Adaptive Disk Storage for Interaction Graphs

1 Summary

Graph databases are a critical component for applications in social networking, telecommunications, and the world-wide web. They store entities as vertices, relationships as edges, and additional information in labels that may be associated with either vertices or edges. Because graph databases not only store the data, but also encode the structure, they allow for queries that are not possible in a traditional relational database.

encode the relationships between entities in their structure, they are ideal for use in social networking, telecommunications, or modeling the world-wide web.

An *interaction graph* is a graph database in which

the entities they model, they are

are becoming increasingly popular for applications in the world-wide web, social networks, and telecommunications. An interaction graph is an append-only graph, where new edges and vertices are added as time progresses. A good example of an interaction graph is Twitter, where users represent vertices, and edges are mentions of other users

In an interaction graph database organized as blocks of temporal neighborlists (as in Gedik et al. [3]), it is important to store the edge and vertex properties locally. For instance, if the edge properties are kept away in a relational table, there will be almost no benefit to the locality optimizations performed for block organization (discussed in Gedik et al. [3], as we would have to go back and forth between the disk blocks to access the edge attributes.

On the other hand, putting all the edge attributes into the disk blocks containing the graph structure is expected to cause significant overhead when only a few of these attributes are read. This is somewhat similar to the problem with row-oriented databases, where the entire row needs to be accessed from the disk despite the query needing only a small fraction of it. Unfortunately, there is no clear correspondence to a column-oriented database layout for the graph databases.

Recall that interaction graphs are temporal. As such, the co-access correlations for the attributes can be different for different temporal regions. It might also be unknown at the insertion time and may be discovered later depending on the workload. This points to the need for adaptively optimizing the layout (somewhat similar to the H2O [6]).

I imagine a solution to this problem, which I name the 'rail layout'. The idea is to start with large blocks that contain the entire data. As we learn about the access properties for different time regions, we might start splitting such blocks into smaller blocks that run parallel to each other, almost like having two or more graph databases for certain time regions, each containing a different subset of the attributes, but with a link between them in case a query needs to access both. Some attributes can be replicated as well. More details to follow about all this...

2 Project outline

2.1 Field of research

Graph Database work:

```
- Googles Pregel~[7]
- Apache Giraph~[1]
- GPS~[11]
- Trinity~[12]
- PowerGraph~[4]
- Prabhakaran et al.~[9]
- X-Stream~[10]
- Xie et al. ~[15]
- Neo4j ~[8]
- HyperGraphDB ~[5]
- Tao (Facebook)~[14]
- Kineograph~[2]
-  Titan ~[13]
```

Adaptive storage:

```
H2O: A Hands-free Adaptive Store~citealagiannis14
http://stratos.seas.harvard.edu/publications/h2o-hands-free-adaptive-store
```

2.2 Goals and objectives

2.3 Significance of the project

3 Work plan

The work described in this proposal consists of five threads of activity.

- *Define cost model*
- *Design partitioning algorithm*
- *Implement prototype*
- *Micro-benchmarks*
- *Real work applications*

References

[1] Apache giraph. http://giraph.apache.org/.

[2] R. Cheng, J. Hong, A. Kyrola, Y. Miao, X. Weng, M. Wu, F. Yang, L. Zhou, F. Zhao, and E. Chen. Kineograph: Taking the pulse of a fast-changing and connected world. pages 85–98, 2012.

[3] B. Gedik and R. Bordawekar. Disk-based management of interaction graphs. *IEEE Transactions on Knowledge and Data Engineering*, 10(1109), 2014.

[4] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. pages 17–30, 2012.

[5] Hypergraphdb. http://www.hypergraphdb.org/index.

[6] I. A. I, S. Idreos, and A. Ailamaki. ”h2o: A hands-free adaptive store”. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2014.

[7] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A System for Large-scale Graph Processing. pages 135–146, 2010.

[8] Neo4j. http://www.neo4j.org.

[9] V. Prabhakaran, M. Wu, X. Weng, F. McSherry, L. Zhou, and M. Haridasan. Managing large graphs on multi-cores with graph awareness. pages 4–4, 2012.

[10] A. Roy, I. Mihailovic, and W. Zwaenepoel. X-stream: Edge-centric graph processing using streaming partitions. pages 472–488, 2013.

[11] S. Salihoglu and J. Widom. Gps: A graph processing system. pages 1–12, 2013.

[12] B. Shao, H. Wang, and Y. Li. Trinity: A distributed graph engine on a memory cloud. pages 505–516, 2013.

[13] Titan. http://thinkaurelius.github.io/titan/.

[14] V. Venkataramani, Z. Amsden, N. Bronson, G. Cabrera III, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, J. Hoon, S. Kulkarni, N. Lawrence, M. Marchukov, D. Petrov, and L. Puzar. Tao: How facebook serves the social graph. pages 791–792, 2012.

[15] W. Xie, G. Wang, D. Bindel, A. Demers, and J. Gehrke. Fast iterative graph computation with block updates. *The VLDB Journal*, 6(14):2014–2025, Sept. 2013.