Full Disclosure Report of the LDBC Social Network Benchmark

Audit of the LDBC Social Network Benchmark's Business Intelligence Workload over TuGraph

December 3, 2023

General Terms

Executive Summary

This report documents an audited execution of the LDBC SNB BI (Social Network Benchmark Business Intelligence) workload for TuGraph[1], from Ant Yunchuang Digital Technology (Beijing) Co., Ltd.

TuGraph is a propietary graph compute system written in Java. This document describes an implementation of the LDBC Social Network Benchmark Business Intelligence workload using TuGraph. This implementation makes use of Gremlin. This system runs distributed and uses hash sharding as a partitioning strategy. In this audit, the system was deployed distributed using Kubernetes and ran on the Aliyun cloud[2]. The system under test and the driver communicate using remote procedure calls (RPC), implemented in Java.

Declaration of Audit Success

This report contains details of a successful execution of the LDBC SNB BI benchmark. The results have been gathered by an independent auditor who has validated the implementation of the queries and verified the system's configuration conforms to the description of the benchmark and its strict requirements.

Sponsorship and Funding Disclaimer

TuGraph, as an LDBC member, are the Test Sponsor of this audit. The audit and its associated execution costs are funded by TuGraph.

[1] https://www.tugraph.org/

[2] https://alibabacloud.com

DocuSigned by:
David Puroja
C7FE4F71E2554AE...

12/4/2023

Mr. David Püroja
(Auditor)

Date

Pometry
(Audit Company)

DocuSigned by:
Gabor Szarnyas
FA33C2C8CB944A5...

12/3/2023

Dr. Gábor Szárnyas
(Leader of LDBC SNB Task Force)

Date

DocuSigned by:
Chuntao Hong
3FE773CDAEA8496...

12/4/2023

Dr. Chuntao Hong
(Test Sponsor Representative)

Date

Table of Contents

1 Benchmark Description

The audit was conducted in compliance with the Social Network Benchmark Business Intelligence workload's specification.

Table 1.1: Benchmark Overview

Artifact	Version	URL
Specification	2.2.0	https://arxiv.org/pdf/2001.02299v7.pdf
Data generator	0.5.1	https://github.com/ldbc/ldbc_snb_datagen_spark/releases/tag/v0.5.1
Driver and implementations	1.0.3	https://github.com/ldbc/ldbc_snb_bi/releases/tag/v1.0.3

2 System Description and Pricing Summary

2.1 Details of machines driving and running the workload

2.1.1 Machine overview

The audit was conducted using multiple virtual machines in the Aliyun Cloud. For SF30000, 1 virtual machine was used for the driver and 72 machines for the database instance. The machine types are shown below. The details below where obtained from the Aliyun console.

Table 2.1: Machine Type and Location

Cloud provider	Aliyun Cloud
Machine region	China (Hangzhou)
Common name of the item (worker)	72 ecs.r7.16xlarge (64 vCPU 512 GiB) with ESSD AutoPL (1024GB)
Common name of the item (driver)	1 ecs.g6.2xlarge (8 vCPU 32 GiB) ESSD P0 (capacity 80G)
Operating system	Alibaba Group Enterprise Linux Server release 7.2 (Paladin)
Kernel version	4.19.91-27.4.al7.x86_64

2.1.2 CPU details

The details below were obtained using the command `lscpu` (Listing A.1) issued from the machine instance.

Table 2.2: CPU details summary

Type	Intel® Xeon® Platinum 8369B
Total number	1
Cores per CPU	32
Threads per CPU core	2
CPU clock frequency	2.70 GHz
Total cache size per CPU	L1i cache: 32KiB L1d cache: 48KiB L2 cache: 1280KiB L3 cache: 48MiB

2.1.3 Memory details

The total size of the memory installed on the worker machine is 512GiB and the type of memory is DDR4 with frequency 3200MHz. This information was obtained using the `sudo lshw -c memory` command (Listing A.3) issued from the virtual machine instance.

2.1.4 Disk and storage details

The virtual machine instance used Aliyun ESSD AutoPL storage, formatted using the `ext4` file system.

Disk controller or motherboard type was not obtainable from the virtual machine instance. Information on Alibaba Cloud enhanced SSDs can be found on the website `https://www.alibabacloud.com/help/en/ecs/user-guide/essds` and more information on ESSD AutoPL disks can be found at `https://www.alibabacloud.com/help/en/ecs/user-guide/essd-autopl-disks?spm=a2c63.p38356.0.0.1013669544yRnw` (accessed: November 15, 2023).

The 4KB QD1 write performance on the data disk was measured with the `fio` command and the output (Listing B.1) showed an average of $1,718$ IOPS.

2.1.5 Network details

Each machine is equiped using two 10Gb virtual network devices. No performance information could be obtained.

2.1.6 Machine pricing

The system pricing summary in US dollars (USD) is included in the table below. The pricing of the Aliyun machine instance is the price for a 3-year reserved dedicated instance machine without upfront payment, including a 1024GiB ESSD AutoPL System disk [1].

Table 2.3: Pricing summary

Item	**Price**
ecs.r7.16xlarge reserved instance machine in Aliyun (standard 3-year term)	4,165,500 USD
Software license (3 years)	1,060,000 USD
Maintenance fee (3 years)	424,000 USD
Total cost	5,649,500 USD

2.1.7 System version and availability

Table 2.4: System versions

System	**Version**	**License**
TuGraph-Analytics Enterprise	v0.9	Enterprise Licence provided by TuGraph

[1] `https://www.alibabacloud.com/product/ecs-pricing-list/en#/?_k=m5ooui`

3 Dataset Generation

3.1 General information

The data generation settings of the LDBC Datagen are described below.

Table 3.1: Datagen settings summary

Data format for TuGraph	composite-projected-fk layout, compressed CSV files
Scale factors for TuGraph	10 (validation) and 30,000 (benchmark)
Data format for Umbra	composite-merged-fk layout, compressed CSV files
Scale factors for Umbra	10 (validation)

3.2 Datagen configurations

The datasets and query substitution parameters used for the benchmark and cross-validation runs were retrieved from the following URLs and copied to an Aliyun Storage Bucket, with the `tar.zst` files uncompressed. The URLs are served by LDBC's official data repository, available as a public bucket in the Cloudflare R2 object storage.[1]. The substitution parameters for SF30000 is provided by LDBC and the dataset was generated on Aliyun, with checking of the number of nodes and entities afterwards to verify the validity.

3.2.1 SF10

- `https://pub-383410a98aef4cb686f0c7601eddd25f.r2.dev/bi-pre-audit/parameters-2022-10-01.zip`
- `https://pub-383410a98aef4cb686f0c7601eddd25f.r2.dev/bi-pre-audit/bi-sf10-composite-merged-fk.tar.zst`
- `https://pub-383410a98aef4cb686f0c7601eddd25f.r2.dev/bi-pre-audit/bi-sf10-composite-projected-fk.tar.zst`

To re-generate these datasets from scratch, use the instructions provided in Appendix C.

3.3 Data loading and data schema

The data is loaded using a Gremlin/SQL script, using the inner bulk load capabilities. An example for loading the City table is shown in Listing 3.1.

```
create table file_City (
  id BIGINT,
  name VARCHAR,
  url VARCHAR,
  type VARCHAR
) with (
   input.type='file',
   file.path='$dataDir/initial_snapshot/static/Place/*.gz',
   skip.header='true',
   column.separator='|'
);

insert into ldbc_bi_graph(City.id, City.name, City.url)
select id, name, url from file_City where type = 'City';
```

Listing 3.1: Script to load the data on TuGraph

The data preprocessing and loading times are reported below. Values were measured using the GNU Time tool (`/usr/bin/date`) with the `+%s.%3N` formatting applied to the timestamps (yielding millisecond precision).

[1] `https://www.cloudflare.com/products/r2/`

Using a Python script the load time was calculated by substracting the end time from the start time. The column **Data preprocessing time** shows how much time it took to precompute the root post edges. More information on the precomputed auxiliary data structures is provided in Section 4.2. For this benchmark execution, the preprocessing only consisted of decompressing the `.csv.gz` files. The column **Data loading time** shows how long it took to create a graph from the input CSV files and perform the initial indexing, compilation of the queries and precomputation. The decomposition of the loading time is output from the `ddl/setup.sh` script during loading the data, which uses the `$SECONDS` variable from bash to calculate elapsed time.
The column **Total time** contains the sum of the data preprocessing and loading times.

The TuGraph data schema is shown in Listing D.1.

Table 3.2: Data preprocessing and loading times for TuGraph

Scale factor	Data preprocessing time (s)	Data loading time (s)	Total time (s)
30,000	660	5,761	6,421

4 Implementation Details

4.1 Execution mode

Section 7.5.2.2 of the SNB specification defines two execution modes for the *throughput batches*. In *disjoint read-write mode*, the updates for each day of the benchmark's simulation are applied in bulk, separately from the read queries (i.e. there are no overlapping read and write operations). In *concurrent read-write mode*, the updates are applied concurrently with the reads. Systems opting for concurrent read-write mode are subject to the LDBC ACID test[1].

In the current audited run, TuGraph was executed using the *disjoint read-write mode*. Therefore, no ACID tests were conducted.

4.2 Use of auxiliary data structures

The TuGraph implementation precomputes the following auxiliary data structures. These are executed in each batch after the writes have been applied. The root post is computed after loading the data and after each batch.

- **Root Post:** For each Message node (Comments and Posts), an edge to the corresponding Message thread's root Post is inserted. These derived edges are maintained incrementally, i.e. root Post edges are inserted for newly inserted Messages and removed for deleted Messages.
- **Q4:** For each Forum, the maximum number of members (for number of members per country) is precomputed.
- **Q6:** For each Message, the popularityScore defined in the query is precomputed.
- **Q19:** The weight attributes on the knows edges are precomputed based on the number of interactions between the two Person nodes.
- **Q20:** The weight attributes on the knows edges are precomputed based on the classYear attributes on the studyAt edges that point to the same University from the endpoint Person nodes.

We display the runtime of these operations in Table 5.3.

[1] https://github.com/ldbc/ldbc_acid

4.3 Benchmark execution

The benchmark is executed using the following commands:

```
# Change in vars.sh
# export SF=30000
# export NUM_NODES=72
# export PARTITION=1151

# setup cluster.
nohup sh scripts/deploy.sh > deploy.log 2>&1 < /dev/null &
tail -f deploy.log

# run benchmark
nohup sh scripts/run-benchmark.sh 1>benchmark.log 2>&1 < /dev/null &
tail -f benchmark.log
```

Listing 4.1: Script to execute the benchmark on TuGraph for SF30000

5 Performance Results

5.1 TuGraph performance results

Table 5.1: Summary of results for TuGraph on scale factor 30000

Benchmark duration	Power@SF	Power@SF/$	Throughput@SF	Throughput@SF/$
1,390.44 minutes	111,775.39	19.79	56,920.48	10.08

Table 5.2: Detailed **power test results** for TuGraph on scale factor 30000. Execution times are reported in seconds.

Query	Count	Min.	Max.	Mean	P_{50}	P_{90}	P_{95}	P_{99}
1	30	10.006	46.945	13.876	11.465	17.599	29.982	46.945
2a	30	19.473	39.527	22.111	20.803	22.248	39.352	39.527
2b	30	19.438	20.894	20.066	20.033	20.542	20.752	20.894
3	30	111.490	120.688	114.098	113.684	116.102	117.309	120.688
4	30	30.479	39.888	33.062	32.628	34.407	34.629	39.888
5	30	10.705	24.277	13.851	12.899	18.116	21.312	24.277
6	30	18.967	26.618	21.416	21.213	23.877	24.553	26.618
7	30	29.157	61.238	44.111	46.360	53.832	56.712	61.238
8a	30	20.593	25.325	23.180	23.209	24.519	24.894	25.325
8b	30	19.948	25.289	21.943	21.740	23.299	23.874	25.289
9	30	21.178	59.418	23.957	22.652	23.420	26.736	59.418
10a	30	33.282	91.488	63.029	62.298	81.601	84.688	91.488
10b	30	12.914	24.357	17.018	16.209	21.364	22.537	24.357
11	30	19.654	25.358	22.634	22.728	24.574	24.855	25.358
12	30	106.256	182.534	149.131	139.850	174.277	178.668	182.534
13	30	118.240	181.585	123.831	121.104	124.730	134.566	181.585
14a	30	74.034	88.035	77.159	76.383	79.252	85.236	88.035
14b	30	14.662	19.662	17.104	17.030	18.554	19.274	19.662
15a	30	57.222	64.553	61.529	61.882	63.070	64.272	64.553
15b	30	86.809	140.374	110.683	101.772	131.029	133.596	140.374
16a	30	30.104	86.644	41.485	38.005	48.876	64.783	86.644
16b	30	22.106	28.526	25.122	25.233	27.395	27.948	28.526
17	30	22.341	32.743	27.288	27.547	29.898	31.131	32.743
18	30	23.953	31.953	25.893	25.345	28.549	28.889	31.953
19a	30	17.621	22.521	20.791	20.952	22.018	22.143	22.521
19b	30	18.794	22.325	20.489	20.320	21.494	21.805	22.325
20a	30	4.519	26.139	7.723	6.553	10.638	13.414	26.139
20b	30	7.011	8.497	7.275	7.121	7.723	7.811	8.497

Table 5.3: Operations in the **power test** for TuGraph on SF30000. Execution times are reported in seconds. Root Post precomputations are performed for each Comment insertion and deletion operation, therefore, they are reported as part of the writes.

Operation	Time
reads	35,096.512
writes	6,136.150
q4_precomputation	748.223
q6_precomputation	122.918
q19_precomputation	4,787.131
q20_precomputation	32.374

6 Validation of the Results

The results were cross-validated against the Umbra reference implementation[1] on scale factor 10, using Umbra version `664890d7f`. Umbra is an in-memory relational database management system developed at the Technische Universität München. The queries of the BI workload are implemented using PostgreSQL-compatible SQL queries that use the `WITH RECURSIVE` clause to implement graph traversal operations.

Listing 6.1: Output of the Umbra–TuGraph cross-validation command

```
$ export SF=10
$ scripts/cross-validate.sh umbra tugraph

+++  Files "umbra/output/output-sf10/results.csv" and "tugraph/output/output-sf10/results.csv" are equal
```

6.1 Number of entities after load

To verify the number of entities loaded before starting the benchmark, the script below was used. The queries are available in the supplementary package.

Listing 6.2: Executing the statistics script

```
nohup sh scripts/data-statistics.sh 1>statistics.log 2>&1 < /dev/null &
```

The number of entities aligned with the specification.

[1] https://github.com/ldbc/ldbc_snb_bi/tree/a2d3ac18c946d6a698c6aa5e6cf5d8954296be63/umbra

7 Supplementary Materials

Table 7.1: Supplementary materials

File or Directory	Purpose
umbra/output/output-sf10	Output of the Umbra reference implementation
parameters/parameters-sf{10,30000}.tar	Query substitution parameters
scoring/calculate-scores.py	Python script to calculate the scores of the benchmark run
tugraph/output/output-sf{10,30000}	Benchmark logs and outputs
ldbc_snb_bi-1.0.3.tar.gz	Benchmark driver and reference implementations
ldbc_snb_datagen_spark-0.5.1.tar.gz	Data generator
ldbc_snb_specification-2.2.0.pdf	Benchmark specification
tugraph-ldbc-bi-master.zip	Repository with TuGraph implementation
tugraph-encrpt-0.9.jar	TuGraph Binary

The ldbc_snb_bi_tugraph_sf30000_attachments folder's directory structure is as follows:

```
ldbc_snb_bi_tugraph_sf30000_attachments
├── umbra
│   └── output
│       └── output-sf10
│           └── results.csv
├── parameters
│   ├── parameters-sf10.tar.gz
│   └── parameters-sf30000.tar.gz
├── scoring
│   └── calculate-scores.py
├── tugraph
│   └── output
│       ├── output-sf10
│       │   ├── benchmark.csv
│       │   ├── load.csv
│       │   ├── results.csv
│       │   └── timings.csv
│       └── output-sf30000
│           ├── benchmark.csv
│           ├── load.csv
│           ├── results.csv
│           └── timings.csv
├── ldbc_snb_bi-1.0.3.zip
├── ldbc_snb_datagen_spark-0.5.1.zip
├── ldbc_snb_specification-2.2.3.pdf
├── tugraph-ldbc-bi-master.zip
└── tugraph-encrpt-0.9.jar
```

A CPU and Memory details

Listing A.1: Output of the `lscpu` command

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                64
On-line CPU(s) list:   0-63
Thread(s) per core:    2
Core(s) per socket:    32
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 106
Model name:            Intel(R) Xeon(R) Platinum 8369B CPU @ 2.70GHz
Stepping:              6
CPU MHz:               2699.998
BogoMIPS:              5399.99
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             48K
L1i cache:             32K
L2 cache:              1280K
L3 cache:              49152K
NUMA node0 CPU(s):     0-63
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr
    sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid tsc_known_freq pni
    pclmulqdq monitor ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c rdrand hypervisor
    lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single ibrs_enhanced fsgsbase tsc_adjust bmi1 avx2 smep bmi2
    erms invpcid avx512f avx512dq rdseed adx smap avx512ifma clflushopt clwb avx512cd sha_ni avx512bw avx512vl
    xsaveopt xsavec xgetbv1 xsaves wbnoinvd arat avx512vbmi pku ospke avx512_vbmi2 gfni vaes vpclmulqdq
    avx512_vnni avx512_bitalg avx512_vpopcntdq rdpid fsrm arch_capabilities
```

Listing A.2: Output of the `cat /proc/meminfo` command

```
MemTotal:       519666900 kB
MemFree:        513586136 kB
MemAvailable:   513422012 kB
Buffers:           413516 kB
Cached:           2918080 kB
SwapCached:             0 kB
Active:           1896928 kB
Inactive:         2484228 kB
Active(anon):        3544 kB
Inactive(anon):   1048372 kB
Active(file):     1893384 kB
Inactive(file):   1435856 kB
Unevictable:            0 kB
Mlocked:                0 kB
SwapTotal:              0 kB
SwapFree:               0 kB
Dirty:               1244 kB
Writeback:              0 kB
AnonPages:         961072 kB
Mapped:            461000 kB
Shmem:               2328 kB
```

```
Slab:             726288 kB
SReclaimable:     471416 kB
SUnreclaim:       254872 kB
KernelStack:       25088 kB
PageTables:        15140 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    259833448 kB
Committed_AS:     6571804 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
Percpu:            43776 kB
HardwareCorrupted:     0 kB
AnonHugePages:    401408 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
FileHugePages:         0 kB
FilePmdMapped:         0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
DirectMap4k:      512700 kB
DirectMap2M:    69724160 kB
DirectMap1G:    460324864 kB
```

Listing A.3: Output of the `lshw -C memory` command with the first bank out of 32

```
*-memory
     description: System Memory
     physical id: 1000
     size: 512GiB
     capabilities: ecc
     configuration: errordetection=multi-bit-ecc
   *-bank:0
        description: DIMM RAM
        vendor: Alibaba Cloud
        physical id: 0
        slot: DIMM 0
        size: 16GiB
   ----
```

B IO PERFORMANCE

Listing B.1: Output of the `fio` command

```
[root@ldbc30k003 home]# fio --rw=write --ioengine=sync --fdatasync=1 --direct=1 --directory=io-test-data --size=2
    g --bs=4k --name=iotest
iotest: (g=0): rw=write, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=sync, iodepth=1
fio-3.7
Starting 1 process
iotest: Laying out IO file (1 file / 2048MiB)
Jobs: 1 (f=1): [W(1)][100.0%][r=0KiB/s,w=6826KiB/s][r=0,w=1706 IOPS][eta 00m:00s]
iotest: (groupid=0, jobs=1): err= 0: pid=377719: Mon Oct 30 21:19:23 2023
  write: IOPS=1718, BW=6873KiB/s (7038kB/s)(2048MiB/305114msec)
    clat (usec): min=106, max=5574, avg=163.48, stdev=123.47
     lat (usec): min=106, max=5574, avg=163.53, stdev=123.47
    clat percentiles (usec):
     |  1.00th=[  115],  5.00th=[  119], 10.00th=[  123], 20.00th=[  127],
     | 30.00th=[  131], 40.00th=[  135], 50.00th=[  137], 60.00th=[  143],
     | 70.00th=[  147], 80.00th=[  157], 90.00th=[  188], 95.00th=[  273],
     | 99.00th=[  742], 99.50th=[  930], 99.90th=[ 1582], 99.95th=[ 1958],
     | 99.99th=[ 2999]
   bw (  KiB/s): min= 5288, max= 7672, per=100.00%, avg=6872.82, stdev=318.17, samples=610
   iops        : min= 1322, max= 1918, avg=1718.20, stdev=79.54, samples=610
  lat (usec)   : 250=94.33%, 500=3.47%, 750=1.22%, 1000=0.57%
  lat (msec)   : 2=0.36%, 4=0.04%, 10=0.01%
  fsync/fdatasync/sync_file_range:
    sync (usec): min=311, max=11455, avg=417.96, stdev=207.43
    sync percentiles (usec):
     |  1.00th=[  326],  5.00th=[  338], 10.00th=[  343], 20.00th=[  351],
     | 30.00th=[  355], 40.00th=[  363], 50.00th=[  371], 60.00th=[  379],
     | 70.00th=[  396], 80.00th=[  416], 90.00th=[  478], 95.00th=[  635],
     | 99.00th=[ 1336], 99.50th=[ 1663], 99.90th=[ 2737], 99.95th=[ 3359],
     | 99.99th=[ 5407]
  cpu          : usr=0.25%, sys=1.43%, ctx=1048906, majf=0, minf=13
  IO depths    : 1=200.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued rwts: total=0,524288,0,0 short=524287,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
  WRITE: bw=6873KiB/s (7038kB/s), 6873KiB/s-6873KiB/s (7038kB/s-7038kB/s), io=2048MiB (2147MB), run=305114-305114
    msec

Disk stats (read/write):
  vda: ios=0/1575558, merge=0/1057771, ticks=0/298083, in_queue=296276, util=96.51%
```

C Dataset generation instructions

The datasets can be generated using the LDBC SNB Datagen. To regenerate the data sets used in this benchmark, build the Datagen JAR as described in the project's README, configure the AWS EMR environment, upload the JAR to the S3 bucket (denoted as `${BUCKET_NAME}`) and run the following commands to generate the datasets used in this audit.
Note that while the datasets for TuGraph were generated as gzip-compressed archives, they are decompressed during load. Additionally, the dataset for SF10 was downloaded from the official LDBC CloudFlare R2 repository.

Listing C.1: Script to generate the SF10 dataset for TuGraph in AWS EMR. This dataset is only used for cross-validation

```
export SCALE_FACTOR=10
export JOB_NAME=sf${SCALE_FACTOR}-projected-csv-gz

./tools/emr/submit_datagen_job.py \
    --use-spot \
    --bucket ${BUCKET_NAME} \
    --copy-all \
    --az us-east-2c \
    ${JOB_NAME} \
    ${SCALE_FACTOR} \
    csv \
    bi \
    -- \
    --explode-edges \
    --format-options compression=gzip \
    --generate-factors
```

Listing C.2: Script to generate the SF10 dataset locally. This dataset is only used for cross-validation.

```
export SCALE_FACTOR=10
export LDBC_SNB_DATAGEN_MAX_MEM=60G
export LDBC_SNB_DATAGEN_JAR=$(sbt -batch -error 'print assembly / assemblyOutputPath')

tools/run.py \
    --cores $(nproc) \
    --memory ${LDBC_SNB_DATAGEN_MAX_MEM} \
    -- \
    --format csv \
    --scale-factor ${SCALE_FACTOR} \
    --explode-edges \
    --mode bi \
    --output-dir out-sf${SCALE_FACTOR}/ \
    --generate-factors \
    --format-options compression=gzip
```

Listing C.3: Script to generate the SF30000 dataset. This dataset is generated on AWS..

```
export LDBC_SNB_DATAGEN_JAR=$(sbt -batch -error 'print assembly / assemblyOutputPath')
export JAR_NAME=$(basename ${LDBC_SNB_DATAGEN_JAR})
export SCALE_FACTOR=30000
export JOB_NAME=sf${SCALE_FACTOR}-for-surf

./tools/emr/submit_datagen_job.py \
    --use-spot \
    --sf-per-executor 250 \
```

```
    --instance-type i3.8xlarge \
    --jar ${JAR_NAME} \
    --bucket ${BUCKET_NAME} \
    --copy-all \
    --az us-east-2a \
    ${JOB_NAME} \
    ${SCALE_FACTOR} \
    csv \
    bi \
    -- \
    --explode-edges \
    --format-options compression=gzip
```

Data used for SF30000

The data used for SF30000 was generated on the Aliyun cluster and stored in an Aliyun Object Storage Service. The number of entities generated were checked according to the number of entities from the LDBC SNB Specification [1].

[1] Specification can be retrieved from `https://ldbcouncil.org/ldbc_snb_docs/ldbc-snb-specification.pdf`, p155

D Data schema

Listing D.1: Content of the Gremlin schema used by TuGraph

```
create graph view ldbc_bi_graph (
  vertex City (
    id BIGINT,
    name VARCHAR,
    url VARCHAR,
    identify id(id, 'City')
  ),
  vertex Comment (
    id BIGINT,
    browserUsed VARCHAR,
    creationDate BIGINT,
    locationIP VARCHAR,
    content VARCHAR,
    length INTEGER,
    identify id(id, 'Comment')
  ),
  vertex Company (
    id BIGINT,
    name VARCHAR,
    url VARCHAR,
    identify id(id, 'Company')
  ),
  vertex Continent (
    id BIGINT,
    name VARCHAR,
    url VARCHAR,
    identify id(id, 'Continent')
  ),
  vertex Country (
    id BIGINT,
    name VARCHAR,
    url VARCHAR,
    identify id(id, 'Country')
  ),
  vertex Forum (
    id BIGINT,
    title VARCHAR,
    creationDate BIGINT,
    memberCount INTEGER,
    identify id(id, 'Forum')
  ),
  vertex Person (
    id BIGINT,
    firstName VARCHAR,
    lastName VARCHAR,
    gender VARCHAR,
    birthday VARCHAR,
    locationIP VARCHAR,
    browserUsed VARCHAR,
    language VARCHAR,
    email VARCHAR
    creationDate BIGINT,
    popularityScore INTEGER,
    identify id(id, 'Person')
```

```
  ),
  vertex Post (
    id BIGINT,
    browserUsed VARCHAR,
    creationDate BIGINT,
    locationIP VARCHAR,
    content VARCHAR,
    length INTEGER,
    language VARCHAR,
    imageFile VARCHAR,
    identify id(id, 'Post')
  ),
  vertex Tag (
    id BIGINT,
    name VARCHAR,
    url VARCHAR,
    identify id(id, 'Tag')
  ),
  vertex TagClass (
    id BIGINT,
    name VARCHAR,
    url VARCHAR,
    identify id(id, 'TagClass')
  )
  vertex University (
    id BIGINT,
    name VARCHAR,
    url VARCHAR,
    identify id(id, 'University')
  ),
  edge containerOf (
    srcId BIGINT,
    tarId BIGINT,
    edge prop(srcId, tarId, 0, 'containerOf')
  ),
  edge hasCreator (
    srcId BIGINT,
    tarId BIGINT,
    edge prop(srcId, tarId, 0, 'hasCreator')
  ),
  edge hasInterest (
    srcId BIGINT,
    tarId BIGINT,
    edge prop(srcId, tarId, 0, 'hasInterest')
  ),
  edge hasMember (
    srcId BIGINT,
    tarId BIGINT,
    creationDate BIGINT,
    edge prop(srcId, tarId, creationDate, 'hasMember')
  ),
  edge hasModerator (
    srcId BIGINT,
    tarId BIGINT,
    edge prop(srcId, tarId, 0, 'hasModerator')
  ),
  edge hasTag (
    srcId BIGINT,
```

```
    tarId BIGINT,
    edge prop(srcId, tarId, 0, 'hasTag')
  ),
  edge hasType (
    srcId BIGINT,
    tarId BIGINT,
    edge prop(srcId, tarId, 0, 'hasType')
  ),
  edge isLocatedIn (
    srcId BIGINT,
    tarId BIGINT,
    edge prop(srcId, tarId, 0, 'isLocatedIn')
  ),
  edge isPartOf (
    srcId BIGINT,
    tarId BIGINT,
    edge prop(srcId, tarId, 0, 'isPartOf')
  ),
  edge isSubClassOf (
    srcId BIGINT,
    tarId BIGINT,
    edge prop(srcId, tarId, 0, 'isSubClassOf')
  ),
  edge knows (
    srcId BIGINT,
    tarId BIGINT,
    creationDate BIGINT,
    edge prop(srcId, tarId, creationDate, 'knows')
  ),
  edge likes (
    srcId BIGINT,
    tarId BIGINT,
    creationDate BIGINT,
    edge prop(srcId, tarId, creationDate, 'likes')
  ),
  edge replyOf (
    srcId BIGINT,
    tarId BIGINT,
    edge prop(srcId, tarId, 0, 'replyOf')
  ),
  edge studyAt (
    srcId BIGINT,
    tarId BIGINT,
    classYear INTEGER,
    edge prop(srcId, tarId, 0, 'studyAt')
  ),
  edge workAt (
    srcId BIGINT,
    tarId BIGINT,
    workFrom INTEGER,
    edge prop(srcId, tarId, 0, 'workAt')
  ),
  edge rootPost (
    srcId BIGINT,
    tarId BIGINT,
    creationDate BIGINT,
    edge prop(srcId, tarId, creationDate, 'rootPost')
  ),
```

```
  edge weight19 (
    srcId BIGINT,
    tarId BIGINT,
    weight INTEGER,
    edge prop(srcId, tarId, weight, 'weight19')
  ),
  edge weight20 (
    srcId BIGINT,
    tarId BIGINT,
    weight INTEGER,
    edge prop(srcId, tarId, weight, 'weight20')
  )
)
```