The past decades have witnessed the massive growth of Internet. Vast amount of graph data was produced by the boom of social network, e-commerce and online education. To analyze and manage graph data, there are two branches of development. One focus on distributed graph computing. Solving problems like PageRank or other algorithms that fit into the Bulk Synchronous Parallel (BSP) model. Systems like Pregel and GraphLab are proposed for this branch. The other branch focus on management of graph data, providing support like OLTP and graph queries. In this branch, graph databases like Neo4j and Titan are developed for management of property graph.

The property graph is a directed, labeled graph with multi-edges, i.e., edges with the same source vertex and destination vertex. Vertices and edges can be associated with any number of properties. Since it can represent graph data in most scenarios, the property graph model has numerous applications in industry.

However, traditional graph databases encounter significant performance degradation when the graph contains large amount of multi-edges.

We reveal the cause of this in Titan. It’s an open source distributed graph database, which has attracted wide attention in industry. Furthermore, we propose HybriG, a better distributed architecture based on Titan and HBase for this scenario.

Titan stores graphs in adjacency list format, where a graph is stored as a collection of vertices with their adjacency lists. Each entry of the adjacency list stores an edge or a vertex property.

When querying about adjacent vertices, Titan has to look through the entire adjacency list of the source vertex, which is a waste since we just need the connected vertices but not the multi-edges. This cost hurts the performance when the multi-edge set grows explosively. For high level queries who bases on adjacent vertices query, e.g. path queries, local cluster coefficient queries etc., the situation will be worse.

HybriG implements property graph model as well. It stores the vertex data and graph structure in Titan, the edge data in HBase, respectively. We chose HBase since it’s one of the storage engines of Titan and is widely used in industry. Storing part of the graph data into HBase won’t bring too much cost because it’s also the data store of Titan.

This separation helps HybriG to keep a concise adjacency list about the graph structure, which helps to gain an order of magnitude improvement in execution of adjacent vertices based queries and batch loading of edge set. The difficulty of this separation solution is that we should guarantee the consistency of edge data between Titan and HBase.

In most of the scenarios with large multi-edges set, multi-edges are representing event-like data, e.g. phone call records between two people, which won’t be modified after insertion. Thus we can relax the consistency constrain of edge data to just guarantee consistency for insertion. We leverage the transaction in Titan to achieve consistency with HBase in edge insertion, especially in batch loading of edge data. Finally, extensive experiments have been conducted to show the outstanding performance of HybriG.