IOGP: An Incremental Online Graph Partitioning for Large-Scale Distributed Graph Databases

Dong Dai

Texas Tech University
dong.dai@ttu.edu

Wei Zhang

Texas Tech University
X-Spirit.zhang@ttu.edu

Yong Chen

Texas Tech University
yong.chen@ttu.edu

CCS Concepts • Information systems → Graph-based database models; DBMS engine architectures; Distributed storage

Keywords Graph Database; OLTP; Graph Partition

1. Introduction

Large-scale graphs databases are designed to serve continuous updates from the clients, and, at the same time, answer complex queries towards current graph in an on-line manner. This is different from another important set of tools, namely graph processing engines, which focus on performing complex analysis on graphs, like Pregel and GraphX. In fact, the boundary between graph databases and graph processing engines is a bit fuzzy. Most graph databases are capable of delivering graph computations through defining complex graph traversal, and, on the other hand, many graph computation engines also allow graphs to be updated and evolved like a database. In this study, we differentiate them according to the workloads they are optimized for. We consider graph databases are optimized for OLTP (online transaction processing) workloads like INSERT, UPDATE, GET, and TRVEL queries. Those operations should be performed in an interactive manner and expected to finish fast. On the other hand, graph processing engines are designed for OLAP (online analysis processing) workloads like running PageRank on the whole graph or finding the structure of social graph. Those differences lead to significantly different preferences in performance optimizations and also affect the choices of graph partitioning fundamentally.

First, since graph computation engines run analysis tasks on the whole graph which usually takes a long time, they are allowed to spend a long time in optimally partitioning the graphs. But, this is not the case for graph databases as

each transaction is normally short, has to finish fast and takes effect immediately. Second, in graph computation engines, when partition the graph, the graph structure (i.e., graph connectivity) should already be known. In fact, such information is critical to deliver an optimized partitioning in existing algorithms. For example, when a vertex is inserted, most of its connected edges should be known to place the vertex. However, such graph structures are not available to partitioning algorithms in graph databases, where vertex and its connected edges are inserted separately and independently. Third, graph processing engines mainly run analytic tasks applied on the whole graph, so they optimize for the best overall throughput. However, graph databases mostly serve single-point accesses on vertices and edges, or graph traversal starting from several vertices. This leads to different considerations on the efficiency of partitioning. Only considering the goals of minimizing “edge cuts” and maximizing “balance” among partitions, like most existing graph partitioning algorithms did, does not necessarily generate a good partition for graph databases. For example, the best partitioning on a graph that consists of k equal size disconnected subgraphs, would be putting each subgraph to one server to achieve the minimized ‘cut’ and best load balance. However, from graph databases’ perspective, if any of these subgraphs contains high-degree vertices, graph traversal will be significantly slow due to throughput bottleneck of single machine.

2. IOGP Algorithm

In this paper, we introduce a new graph partitioning solution, namely *Incremental Online Graph Partitioning*, for distributed graph databases. It consists of three stages: *quiet*, *vertex reassign*, and *edge splitting*.

Quiet Stage. It is the default stage. A new vertex will be placed into the server chosen by its hashed value; all edges will be placed together with their connected vertices. Hashing calculates fast, hence supports single-point accesses on vertices and edges with minimized overheads. But, it does not consider the locality affinity of vertices, which grows while more edges are inserted. IOGP solves this in *vertex reassign stage*. In addition, as vertices and their edges are placed together, this may lead to hot-spots if vertices are

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

PPoPP '17 February 04-08, 2017, Austin, TX, USA

Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4493-7/17/02.

DOI: <http://dx.doi.org/10.1145/3018743.3019037>

having too many edges, which, in fact, are commonly seen in power-law graphs. We tackle this in *edge splitting stage*.

Vertex Reassign Stage. In quiet stage, vertices have not gotten enough edges to expose connectivity information, hence random hashing is a good option. But, along with more edges are inserted, we should leverage connectivity information to re-assign the vertex to the best partition. The intuition behind vertex reassignment is straightforward: vertex should be moved to the partition that stores the most of its neighbors and all partitions should be balanced to avoid hotspot. To decide which partition is the best choice, we leverage the heuristic score defined in Fennel (Tsourakakis et al. 2014). Basically, for each vertex v , it assign a score for each partition p_i based on the number of v 's neighbors in p_i and also the total size of p_i . The best one should be chosen as the location. The key to calculate the heuristic score is to get the number of neighbors of v in a specific partition p_i . However, this is nontrivial while serving OLTP requests. A plain calculation needs to scan vertices stored in each storage servers, which is too time-consuming. In IOGP, we actually maintain such value on each partition, with controllable memory footprint, while new vertices and edges are inserted.

Edge Splitting Stage. In a power-law graph, vertex degree could be extremely large. This may lead to significant performance degradation in OLTP operations like scanning neighbors. We introduce the *edge splitting stage* to handle it. The core idea is to split edges of high-degree vertex into multiple servers to amortize the loads. To decide when to split a vertex, a threshold `MAX_EDGES` is defined. It limits the maximal edges of a vertex can have in one partition. If a vertex exceeds this number, it will switch to edge splitting stage. The splitting is quite simple: it will place an outgoing edge together with its destination vertex and place an incoming edge together with its source vertex. One key observation of such edge splitting is that the overall locality does not change since an edge is moved to either its source vertex or destination vertex, without gaining or losing any locality. However, the performance of scanning high-degree vertex will be significantly improved as these operations now can be carried out in parallel after splitting.

3. Evaluation Results

We evaluate the proposed IOGP graph partitioning algorithm using both real world and synthetic graphs. The evaluations were conducted on the CloudLab APT cluster with up to 32 nodes as the storage servers.

Evaluation Datasets. We use the SNAP dataset and also synthetic graphs generated using the RMAT graph generator. All datasets are power-law graphs.

Partition Quality. We compare IOGP and several state-of-art graph partition algorithms: METIS (Karypis and Kumar 1998), Fennel, and Hash on the quality of partitions. METIS is only ran once on the final graph. Fennel is run after we manually force a vertex and all its edges are inserted together. Hashing and IOGP are conducted in an online manner. The results are shown in Figure 1. The results show

the offline METIS achieves the best locality and Hash gives the worst partitioning; IOGP and Fennel are in between of them; and IOGP is better than Fennel in all tested graphs.

Figure 1. Edge cuts ratio comparison.

OLTP Performance. To show IOGP is a competitive alternation to Hash on OLTP workloads, we compare INSERT and TRAVEL operations in a 32-node cluster. Our results on INSERT performance show a maximal 10% overheads comparing to Hash, due to vertex reassignments and edge splitting in IOGP. In Figure 2, we show the travel performance of IOGP and Hash. All traversals are started from the same randomly chosen vertex. We evaluate 2, 4, 6, 8 steps. IOGP achieves clearly better traversal performance than Hash due to better locality. The performance gaps are also increasing while more traversal steps are needed.

Figure 2. Graph traversal performance.

4. Conclusion

In this study, motivated by the needs of large-scale distributed graph databases, we propose an incremental online graph partitioning algorithms optimized for OLTP workloads. The evaluation and comparisons show its performance advantages and indicate the possibility of using it as the default graph partitioning solution for graph databases.

References

- G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.
- C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic. Fennel: Streaming graph partitioning for massive scale graphs. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 333–342. ACM, 2014.