LDBC

The graph & RDF benchmark reference

Full Disclosure Report of the LDBC Social Network Benchmark

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

April 6, 2023

General Terms

Executive Summary

This report documents an audited execution of the LDBC SNB BI (Social Network Benchmark Business Intelligence) workload for TigerGraph.

TigerGraph is a massively parallel processing (MPP) graph database management system, designed for handling hybrid transaction/analytical processing (HTAP) query workloads. It is a distributed platform using a native graph storage format with an edge cut partitioning strategy. Within this, each segment (partition) of the graph holds a similar amount of vertices and processes requests in parallel. TigerGraph offers GSQL, a Turing-complete query language which provides both declarative features (e.g. graph patterns) as well as imperative ones (e.g. for expressing iterative graph algorithms with loops and accumulator primitives). TigerGraph previously passed the LDBC SNB BI benchmark with a single-machine on-premise setup using the SF1 000 dataset.

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

TigerGraph, as an LDBC member, are the Test Sponsor of this audit. The audit and its associated execution costs (compute, storage) were funded by AMD. This arrangement was deemed acceptable by both parties, the LDBC Steering Committee and the Auditor.

DocuSigned by:
David Püroja
C7FE4F71E2554AE...

4/6/2023

Mr. David Püroja
(Auditor)

Date

Pometry
(Audit Company)

DocuSigned by:
Gabor Szarnyas
FA33C2C8CB944A5...

4/6/2023

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

Date

DocuSigned by:
Songting Chen
BD88B2D9F5AA4C8...

4/6/2023

Dr. Songting Chen
(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	0.5.0	https://github.com/ldbc/ldbc_snb_datagen_spark/releases/tag/v0.5.0
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 on a single virtual machine instance in the Amazon Web Service (AWS) cloud. The details below where obtained from the AWS console.

Table 2.1: Machine Type and Location

Cloud provider	Amazon Web Services
Machine region	N. Virginia (us-east-1)
Common name of the item	r6a.4xlarge
Operating system	Amazon Linux 2 (5.10.167-147.601.amzn2.x86_64)

2.1.2 CPU details

The details below were obtained using the command `cat /proc/cpuinfo` (Listing A.1) and `dmidecode -t processor` (Listing A.3) issued from the machine instance.

Table 2.2: CPU details summary

Type	AMD® AMD EPYC® 7R13 CPU
Total number	1
Cores per CPU	8
Threads per CPU core	2
CPU clock frequency	3.725 MHz
Total cache size per CPU	L1i cache: 32KiB L1d cache: 32KiB L2 cache: 512KiB L3 cache: 33MiB

2.1.3 Memory details

The total size of the memory installed is 128GiB and the type of memory is DDR4 with frequency 3200MHz. This information was obtained using the `sudo lshw -c memory` command (Listing A.5) issued from the virtual machine instance.

2.1.4 Disk and storage details

The virtual machine instance used Amazon EBS General Purpose 3 (GP3) storage, formatted using the `xfs` file system, configured with properties shown in Table 2.3.

Table 2.3: Amazon EBS GP3 Configuration

Type	GP3
Size	400GB
IOPS	16000
Throughput	1000

Disk controller or motherboard type was not obtainable from the virtual machine instance. Information on AWS General Storage can be found on the website `https://aws.amazon.com/ebs/general-purpose/` (accessed: March 16, 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 765 IOPS.

2.1.5 Network details

The presented benchmark run only used a single machine, thus network details are not included here.

2.1.6 Machine pricing

The system pricing summary in US dollars (USD) is included in the table below. The pricing of the AWS machine instance is the price for a 3-year reserved dedicated instance machine without upfront payment using the EC2 Instance savings plan [1].

Table 2.4: Pricing summary

Item	Price
r6a.4xlarge reserved instance machine in AWS (standard 3-year term)	10 815 USD
Software license (3 years)	120 000 USD
Maintenance fee (3 years)	12 000 USD
Total cost	142 815 USD

2.1.7 System version and availability

Table 2.5: System versions

System	Version	License
TigerGraph	3.7.0	Enterprise Licence provided by TigerGraph

[1] `https://aws.amazon.com/savingsplans/compute-pricing/`

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 TigerGraph	composite-projected-fk layout, compressed CSV files
Scale factors for TigerGraph	10 (validation), 100 (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 the cross-validation runs were retrieved from the following URLs and copied to an AWS S3 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]

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`

3.2.2 SF100

- `https://pub-383410a98aef4cb686f0c7601eddd25f.r2.dev/bi-pre-audit/parameters-2022-10-01.zip`
- `https://pub-383410a98aef4cb686f0c7601eddd25f.r2.dev/bi-pre-audit/bi-sf100-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 the command shown in Listing 3.1.

```
cd /data/ldbc_snb_bi/tigergraph
# change the following lines in k8s/vars.sh
# export NUM_NODES=1 # number of pods or nodes
# export SF=100
# export TG_DATA_DIR=/home/tigergraph/sf${SF}
# export TG_PARAMETER=/home/tigergraph/parameters-sf${SF}

nohup ./k8s/setup.sh > log.setup 2>&1 < /dev/null &
tail -f log.setup
```

Listing 3.1: Script to load the data on TigerGraph for SF100

The data preprocessing and loading times are reported below. Values were measured using the GNU Time tool (`/usr/bin/time`) with the `-v` flag, reading the *Elapsed (wall clock) time* from the output. The column **Data preprocessing time** shows how much time it took to preprocess the CSV files. For this benchmark execution, the preprocessing only consisted of decompressing the `.csv.gz` files. The column **Data loading time** shows

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

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 TigerGraph data schema is shown in Listing D.1.

Table 3.2: Data preprocessing and loading times for TigerGraph on scale factor 100

Scale factor	Data preprocessing time (s)	Data loading time (s)	Total time (s)
100	165	1 574	1 739

The decomposition of the **Data loading times** are shown in Table 3.3.

Table 3.3: Decomposition of data loading time

	Time (s)
Schema setup	45
Load data	1 228
Query install	118
Precompute	102
Rebuild	82
Total	1 574

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, TigerGraph was executed using the *disjoint read-write mode*. Therefore, no ACID tests were conducted.

4.2 Use of auxiliary data structures

The TigerGraph implementation precomputes the following auxiliary data structures. These are executed in each batch after the writes have been applied.

- **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.
- **Q14:** The weight attributed on the knows edges are precomputed based on the number of interactions between the two Person nodes.
- **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.

The precomputations for Q14 and Q19 are performed together using different scoring methods for establishing the edge weights. We display the runtime of this operation as "precomputation for Q14 and Q19" in Table 5.3.

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

4.3 Benchmark execution

The benchmark is executed using the following commands.[2]

```
cd /data/ldbc_snb_bi/tigergraph
# change the following lines in k8s/vars.sh
# export NUM_NODES=1 # number of pods or nodes
# export SF=100
# export TG_DATA_DIR=/home/tigergraph/sf${SF}
# export TG_PARAMETER=/home/tigergraph/parameters-sf${SF}

nohup ./k8s/benchmark.sh > log.benchmark 2>&1 < /dev/null &
tail -f log.benchmark
```

Listing 4.1: Script to execute the benchmark on TigerGraph for SF100

[2]Despite what the script's name suggests, this is benchmark was executed on an AWS EC2 instance without Kubernetes running, with the TigerGraph instance running on a single server machine and not using any containerization technology.

5 Performance Results

5.1 TigerGraph performance results

Table 5.1: Summary of results for TigerGraph on scale factor 100

Benchmark duration	Power@SF	Power@SF/$	Throughput@SF	Throughput@SF/$
113.07 minutes	6 253.72	43.79	3 723.44	26.07

Table 5.2: Detailed **power test results** for TigerGraph on scale factor 100. Execution times are reported in seconds.

Query	Count	Min.	Max.	Mean	P_{50}	P_{90}	P_{95}	P_{99}
1	30	3.901	4.319	4.126	4.128	4.240	4.307	4.319
2a	30	0.532	3.160	1.334	0.947	2.821	2.958	3.160
2b	30	0.434	0.691	0.563	0.557	0.653	0.689	0.691
3	30	1.564	2.803	1.923	1.731	2.570	2.791	2.803
4	30	1.161	1.384	1.185	1.175	1.196	1.199	1.384
5	30	0.843	0.859	0.851	0.852	0.856	0.856	0.859
6	30	0.649	0.858	0.670	0.662	0.677	0.678	0.858
7	30	0.968	1.014	0.994	0.994	1.010	1.012	1.014
8a	30	0.895	0.971	0.928	0.926	0.942	0.949	0.971
8b	30	0.815	0.826	0.818	0.817	0.823	0.824	0.826
9	30	3.083	3.587	3.348	3.337	3.533	3.554	3.587
10a	30	2.853	3.242	2.958	2.941	3.018	3.220	3.242
10b	30	2.219	2.512	2.396	2.435	2.502	2.508	2.512
11	30	1.365	1.499	1.427	1.427	1.473	1.488	1.499
12	30	2.613	3.997	3.292	3.548	3.966	3.982	3.997
13	30	3.510	3.849	3.588	3.571	3.674	3.771	3.849
14a	30	3.328	4.612	3.658	3.596	3.712	4.484	4.612
14b	30	2.644	2.664	2.652	2.650	2.663	2.664	2.664
15a	30	3.024	3.631	3.462	3.488	3.579	3.619	3.631
15b	30	6.594	12.671	9.787	9.820	11.906	12.140	12.671
16a	30	1.683	2.340	2.008	2.038	2.205	2.278	2.340
16b	30	1.418	1.575	1.460	1.430	1.544	1.564	1.575
17	30	2.098	2.203	2.151	2.154	2.170	2.172	2.203
18	30	2.928	3.352	3.073	3.058	3.216	3.292	3.352
19a	30	1.060	1.248	1.172	1.187	1.232	1.245	1.248
19b	30	1.085	1.297	1.207	1.204	1.252	1.263	1.297
20a	30	0.409	2.042	1.187	1.420	2.039	2.041	2.042
20b	30	0.610	1.834	1.139	1.016	1.626	1.829	1.834

Table 5.3: Operations in the **power test** for TigerGraph on scale factor 100. 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	1 901.047
writes	327.051
q4precomputation	16.061
q6precomputation	19.567
q19precomputation	143.582
q20precomputation	6.773

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–TigerGraph cross-validation command

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

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

[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{10,100}.tar	Query substitution parameters
scoring/calculate-scores.py	Python script to calculate the scores of the benchmark run
tigergraph/output/output-sf{10,100}	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
tigergraph-3.7.0-offline.tar.gz	Installer of Tigergraph

The `ldbc_snb_bi_tigergraph_sf100_attachments` folder's directory structure is as follows:

```
ldbc_snb_bi_tigergraph_sf100_attachments
├── umbra
│   └── output
│       └── output-sf10
│           └── results.csv
├── parameters
│   ├── parameters-sf10.tar
│   └── parameters-sf100.tar
├── scoring
│   └── calculate-scores.py
├── tigergraph
│   └── output
│       ├── output-sf10
│       │   ├── benchmark.csv
│       │   ├── load.csv
│       │   ├── results.csv
│       │   └── timings.csv
│       └── output-sf100
│           ├── benchmark.csv
│           ├── load.csv
│           ├── results.csv
│           └── timings.csv
├── ldbc_snb_bi-1.0.3.tar.gz
├── ldbc_snb_datagen_spark-0.5.1.tar.gz
├── ldbc_snb_specification-2.2.0.pdf
└── tigergraph-3.7.0-offline.tar.gz
```

A CPU and Memory details

Listing A.1: Output of the `cat /proc/cpuinfo` command for a single CPU core

```
processor   : 0
vendor_id   : AuthenticAMD
cpu family  : 25
model               : 1
model name  : AMD EPYC 7R13 Processor
stepping    : 1
microcode   : 0xa001173
cpu MHz             : 1727.788
cache size  : 512 KB
physical id : 0
siblings    : 16
core id             : 0
cpu cores   : 8
apicid              : 0
initial apicid      : 0
fpu         : yes
fpu_exception       : yes
cpuid level : 16
wp          : yes
flags               : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid aperfmperf tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm cmp_legacy cr8_legacy abm sse4a misalignsse 3dnowprefetch topoext invpcid_single ssbd ibrs ibpb stibp vmmcall fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 clzero xsaveerptr rdpru wbnoinvd arat npt nrip_save vaes vpclmulqdq rdpid
bugs                : sysret_ss_attrs null_seg spectre_v1 spectre_v2 spec_store_bypass
bogomips    : 5299.98
TLB size    : 2560 4K pages
clflush size        : 64
cache_alignment     : 64
address sizes       : 48 bits physical, 48 bits virtual
power management:
```

Listing A.2: Output of the `lscpu` command

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              16
On-line CPU(s) list: 0-15
Thread(s) per core:  2
Core(s) per socket:  8
Socket(s):           1
NUMA node(s):        1
Vendor ID:           AuthenticAMD
CPU family:          25
Model:               1
Model name:          AMD EPYC 7R13 Processor
Stepping:            1
CPU MHz:             1808.883
BogoMIPS:            5299.98
Hypervisor vendor:   KVM
Virtualization type: full
```

```
  L1d cache:           32K
  L1i cache:           32K
  L2 cache:            512K
  L3 cache:            32768K
  NUMA node0 CPU(s):   0-15
  Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr
   sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid
  extd_apicid aperfmperf tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes
   xsave avx f16c rdrand hypervisor lahf_lm cmp_legacy cr8_legacy abm sse4a misalignsse 3dnowprefetch topoext
  invpcid_single ssbd ibrs ibpb stibp vmmcall fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap
  clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 clzero xsaveerptr rdpru wbnoinvd arat npt nrip_save vaes
  vpclmulqdq rdpid
```

Listing A.3: Output of the `dmidecode -t processor` command

```
# dmidecode 3.2
Getting SMBIOS data from sysfs.
SMBIOS 2.7 present.

Handle 0x0004, DMI type 4, 42 bytes
Processor Information
  Socket Designation: CPU 0
  Type: Central Processor
  Family: Zen
  Manufacturer: Advanced Micro Devices, Inc.
  ID: 11 0F A0 00 FF FB 8B 17
  Signature: Family 25, Model 1, Stepping 1
  Flags:
    FPU (Floating-point unit on-chip)
    VME (Virtual mode extension)
    DE (Debugging extension)
    PSE (Page size extension)
    TSC (Time stamp counter)
    MSR (Model specific registers)
    PAE (Physical address extension)
    MCE (Machine check exception)
    CX8 (CMPXCHG8 instruction supported)
    APIC (On-chip APIC hardware supported)
    SEP (Fast system call)
    MTRR (Memory type range registers)
    PGE (Page global enable)
    MCA (Machine check architecture)
    CMOV (Conditional move instruction supported)
    PAT (Page attribute table)
    PSE-36 (36-bit page size extension)
    CLFSH (CLFLUSH instruction supported)
    MMX (MMX technology supported)
    FXSR (FXSAVE and FXSTOR instructions supported)
    SSE (Streaming SIMD extensions)
    SSE2 (Streaming SIMD extensions 2)
    HTT (Multi-threading)
  Version: AMD EPYC 7R13 Processor
  Voltage: 1.1 V
  External Clock: 100 MHz
  Max Speed: 3725 MHz
  Current Speed: 2650 MHz
  Status: Populated, Enabled
  Upgrade: Socket SP3
```

```
L1 Cache Handle: 0x0005
L2 Cache Handle: 0x0006
L3 Cache Handle: 0x0007
Serial Number: Not Specified
Asset Tag: Not Specified
Part Number: Not Specified
Core Count: 8
Core Enabled: 8
Thread Count: 16
Characteristics:
  64-bit capable
  Multi-Core
  Hardware Thread
  Execute Protection
```

Listing A.4: Output of the `cat /proc/meminfo` command

```
MemTotal:       129142308 kB
MemFree:        127394340 kB
MemAvailable:   127811256 kB
Buffers:             2704 kB
Cached:           1407196 kB
SwapCached:             0 kB
Active:            269236 kB
Inactive:         1176008 kB
Active(anon):         264 kB
Inactive(anon):     35540 kB
Active(file):      268972 kB
Inactive(file):   1140468 kB
Unevictable:            0 kB
Mlocked:                0 kB
SwapTotal:              0 kB
SwapFree:               0 kB
Dirty:                 32 kB
Writeback:              0 kB
AnonPages:          35548 kB
Mapped:             50820 kB
Shmem:                460 kB
KReclaimable:       76524 kB
Slab:              159884 kB
SReclaimable:       76524 kB
SUnreclaim:         83360 kB
KernelStack:         4048 kB
PageTables:          4296 kB
NFS_Unstable:           0 kB
Bounce:                 0 kB
WritebackTmp:           0 kB
CommitLimit:     64571152 kB
Committed_AS:      283340 kB
VmallocTotal:   34359738367 kB
VmallocUsed:        10660 kB
VmallocChunk:           0 kB
Percpu:              4608 kB
HardwareCorrupted:      0 kB
AnonHugePages:          0 kB
ShmemHugePages:         0 kB
ShmemPmdMapped:         0 kB
FileHugePages:          0 kB
```

```
FilePmdMapped:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
DirectMap4k:      102308 kB
DirectMap2M:     4423680 kB
DirectMap1G:   127926272 kB
```

Listing A.5: Output of the `lshw -C memory` command

```
  *-memory
       description: System Memory
       physical id: 8
       slot: System board or motherboard
       size: 128GiB
  *-bank
       description: DIMM DDR4 Static column Pseudo-static Synchronous Window DRAM 3200 MHz (0.3 ns)
       physical id: 0
       size: 128GiB
       width: 64 bits
       clock: 3200MHz (0.3ns)
```

B IO performance

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

```
iotest: (g=0): rw=write, bs=4K-4K/4K-4K/4K-4K, ioengine=sync, iodepth=1
fio-2.14
Starting 1 process
iotest: Laying out IO file(s) (1 file(s) / 2048MB)
Jobs: 1 (f=1): [W(1)] [100.0% done] [0KB/2966KB/0KB /s] [0/741/0 iops] [eta 00m:00s]
iotest: (groupid=0, jobs=1): err= 0: pid=6377: Mon Mar  6 09:36:21 2023
  write: io=2048.0MB, bw=3063.3KB/s, iops=765, runt=684628msec
    clat (usec): min=312, max=29041, avg=656.48, stdev=310.05
     lat (usec): min=312, max=29042, avg=656.63, stdev=310.05
    clat percentiles (usec):
     |  1.00th=[  382],  5.00th=[  414], 10.00th=[  438], 20.00th=[  478],
     | 30.00th=[  516], 40.00th=[  548], 50.00th=[  596], 60.00th=[  636],
     | 70.00th=[  684], 80.00th=[  756], 90.00th=[  908], 95.00th=[ 1112],
     | 99.00th=[ 1736], 99.50th=[ 2096], 99.90th=[ 3984], 99.95th=[ 4512],
     | 99.99th=[ 7328]
    lat (usec) : 500=26.46%, 750=52.64%, 1000=13.86%
    lat (msec) : 2=6.47%, 4=0.48%, 10=0.09%, 20=0.01%, 50=0.01%
  cpu          : usr=0.39%, sys=2.01%, ctx=1575480, majf=0, minf=12
  IO depths    : 1=100.0%, 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    : total=r=0/w=524288/d=0, short=r=0/w=0/d=0, drop=r=0/w=0/d=0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
  WRITE: io=2048.0MB, aggrb=3063KB/s, minb=3063KB/s, maxb=3063KB/s, mint=684628msec, maxt=684628msec

Disk stats (read/write):
  nvme0n1: ios=0/1069347, merge=0/637, ticks=0/678806, in_queue=678806, util=100.00%
```

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 TigerGraph were generated as gzip-compressed archives, they are decompressed during preprocessing. Decompressing the SF100 data set took 2 minutes 45 seconds (wall clock) when performed by the following command: `time find /home/tigergraph/sf100 -name "*.csv.gz" -print0 | parallel -q0 gunzip`

Listing C.1: Script to generate the SF10 dataset for TigerGraph 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 SF100 dataset for TigerGraph in AWS EMR. This dataset is used for the benchmark run

```
export SCALE_FACTOR=100
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.3: Script to generate the SF10 dataset for Umbra 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
```

D Data schema

Listing D.1: Content of the GSQL schema used by TigerGraph

```
## Message
CREATE VERTEX Comment (PRIMARY_ID id UINT, creationDate INT, locationIP STRING, browserUsed STRING, content STRING, length UINT) WITH primary_id_as_attribute="TRUE"
CREATE VERTEX Post (PRIMARY_ID id UINT, imageFile STRING, creationDate INT, locationIP STRING, browserUsed STRING, language STRING, content STRING, length UINT) WITH primary_id_as_attribute="TRUE"
## organisation
CREATE VERTEX Company (PRIMARY_ID id UINT, name STRING, url STRING) WITH primary_id_as_attribute="TRUE"
CREATE VERTEX University (PRIMARY_ID id UINT, name STRING, url STRING) WITH primary_id_as_attribute="TRUE"
## place
CREATE VERTEX City (PRIMARY_ID id UINT, name STRING, url STRING) WITH primary_id_as_attribute="TRUE"
CREATE VERTEX Country (PRIMARY_ID id UINT, name STRING, url STRING) WITH primary_id_as_attribute="TRUE"
CREATE VERTEX Continent (PRIMARY_ID id UINT, name STRING, url STRING) WITH primary_id_as_attribute="TRUE"
## etc
CREATE VERTEX Forum (PRIMARY_ID id UINT, title STRING, creationDate INT,
    maxMember UINT) WITH primary_id_as_attribute="TRUE" // maxMember is for precompute in BI-4
CREATE VERTEX Person (PRIMARY_ID id UINT, firstName STRING, lastName STRING, gender STRING, birthday INT, creationDate INT, locationIP STRING, browserUsed STRING, speaks SET<STRING>, email SET<STRING>,
    popularityScore UINT) WITH primary_id_as_attribute="TRUE" // popularityScore is for precompute in BI-6
CREATE VERTEX Tag (PRIMARY_ID id UINT, name STRING, url STRING) WITH primary_id_as_attribute="TRUE"
CREATE VERTEX TagClass (PRIMARY_ID id UINT, name STRING, url STRING) WITH primary_id_as_attribute="TRUE"


# create edge
CREATE DIRECTED EDGE CONTAINER_OF (FROM Forum, TO Post) WITH REVERSE_EDGE="CONTAINER_OF_REVERSE"
CREATE DIRECTED EDGE HAS_CREATOR (FROM Comment|Post, TO Person) WITH REVERSE_EDGE="HAS_CREATOR_REVERSE"
CREATE DIRECTED EDGE HAS_INTEREST (FROM Person, TO Tag) WITH REVERSE_EDGE="HAS_INTEREST_REVERSE"
CREATE DIRECTED EDGE HAS_MEMBER (FROM Forum, TO Person, creationDate INT) WITH REVERSE_EDGE="HAS_MEMBER_REVERSE"
CREATE DIRECTED EDGE HAS_MODERATOR (FROM Forum, TO Person) WITH REVERSE_EDGE="HAS_MODERATOR_REVERSE"
CREATE DIRECTED EDGE HAS_TAG (FROM Comment|Post|Forum, TO Tag) WITH REVERSE_EDGE="HAS_TAG_REVERSE"
CREATE DIRECTED EDGE HAS_TYPE (FROM Tag, TO TagClass) WITH REVERSE_EDGE="HAS_TYPE_REVERSE"
CREATE DIRECTED EDGE IS_LOCATED_IN (FROM Company, TO Country | FROM Person, TO City | FROM University, TO City) WITH REVERSE_EDGE="IS_LOCATED_IN_REVERSE"
CREATE DIRECTED EDGE MESG_LOCATED_IN (FROM Comment, TO Country | FROM Post, TO Country) // Reverse edge of Comment/Post -IS_Located_IN-> Country will cause Country connected by too many edges, which makes loading slow
CREATE DIRECTED EDGE IS_PART_OF (FROM City, TO Country | FROM Country, TO Continent) WITH REVERSE_EDGE="IS_PART_OF_REVERSE"
CREATE DIRECTED EDGE IS_SUBCLASS_OF (FROM TagClass, TO TagClass) WITH REVERSE_EDGE="IS_SUBCLASS_OF_REVERSE"
CREATE UNDIRECTED EDGE KNOWS (FROM Person, TO Person, creationDate INT, weight19 UINT, weight20 UINT DEFAULT 10000)
CREATE DIRECTED EDGE LIKES (FROM Person, TO Comment|Post, creationDate INT) WITH REVERSE_EDGE="LIKES_REVERSE"
CREATE DIRECTED EDGE REPLY_OF (FROM Comment, TO Comment|Post) WITH REVERSE_EDGE="REPLY_OF_REVERSE"
CREATE DIRECTED EDGE STUDY_AT (FROM Person, TO University, classYear INT) WITH REVERSE_EDGE="STUDY_AT_REVERSE"
CREATE DIRECTED EDGE WORK_AT (FROM Person, TO Company, workFrom INT) WITH REVERSE_EDGE="WORK_AT_REVERSE"

CREATE DIRECTED EDGE ROOT_POST (FROM Comment, TO Post) WITH REVERSE_EDGE="ROOT_POST_REVERSE" //FOR BI-3,9,17
CREATE DIRECTED EDGE REPLY_COUNT (FROM Person, TO Person, cnt UINT)

CREATE GLOBAL SCHEMA_CHANGE JOB addIndex {
  ALTER VERTEX Country ADD INDEX country_name ON (name);
  ALTER VERTEX Company ADD INDEX company_name ON (name);
  ALTER VERTEX University ADD INDEX university_name ON (name);
  ALTER VERTEX Tag ADD INDEX tag_name ON (name);
  ALTER VERTEX TagClass ADD INDEX tagclass_name ON (name);
```

```
}

RUN GLOBAL SCHEMA_CHANGE JOB addIndex
CREATE GRAPH ldbc_snb (*)
```