Summary on “A+ Indexes: Tunable and Space-Efficient Adjacency Lists in Graph Database Management System”

Yaling Qing (yq2565)

GDMBS is a contemporary DBMS that uses graphs to store data. A graph database consists of nodes and directed edges. Each node denotes an entity while each edge denotes the relationship between nodes. Each node and edge has a unique ID and some related attributes. Edge has an extra "label" field to indicate the relationship type. To process a query, the system will traverse specific paths in the graph.

Many popular commercial GDBMS, like Neoj4, GraphflowDB, use fixed adjacency list indexes to facilitate queries. It is achieved by allowing constant time access to adjacency edges of each node. The adjacency list indexes are built in 3 steps. 1) partitioning edges based on their source node or destination node ID into forward indexes and backward indexes. 2) further partitioning edges based on their labels. 3) Sorting partitioned edges based on either an edge attribute or neighbor node ID. We call the above method fixed adjacency list index because the property to partition edges on is fixed by the system. Although it can facilitate query, it is only highly efficient on physical data with some particular characteristics. The physical data dependency of this indexing method limits its efficiency.

In order to solve the physical data dependency problem in fixed adjacency list indexes, this paper introduces a tunable and space-efficient adjacency list index called A+ index. A+ index supports 3 indexing subsystem: 1) primary A+ indexes. 2) secondary vertex-partitioned A+ index 3) secondary edge-partitioned A+ index. I will explain each in the following.

Primary A+ Index is similar to fixed adjacency list indexes. It partitions the edges twice and sorts the most granular adjacency lists. Differently, it only fixes the property of the first partition. It allows customization on the the properties of 2nd partition and sorting. In another word, if many queries query on a specific edge attribute, the system can reconfigure indexes to conduct second partitioning on such attribute.

Example: e_adj = adjacent edge; v_nbr = neighbor node;
RECONFIGURE PRIMARY INDEXES
PARTITON BY *e_adj .label, e_adj .currency* **SORT BY** *v_nbr.city*

Primary A+ index provides a “global view” on adjacent edges of each node. However, it cannot facilitate queries with selection predicates. The 2 secondary indexes aim to solve this problem. **Secondary vertex-partitioned A+ index** provides 1-hop view on each node. A 1-hop view contains an adjacent edge when the edge itself or its connected nodes satisfy the customized selection predicates. The selected adjacent edges are first partitioned on node ID, and then partitioned and sorted based on primary A+ index config. Such vertex-partitioned index is to provide fast access to adjacency nodes.

Example: selection predicate is on “currency” & “amt”
CREATE 1−HOP VIEW *LargeUSDTrnx*
MATCH *vs−[e_adj]−>vd*
WHERE *e_adj .currency=USD, e_adj .amt>10000*
INDEX AS FW−BW
PARTITION BY *e_adj .label*   **SORT BY** *v_nbr.ID*

Secondary edge-partitioned A+ index provides 2-hop view on each edge. A 2-hop view contains a pair of edges when the two edges are adjacent to each other and they both satisfy the customized selection predicates. The selected edges are first partitioned on edge ID, and then partitioned and sorted based on primary A+ index config. Such edge-partitioned index is to provide fast access to adjacency edges.

Example:
CREATE 2−HOP VIEW *MoneyFlow*
MATCH *vs−[e_b]→vd−[e_adj]→v_nbr*
WHERE *e_b.date<e_adj .date, e_adj .amt<e_b.amt*
INDEX AS PARTITION BY *e_adj .label*   ***SORT BY*** *v_nbr.city*

The edges selected by secondary indexes are a subset of adjacency edges stored in the primary index list. To reduce memory overhead, secondary indexes use offset lists. Instead of simply storing edge ID in list, they store the edge offset where the corresponding primary index list the edge is on.

The paper uses 1) labelled subgraph queries; 2) recommendation; 3) financial fraud detection; tasks to evaluate the performance and memory overhead of A+ indexes. They found no matter tuning primary A+ index or adding any of the 2 secondary indexes, the system can achieve much better performance with no or minimum memory overhead

Reference:
Mhedhbi, Amine, et al. "A+ indexes: Tunable and space-efficient adjacency lists in graph database management systems." 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE, 2021.