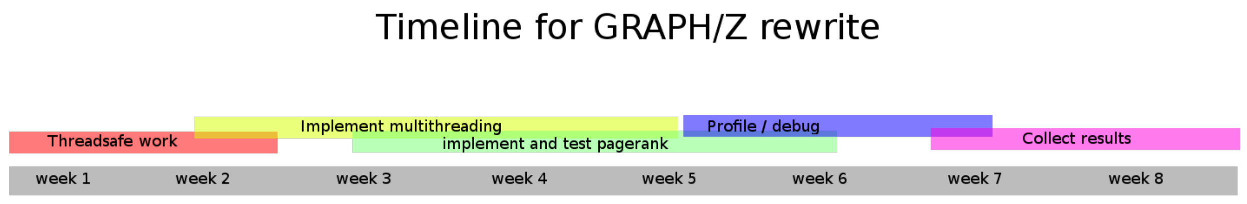
5 Evaluation

As the main goal for rewriting GRAPH/Z is performance on a single node, we will be using profiling tools such as valgrind and callgrind, along with basic runtime measurement, to measure the efficiency and speed of GRAPH/Z in relation to Graphlab. We will be using a modified pagerank algorithm defined as
$\forall t \in P : r^{(t)}(t) = (1-\alpha) \cdot r(t) + \alpha \sum_{(s,t)\in L} \frac{r^{(t-1)}(s)}{|L(s)|}$
and a graph partitioning algorithm defined as:
Given a graph $G = (V, E)$ and a a set of partitions, P, an output partition $V = v_0 \cup V_1 \cup V_2 \cup \ldots \cup V_{p-1}$ such that

1. V_i are disjoint $\Rightarrow V_i \cap V_i = \emptyset$
2. V_i are roughly balanced $\Rightarrow |V_i|\ |V_i|$
3. let $E_{cut} \equiv (u,v)|u \in V_i, v \in V_i, i \neq j$
   Minimize $|E_{cut}|$

While profiling at a function call level will help us achieve our goal, the metrics used to determine success will be overall runtime and memory use. Scaling will be determined on multiple cores, and increasing data size, but on a single node. These metrics will be used for the eventual later goal of scaling to more than one node.

6 Timeline



7 Deliverables

The main component of the deliverables will be a poster outlining the new GRAPH/Z's strengths and weaknesses and highlighting the changes that we made from the original project. The poster will also include the abstract and writeup needed for entering it into the Supercomputing conference. Deliverables will also include the finished GRAPH/Z processing system and information comparing it to GraphLab and other existing similar tools. Also included with be data from profiling and traces.

8 Conclusion

The original GRAPH/Z was underperforming compared to most other productions graph processing systems. We hope by rewriting it from scratch with a new backend storage system we will discover what may be the cause of the lack of performance. In order to determine if the ZHT distributed hash table is an IO bottleneck, we will confine our implementation to a single node, and use an alternate backend besides ZHT.

References

[1] Gagan Munisiddha Gowda, Benjamin L Miwa, and Anirudh Sunkineni. Zht+: Design and implementation of a graph database using zht.

[2] Christian Kohlschütter, Paul-Alexandru Chirita, and Wolfgang Nejdl. *Advances in Information Retrieval: 28th European Conference on IR Research, ECIR 2006, London, UK, April 10-12, 2006. Proceedings*, chapter Efficient Parallel Computation of PageRank, pages 241–252. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[3] T. Li, C. Ma, J. Li, X. Zhou, K. Wang, D. Zhao, I. Sadooghi, and I. Raicu. Graph/z: A key-value store based scalable graph processing system. In *2015 IEEE International Conference on Cluster Computing*, pages 516–517, Sept 2015.

[4] T. Li, X. Zhou, K. Brandstatter, D. Zhao, K. Wang, A. Rajendran, Z. Zhang, and I. Raicu. Zht: A lightweight reliable persistent dynamic scalable zero-hop distributed hash table. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 775–787, May 2013.

[5] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.