Supplemental material for the evaluation based on LinGBM

1 Evaluation based on LinGBM

To show the generalizability of our system, we conduct an evaluation based on LinGBM, a performance benchmark for GraphQL server implementations. LinGBM provides tools for generating datasets (data generator) and queries (query generator), and for testing execution time and response time (test driver). The generated dataset is a scalable and synthetic dataset regarding the *University* domain, including several entity types (e.g., universities and departments). We generate data in scale factors (*sf*) 4, 20 and 100. We then create three MySQL database instances to store the data in such three scale factors, respectively. We use a modified version of the GraphQL schema provided by LinGBM for our GraphQL server, and define RML mappings according to the work in morph-graphql. The modification part is regarding input object type definitions so that we can use input objects to represent filtering conditions. The experiments are performed over eight query sets, where each set contains 100 queries that are generated using the query generator based on a query template (QT). A query template has placeholders where each placeholder represents that an input argument can be assigned. The query generator can generate a set of actual queries (query instances) based on a query template where the placeholder in the query template is replaced by an actual value. We select eight query templates (QT1–QT6, QT10 and QT11) for constructing these eight query sets (QS1–QS8). The other six query templates from LinGBM requires GraphQL servers to have implementations for functionalities such as ordering and paging which are not considered currently by OBG-gen. However, these functionalities are interesting for future extension of OBG-gen. Same as the real case evaluation, we evaluate the query execution time (QET) of our system on the three datasets. Each query from a query set is evaluated once. We show the average query execution time for the different query sets in Table 1. Based on the obtained measurements, we observe that our system has slight increases for QS1, QS2, QS4, QS6 and QS7 in terms of the average QETs. For QS3, the average QET is stable for all the three datasets. For QT5, the increase from 0.51 seconds at data scale factor 20 to 13.85 seconds at data scale factor 100 is due to the dramatic increase in result size. More specifically, the queries in QS5 and QS8 need to access the *'graduateStudent'* table which increases dramatically in size from 50,482 (sf=20) to 252,562 (sf=100). This is the reason for the average QET of QS8 increasing in sf=100. Additionally, each query in QS5 repeats a cycle two times (*'university'* to *'graduateStudent'* to *'university'*) and requests the students' emails and addresses along the way. This causes the larger increase in average QET of QS5. The above synthetic experiments indicate that our system can work in a general domain.

Table 1: Average QET (in seconds).

sf	QS1 (QT1)	QS2 (QT2)	QS3 (QT3)	QS4 (QT4)	QS5 (QT5)	QS6 (QT6)	QS7 (QT10)	QS8 (QT11)
4	0.11	0.13	0.12	0.15	0.19	0.13	0.10	0.26
20	0.12	0.15	0.12	0.18	0.51	0.15	0.18	0.90
100	0.15	0.27	0.12	0.26	13.85	0.23	0.72	4.41