High Performance Distributed OLAP on Property Graphs with Grasper

Hongzhi Chen, Bowen Wu, Shiyuan Deng, Chenghuan Huang, Changji Li
Yichao Li, James Cheng
Department of Computer Science and Engineering
The Chinese University of Hong Kong
{hzchen,bwwu7,sydeng7,chhuang,cjli,liyc,jcheng}@cse.cuhk.edu.hk

ABSTRACT

Achieving high performance OLAP over large graphs is a challenging problem and has received great attention recently because of its broad spectrum of applications. Existing systems have various performance bottlenecks due to limitations such as low parallelism and high network overheads. This Demo presents *Grasper*, an RDMA-enabled distributed graph OLAP system, which adopts a series of new system designs to overcome the challenges of OLAP on graphs. The take-aways for Demo attendees are: (1) a good understanding of the challenges of processing graph OLAP queries; (2) useful insights about where Grasper's good performance comes from; (3) inspirations about how to design an efficient graph OLAP system by comparing Grasper with existing systems.

ACM Reference Format:
Hongzhi Chen, Bowen Wu, Shiyuan Deng, Chenghuan Huang, Changji Li and Yichao Li, James Cheng. 2020. High Performance Distributed OLAP on Property Graphs with Grasper. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3318464.3384685

1 INTRODUCTION

Graph data analytics has found many applications in industry nowadays. As a result, many graph processing systems have been proposed [9]. Most of these systems focus on offline graph computation workloads like PageRank, BFS and connected components, but relatively few are designed for online analytical query processing (OLAP). Existing graph databases such as Neo4J [4], Titan [1], JanusGraph [2], TigerGraph [6] and OrientDB [5] still have room to improve in terms of both latency (e.g., within milliseconds) and throughput (e.g., 10K QPS) for processing large graphs as we reported in [7]. These existing systems suffer from various performance problems due to the challenges of graph OLAP workloads listed as follows.

- **Diverse query complexity**: OLAP graph queries differ significantly in their complexity, e.g., from a simple filtering on the property of a vertex to complicated pattern matching. Thus, adaptive parallelism control and load balancing are needed for processing different queries.
- **Diverse data access patterns**: query operators (e.g., filter, traversal, count) usually have diverse access patterns on data, which require different optimization techniques (e.g., cache, indexes). It is challenging to design a unified execution framework that can support different query operations efficiently.
- **High communication and CPU costs**: a query may involve complex execution logics such as graph traversals that access a large portion of the graph, aggregation that collects intermediate results to one place through network, and joins that are CPU- and data-intensive. Such complex logics often result in high communication and computation overheads.

In this paper, we will demonstrate how to use Grasper [7], a high performance graph database, to process various OLAP workloads on large property graphs. We will show SIGMOD attendees how Grasper addresses the aforementioned challenges with its new system designs by leveraging Remote Direct Memory Access (RDMA). We design a visualization panel so that users can see the details of how Grasper constructs the optimized execution plan for each query, how the query engine processes incoming queries concurrently with adaptive parallelism control, and how Grasper's Expert Model uses tailored optimizations to execute different query operations. In addition, we will also demonstrate how RDMA effects the system designs of Grasper to achieve low latency. The Demo will explain (using effective visualizations) the

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3384685

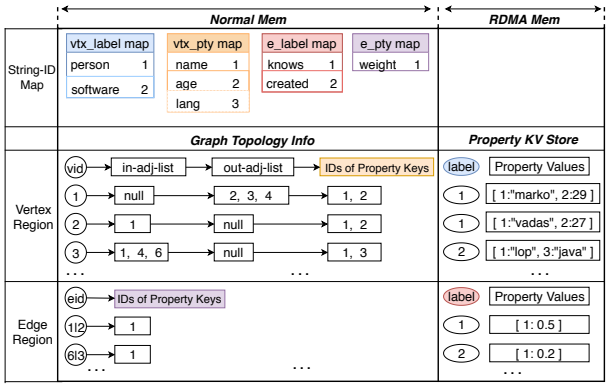

Figure 1: Data store in Grasper

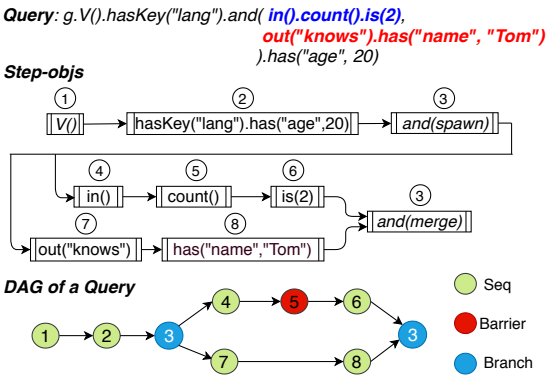

Figure 2: Query plan construction.

design details of Grasper's Expert Model and RDMA-aware features, thereby providing SIGMOD attendees better insights on how to design a high-performance distributed graph OLAP system.

2 AN OVERVIEW OF GRASPER

We introduce the key designs and system components of Grasper. Details can be found in Grasper's full paper [7] and homepage (http://www.cse.cuhk.edu.hk/systems/grasper/).

2.1 Design Goals

Grasper is designed for high-performance OLAP on large property graphs. It uses RDMA to reduce the high network communication cost and supports an RDMA-enabled native PG storage. In designing Grasper, we have the following goals in order to address the limitations of existing systems: (1) We should design an efficient query execution model to guarantee high utilization of CPU and network on various kinds of OLAP workloads. (2) Due to the diversity of graph queries, we should support both *high parallelism* (within a machine and across machines) for processing heavy-workload queries and *high concurrency* for processing as many queries simultaneously as possible. (3) We should avoid using external databases or data stores, but design an integrated data store tightly connected with the query execution engine to eliminate the overheads of moving data from one system to another. (4) The design of the data store should be native for graph storage in order to support efficient graph operations such as traversals. (5) The designs of the data store as well as other system components should be RDMA-friendly.

2.2 Data Store

We use the **property graph (PG)** model to represent graph data, where each vertex/edge in a PG can have a label and a set of properties to describe itself. We divide the in-memory space of each Grasper server into two parts: *Normal Memory* and *RDMA Memory*, as shown in Figure 1. Normal Memory is used to store those graph topology information that will be accessed locally during query execution. RDMA Memory is used to store those properties, in the form of key-value store, which can be remotely accessed by other Grasper servers.

In such a hybrid data storage design, we use index-free adjacency lists to support efficient traversal-based query operations and use RDMA-enabled KVS to achieve low-cost remote access on labels/properties through one-sided RDMA READ. Each query engine only processes queries on its own local partition, while cooperating with other remote engines by *messages* sent by one-sided RDMA WRITE.

2.3 Query Plan

Grasper adopts Gremlin [3] as its query language but constructs its own query execution plan. To enable adaptive parallelism control at query-step level inside a query, we define a new concept, **Flow Type**, to describe the parallel data flow pattern of each query step according to its functionality. There are three flow types:

- **Sequential**: receive and process one message at a time, and then proceed to the next step directly. *Sequential* steps can be processed in parallel safely.
- **Barrier**: collect messages from the previous steps and performs a functional aggregation on the collected data before proceeding to the next step.
- **Branch**: receive and process one message, then copy and send the newly generated message to all subqueries, which can be processed concurrently.

Grasper follows a tree-based parsing rule to recursively parse the query string step by step and construct the logical query plan according to three optimization rules: (1) Decomposition: to decompose branch-type query steps and process them in parallel. (2) Combination: to combine contiguous filter-based steps on vertices/edges/properties into one single step to reduce the overheads of multiple scanning and filtering. (3) Reordering: to reorder query steps with specific constraints and move them to the front of the query so as to

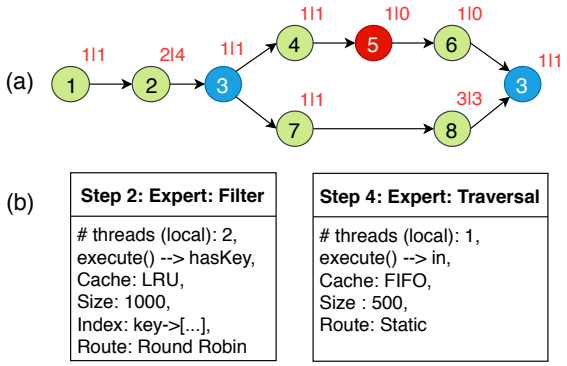

Figure 3: (a) adaptive parallelism at query-step level; (b) two expert examples for the query in Figure 2

generate less intermediate results. Based on the dependency and order of each query step, we generate a directed acyclic graph (DAG) to represent the query's execution plan, as the example shown in Figure 2.

2.4 Expert Model

Expert Model is the central idea of Grasper, which provides a top-down query-specific mechanism to achieve low latency and high throughput with high utilization of CPU and network. It has the following three features: (1) adaptive parallelism control at the query-step level inside each query; (2) tailored optimizations for various query steps according to their own data access patterns; (3) locality-aware thread binding and load balancing. In particular, we integrate these features for each category of query steps (e.g., traversal-based query operations, filter-based query operations) into one **expert**, which is the physical query operator that expertly handles the processing of the specific category. Note that each category of query steps (e.g., *has*, *hasKey* and *hasValue*) share similar functionalities.

Each expert maintains its own data structures (e.g., indexes, cache) for tailored optimizations, its own *execute()* function, and its own routing rules for out-going messages, to handle the concurrent processing of its query steps. Note that an expert may employ multiple threads to concurrently process the query steps of multiple queries with shared optimizations. Feature (1) in Expert Model ensures that the runtime parallelism degree of an expert is adaptive according to its load. Feature (2) in Expert Model improves the efficiency of query processing, and Feature (3) handles the underlying cache locality and load balancing among the threads physically, which are critical for achieving millisecond-level query latency. Figure 3 shows how Expert Model looks like based on the query in Figure 2, where Figure 3(a) depicts the parallelism degree of each query step in the DAG in two query engines and Figure 3(b) gives more runtime details of the two experts at step 2 and step 4.

Table 1: Query latency (in *msec*)

LDBC	IC1	IC2	IC3	IC4	IS1	IS2	IS3	IS4
Grasper	271	16.7	388	77.3	0.30	2.19	0.91	0.32
Titan	66985	13585	5.8E5	11947	0.71	25.9	2.88	1.32
J.G.	56206	9223	4.5E5	22420	0.83	14.5	2.99	1.17
AMiner	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**
Grasper	0.17	0.42	17.3	45.2	104	28.8	2.32	4.41
Titan	1.07	12.4	32341	2.1E5	43809	234	9.11	84.08
J.G.	1.34	8.70	27466	2.4E5	39155	276	5.61	84.71

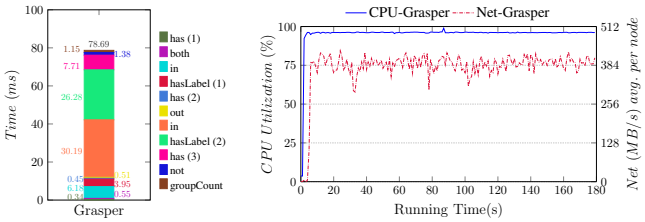

Figure 4: (a) Query latency breakdown of Grasper; (b) CPU and network utilization of Grasper

3 THE DEMONSTRATION PLAN

In this demo, we plan to show SIGMOD attendees: (1) the superior performance of Grasper in terms of query latency and system throughput on various graph OLAP workloads; (2) a real-time monitor to display Grasper's runtime system status and resource utilization, in order to demonstrate where Grasper's high performance comes from; and (3) intuitive visualization to show Grasper's real-time performance and how Expert Model functions on each query to expertly process different types of query steps.

Benchmarks and Set-up. We will use LDBC SNB [8] (i.e., IS1-IS4, IC1-IC4) and the benchmark (i.e., Q1-Q8) proposed in [7] to evaluate Grasper on three large graph datasets. We will deploy Grasper servers on a remote cluster and connect it to a web-based front-end to support user interaction with Demo attendees. The details of the benchmarks and cluster setup are given in [7].

3.1 System Comparison

This part of the Demo will present the performance of Grasper comparing with existing systems (e.g., Neo4J [4], Titan [1], JanusGraph [2], TigerGraph [6], OrientDB [5], etc.) on single-query latency as well as batch-processing throughput. Table 1 briefly shows the query latency of Grasper comparing with Titan and JanusGraph on two benchmarks (more evaluation results are reported in [7]). We can observe that when queries are more complex, Grasper's performance advantages are more obvious. Our Demo will aim to explain (with visualized details) where the performance benefits of Grasper come from by showing how the existing systems' limitations and performance bottlenecks are addressed by Grasper's system designs. For example, Figure 4 provides some insights of Grasper's performance: (a) step-level latency breakdown of

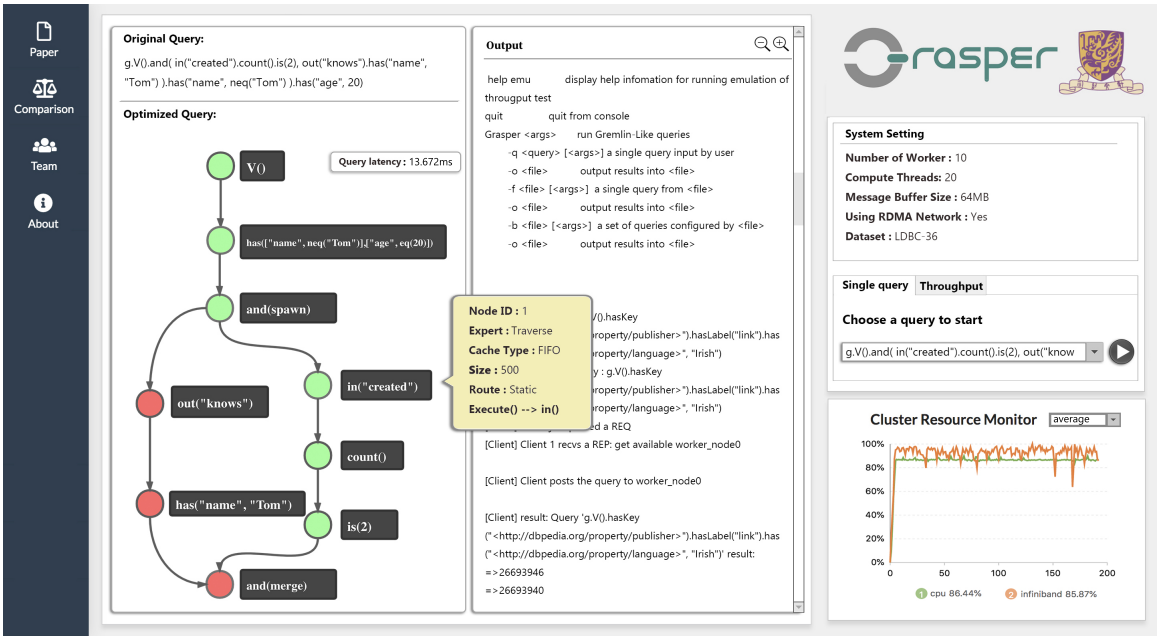

Figure 5: A screenshot of the Grasper Demo front-end

an example query and (b) cluster CPU & network utilization running on a mixed workload. We will also show these statistics together with those of the other systems to Demo attendees to explain the necessary designs of an efficient graph OLAP system.

3.2 Grasper's Front-End Interface

The Demo system will provide a graphical interface for allowing attendees to interact with Grasper, so as to help them understand the key design idea of Grasper and how Expert Model works with the exact optimization set-up to speed up the query processing for specific query steps. As shown in Figure 5, on the right control panel, attendees can input a single query or a batch of queries in a file for batch-processing.

For single-query processing, we will visualize the DAG of current query to show the specific optimized query plan with detailed runtime information (e.g., the parallelism degree of each query step on all workers, the tailored optimizations applied in corresponding experts, the message routing strategy, etc.), and dynamically update the job progress of current query in real time (marked as green/red for ongoing/finished steps) to illustrate the mechanism and effectiveness of Expert Model. When the current query is completed, we will report its latency and also return the final query results from the server side to the front-end's console to attendees for better understanding of the complexity of one OLAP query on a large property graph.

In batch processing mode, another panel will show up to display the real-time throughput of the whole system, which provides an intuitive view of the performance of Grasper throughout the process of query processing. We will also display a detailed view of the current system configuration on the top of the control panel, where attendees can manually configure the system setting (e.g., enable/disable core binding, cache, etc.) to observe the performance benefits brought from various system components, optimization techniques, and their respective trade-offs (if any). In addition, the display panel will also visualize the real-time CPU and network utilization of the cluster occupied by Grasper servers, to demonstrate how efficient Grasper is in terms of resource utilization.

Acknowledgments. We thank the reviewers for their valuable comments. This work was supported in part by ITF 6904945 and GRF 14222816.

REFERENCES

[1] 2015. TITAN. http://titan.thinkaurelius.com/.
[2] 2017. JanusGraph. http://janusgraph.org/.
[3] 2019. Gremlin. http://tinkerpop.apache.org/gremlin.html.
[4] 2019. Neo4j. https://neo4j.com/.
[5] 2019. OrientDB. https://orientdb.com/.
[6] 2019. TigerGraph. https://www.tigergraph.com/.
[7] Hongzhi Chen, Changji Li, Juncheng Fang, Chenghuan Huang, James Cheng, Jian Zhang, Yifan Hou, and Xiao Yan. 2019. Grasper: A High Performance Distributed System for OLAP on Property Graphs. In *ACM Symposium on Cloud Computing*. 87–100.
[8] Orri Erling, Alex Averbuch, Josep-Lluís Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat-Pérez, Minh-Duc Pham, and Peter A. Boncz. 2015. The LDBC Social Network Benchmark: Interactive Workload. In *SIGMOD*. 619–630.
[9] Da Yan, Yuanyuan Tian, and James Cheng. 2017. *Systems for Big Graph Analytics*. Springer. https://doi.org/10.1007/978-3-319-58217-7