Full Disclosure Report of the LDBC Social Network SF30000 Benchmark

LDBC Social Network Benchmark's Business Intelligence Workload over TigerGraph

November 29, 2022

Disclaimer: the benchmark results on this report are not official LDBC benchmark results, as they have not been audited.

Executive Summary

Last year (2021), we reported an LDBC SNB benchmark at 36 TB size, but at that time the first version of the BI workload was not finalized. The old benchmark did not follow the LDBC specifications in several aspects such as the benchmark driver, the generation of substitution parameters and the benchmark workflow. This new benchmark uses a new version of TigerGraph with improved performance in loading and querying (e.g., the loading time is shortened from 35.5 hr to 6.5 hr). The implementation of read and write queries are also improved significantly, and the results are validated by another database on SF-10.

This report documents a complete execution of the LDBC SNB (Social Network Benchmark) BI (Business Intelligence) workload for TigerGraph at SF-30k. This benchmark, pending audit by LDBC, uses the official benchmark driver, query implementation, data and substitution parameter generator reported in LDBC 1T Audit FDR over TigerGraph, but on scale-factor 30k this time. The queries were executed using 5 substitution parameters in each batch, whereas the official benchmark uses 30 different parameters. The *power* and *throughput* benchmark metrics are reported following the guidelines of the LDBC SNB specification.

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 graph partition holds a similar amount of vertices and edges and processes requests in parallel. TigerGraph offers GSQL, a Turing-complete query language that 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).

The focus of this benchmark test is TigerGraph's performance on Business Intelligence (BI) workloads over a sequence of batch-refreshed big graphs. The BI workload includes:

- **20 Read Queries**—the majority of OLAP-style iterative and deep-link graph queries were answered in sub-minute to a couple of minutes. The queries include explosive and redundant multi-joins and multi-source shortest path problems on a weighted graph.

- **Incremental Batch Updates**—the graph is mutated by a set of insert and delete operations. The data to be inserted or deleted are batched for a period of one day.

The TigerGraph server is deployed over 36 Amazon Web Service (AWS) r6a.32xlarge instances with 144TB disk volume. These instances are powered by the 3rd generation AMD EPYC processors. The following high-level summary highlights the novelties in numbers:

- Overall, the full source dataset is about 36TB, containing 539.6 billion relationships and 72.6 billion vertices.
- Total benchmark time is 19.3 hours, including the initial data loading, 1 power batch run, and 1 throughput batch run.
- Hardware cost is $281.27/hr, including 36 AWS machines and 144T GP2 SSD volumes.

1 Benchmark Overview

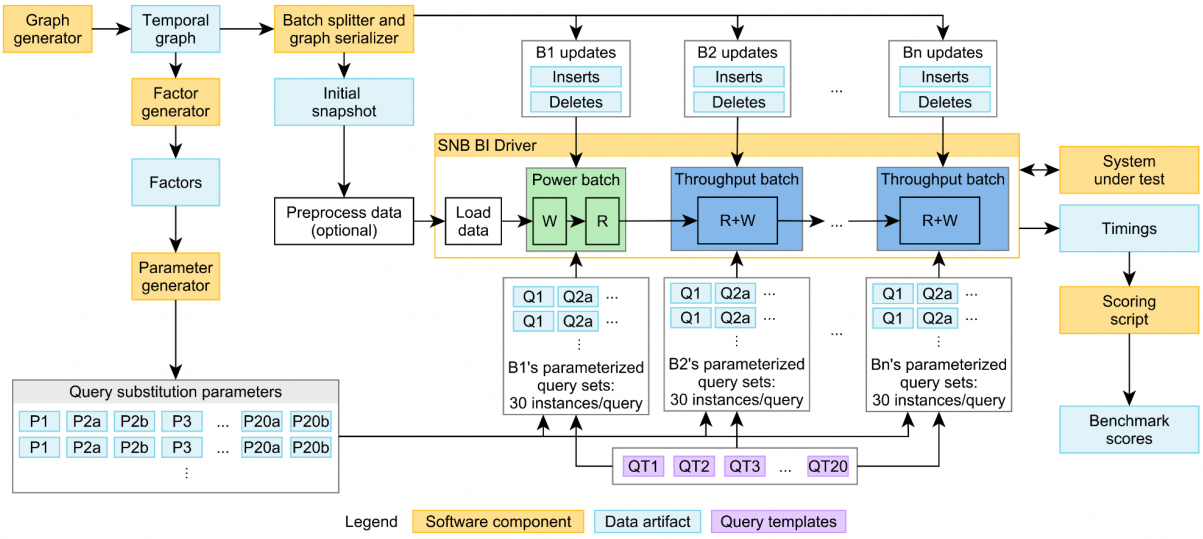

Fig. 1.1 Overview of the LDBC SNB BI workload. The SNB BI driver prompts TigerGraph to load data, perform one power batch and several throughput batches sequentially. Each power/throughput batch consists of write (W) and read (R) operations, where write operations consume inserts/deletes data and read operations run query templates (QT) on substitution parameters (P1, P2a, P2b, etc.) (source: [The LDBC Social Network Benchmark version 2.2.1])

The test was conducted in compliance with the Social Network Benchmark's specification, except that the queries were executed on 5 substitution parameters instead of 30.

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	1.0.2	https://github.com/ldbc/ldbc_snb_bi/releases/tag/v1.0.2

2 System Description and Pricing Summary

2.1 Machine overview

The hardware used for benchmarking LDBC-SNB on Scale Factor 30k are 36 AWS EC2 instances of type r6a.32xlarge. All of these instances use AMD 3rd-generation processors Milan.

Table 2.1: Machine Type

Number of Virtual Machines	36
Instance Type	r6a.32xlarge
Operating System	Amazon Linux 2 AMI (HVM) - Kernel 4.14
vCPU	128/node
Memory	1024G/node

2.2 CPU details

The details below were obtained using the commands `cat /proc/cpuinfo` (Listing A.1) and `lscpu` (Listing A.2).

Table 2.2: CPU details summary

Type	AMD® EPYC® 7R13 Processor
Total number	2
Cores per CPU	32
Threads per CPU	64
Total threads	128
CPU clock frequency	3.164 GHz
Total cache size per CPU	L1d cache: 32K L1i cache: 32K L2 cache: 512K L3 cache: 32768K

2.3 Memory details

The total aggregate size of the memory installed is 35.5TB. This information was obtained using the `cat /proc/meminfo` (Listing A.3) and `lshw -C` memory (Listing A.4) commands.

2.4 Disk and storage details

Table 2.3: Disk details summary

Disk	AWS General Purpose SSD (gp2)
Device Size	4TB
Max IOPS	12000
Max throughput	250 MB/s

2.5 Network details

Table 2.4: Network details summary

Instance	r6a.32xlarge
Network Bandwidth	50 Gbps
EBS bandwidth	26.6 Gbps

2.6 Machine pricing

Table 2.5: Pricing Summary

Item	Price
Total AWS EC2 instance cost for 3 years	$6,866,176 ($261.27/hr)
Total AWS EC2 volume cost for 3 years	$518,400 ($14,400/month)

2.7 System version and availability

Table 2.6: System version

System	Version	License
TigerGraph	3.7.0	Enterprise License provided by TigerGraph

3 Dataset Generation

3.1 General information

Dataset larger than SF10k are currently not available in the official Cloudflare R2. The dataset and query substitution parameters are generated using the LDBC's official data generator v0.5.0 and stored in the GCP bucket hosted by TigerGraph (gs://ldbc_bi/sf3000and gs://ldbc_bi/parameters_sf3000). The data generation settings of the LDBC Datagen are described below.

Table 3.1: Datagen settings summary

Data format	composite-projected-fk layout, compressed CSV files
Scale factors	30000

3.2 Data loading and data schema

The data preprocessing and loading times are reported below. 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 how long it took to create a graph from the input CSV files and perform the initial indexing of vertices and edges, including schema setup, initial data loading, query installation, pre-computation, etc. The initial data loading alone took 19703 s. The column **Total time** contains the sum of the data preprocessing and loading times. The TigerGraph topology data size and compression ration are shown in Fig. 3.1. The TigerGraph data schema is shown in Listing D.1 in Section 9.

The following configurations were updated on top of the default configuration:

```
gadmin config group timeout
```

- `Add "MVExtraCopy=0;"  //default is 1; this turns off backup copy.`

```
gadmin config group timeout
```

- `FileLoader.Factory.DefaultQueryTimeoutSec: 16 -> 6000`
- `KafkaLoader.Factory.DefaultQueryTimeoutSec: 16 -> 6000`
- `RESTPP.Factory.DefaultQueryTimeoutSec: 16 -> 6000`

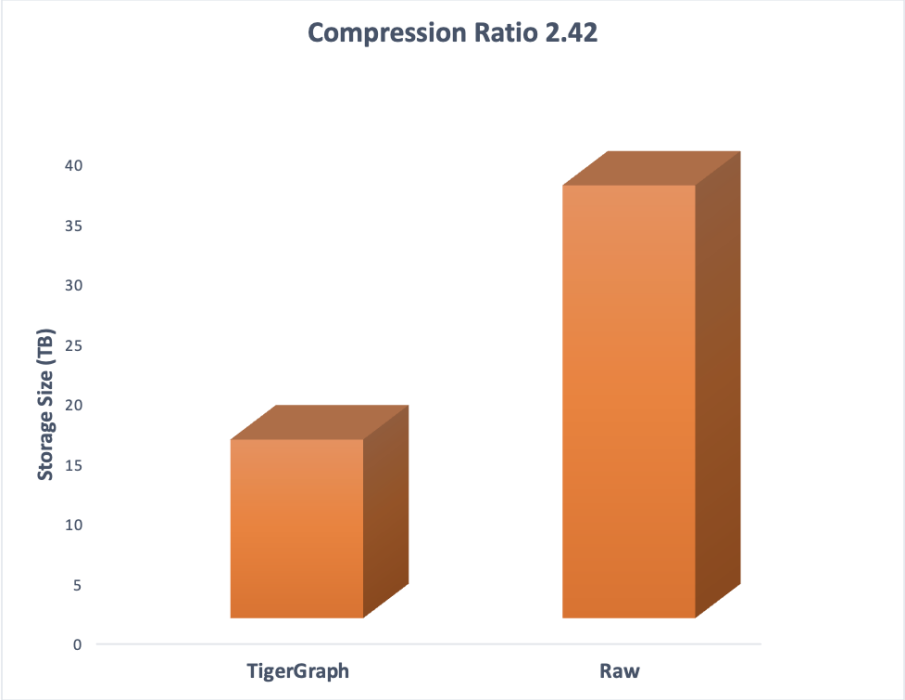

Fig. 3.1 The disk space consumed by the dataset was 2.42 times smaller once loaded into TigerGraph

Table 3.2: Data preprocessing and loading times for TigerGraph on scale factor 30000

Scale factor	Data preprocessing time	Data loading time	Total time
30000	1hr27min (5 214 s)	6h38min (2 3915 s)	8hr5min (29 129 s)

3.3 Data statistics

The statistics of the initial state for each vertex and edge type is shown in the following table.

Table. 3.3 Cardinality for each vertex type (total **72.62B** vertices)

Dynamic		Static	
Vertex Type Name	Cardinality (# of vertices)	Vertex Type Name	Cardinality (# of vertices)
Comment	58,666,958,815	Company	1,575
Post	13,148,296,221	University	6,380
Forum	728,629,666	City	1,343

Person	74,689,437	Country	111
		Continent	6
		Tag	16,080
		TagClass	71

Table. 3.4 Cardinality for each edge type (total **539.57B** edges)

Edge Type Name	Cardinality (# of edges)
CONTAINER_OF	13,148,296,221
HAS_CREATOR	71,815,255,036
HAS_INTEREST	1,747,667,501
HAS_MEMBER	90,652,090,014
HAS_MODERATOR	728,629,666
HAS_TAG	101,534,577,622
IS_LOCATED_IN	74,697,392
MESG_LOCATED_IN	71,815,255,036
KNOWS	5,734,470,022
LIKES	123,425,491,642
REPLY_OF	58,666,958,815
STUDY_AT	59,758,459
WORK_AT	162,518,922
HAS_TYPE	16,080
IS_PART_OF	1,454
IS_SUBCLASS_OF	70

4 Benchmark workflow and implementation

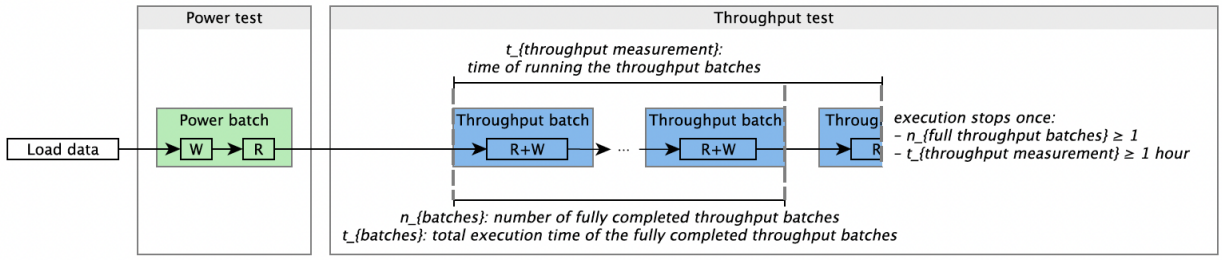

Fig. 4.1 Tests and batches (power and throughput) executed in the BI workload's workflow (source: The LDBC Social Network Benchmark version 2.2.1)

In Fig.4, the benchmark consists of load data, power test and throughput test. The power test, which runs a single power batch, first executes the write operations and then a sequential execution of individual read query variants. The official benchmark requires 28 query variants x 30 substitution parameters = 840 read queries to be executed, but in this benchmark, we executed 5 substitution parameters, and thus a total of 28 x 5 = 140 queries.

The throughput test consists of multiple throughput batches. The type and number of operations in each throughput batch are the same as the power batch. The only difference is that the throughput test allows the operations executed concurrently. In the current implementation, the throughput batch is implemented in the same way as power batch where the write and read operations are performed sequentially.

The implementation is the same as in the previous LDBC audited SF-1000 benchmark except that the queries were executed on 5 substitution parameters instead of 30.

5 Validation of the Results

As mentioned in the SF-1000 benchmark, the implementation has been cross validated against Neo4j on SF-10.

6 Performance Results

Based on the Sec 7.5.4 Scoring Metrics in Specification, the *power score*, *throughput score* and their price adjusted variants *per-$ power score* and *per-$ throughput score* are calculated. The benchmark time shown in Table 6.1 indicates the total elapsed time for both the power and throughput batch. The power score is calculated as

$$power@SF = \frac{3\,600}{\sqrt[29]{w \cdot q_1 \cdot q_{2a} \cdot q_{2b} \cdot \ldots \cdot q_{18} \cdot q_{19a} \cdot q_{19b} \cdot q_{20a} \cdot q_{20b}}} \cdot SF$$

where $w = 8685.72$ is the time in second to perform the writes and $q_1, q_{2a}, q_{2b} \ldots q_{20b}$ are the time in second for executing each variant with 30 different substitution parameters. In this benchmark, $q_1, q_{2a}, q_{2b} \ldots q_{20b}$ are extrapolated by applying a factor of 6 to the time spent on 5 substitution parameters (i.e., the sum column in Table 6.2).
The throughput score is calculated as

$$throughput@SF = (24 \text{ hours} - t_{load}) \cdot \frac{n_{batches}}{t_{batches}} \cdot SF$$

where t_{load} is the load time and is 6hr 38min, $n_{batches} = 1$ is the number of throughput batch in this benchmark, $t_{batches}$ is the time spent in a throughput batch with 30 substitution parameters and is 26 hr.

The price-adjusted score are

$$power@SF/\$ = power@SF \times \frac{1000}{Cost},$$

$$throughput@SF/\$ = throughput@SF \times \frac{1000}{Cost},$$

where the *Cost* only includes the hardware cost, whereas the specification uses the *total cost of ownership (TCO)* that includes both software and hardware cost.

Table 6.1: Summary of results for TigerGraph on scale factor 30000.
The benchmark time includes both the power batch (inserts/deletes+precompute+read) and the throughput batch (inserts/deletes+precompute+read) elapsed time.

Benchmark time	Power@SF	Power@SF/$	Throughput@SF	Throughput@SF/$
12.67 hr	60 068.58	8.13	20 027.19	2.71

Table 6.2: Detailed power test results for TigerGraph on scale factor 30000. Execution times are reported in seconds

BI Query	Sum	Max	Min	Avg
1	91.419	20.145	17.586	18.284
2a	577.132	168.779	59.311	115.426
2b	250.091	57.737	39.141	50.018
3	547.155	162.629	79.105	109.431
4	64.086	15.630	9.736	12.817
5	112.809	24.741	20.608	22.562
6	118.150	25.301	21.346	23.630
7	495.738	102.835	96.859	99.148
8a	176.702	38.936	30.831	35.340
8b	84.273	18.156	16.162	16.855
9	251.649	54.430	46.494	50.330
10a	619.945	145.280	97.929	123.989
10b	123.262	30.970	7.894	24.652
11	220.345	46.906	41.140	44.069
12	511.638	125.021	86.790	102.328
13	1,445.889	294.634	285.058	289.178
14a	932.480	195.364	170.451	186.496
14b	308.624	80.657	48.218	61.725
15a	844.558	174.921	164.657	168.912
15b	2,386.352	543.165	376.914	477.270
16a	732.495	174.636	101.264	146.499
16b	162.310	36.736	28.881	32.462
17	433.761	95.317	79.962	86.752
18	1,881.355	381.545	368.749	376.271
19a	275.493	55.997	54.100	55.099
19b	284.020	59.429	54.873	56.804
20a	29.544	15.383	2.213	5.909
20b	24.718	6.641	2.998	4.944

Table 6.3: Operations in the **power test** for TigerGraph on scale factor 30000. Execution times are reported in seconds. ROOT_POST[1] pre-computations are performed for each Comment insertion and deletion operation. Therefore, they are reported as part of the writes.

Operation	Time (hh:mm:ss)	Time (seconds)
Total read time (5 runs)	3:53:06	13,986.090
Total write time	2:24:46	8,685.720
Precomputation for Q4	0:05:40	340.190
Precomputation for Q6	0:22:43	1,363.020
Precomputation for Q14 and Q19	1:23:16	4,995.530
Precomputation for Q20	0:05:35	335.410

7 Conclusion

The social network benchmark in this report demonstrates TigerGraph's capability in graph analytics and business intelligence on graph-structured data at tens of terabytes scale. The new LDBC SNB BI workloads include the two challenges:

- Micro-batch of insert and delete operations to mutate the current graph
- Complex read queries that touch a significant portion of the data. The queries are designed based on the choke points, such as the challenging aspects of query processing, such as explosive and redundant multi-joins, expressive pathfinding, etc.

TigerGraph is capable of handling the deep-link OLAP style queries on this mutable big graph of 72.6 billion vertices and 539.6 billion edges, half of the data intensive read queries returning results within 1 minute, and the other half spent from 1 minute to 8 minutes.

This benchmark experiment exhibits TigerGraph's ability to deal with big graph workloads in a real production environment, where 10s terabytes of connected data with daily or hourly incremental updates is a norm. No other graph database or relational database vendor has demonstrated equivalent analytical and operational capabilities on this large-scale updatable graph to the best of our knowledge.

[1]In the precomputation phase, an auxiliary edge is added between each comment and its root Post. This edge is called ROOT_POST

8 Acknowledgement

The cloud computing cost used in this benchmark is funded by AMD and is gratefully acknowledged.

9 Supplemental Materials

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         : 1406.820
cache size      : 512 KB
physical id     : 0
siblings        : 64
core id         : 0
cpu cores       : 32
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
xtopology 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
arat npt nrip_save vaes vpclmulqdq rdpid
bugs            : sysret_ss_attrs spectre_v1 spectre_v2 spec_store_bypass
bogomips        : 5299.96
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):              128
On-line CPU(s) list: 0-127
Thread(s) per core:  2
Core(s) per socket:  32
Socket(s):           2
NUMA node(s):        4
Vendor ID:           AuthenticAMD
CPU family:          25
Model:               1
Model name:          AMD EPYC 7R13 Processor
Stepping:            1
CPU MHz:             3594.731
BogoMIPS:            5299.96
Hypervisor vendor:   KVM
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            512K
L3 cache:            32768K
NUMA node0 CPU(s):   0-15,64-79
NUMA node1 CPU(s):   16-31,80-95
NUMA node2 CPU(s):   32-47,96-111
NUMA node3 CPU(s):   48-63,112-127
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 xtopology 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 arat npt nrip_save vaes vpclmulqdq rdpid
```

Listing A.3: Output of the `cat /proc/meminfo` command

```
MemTotal:       1034754828 kB
MemFree:         3290352 kB
MemAvailable:   545989372 kB
Buffers:               0 kB
Cached:         537348304 kB
SwapCached:            0 kB
Active:         486301272 kB
Inactive:       531443264 kB
Active(anon):   480445920 kB
Inactive(anon):   360728 kB
Active(file):    5855352 kB
Inactive(file): 531082536 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:            573016 kB
Writeback:             0 kB
AnonPages:      480390744 kB
```

```
Mapped:           413680 kB
Shmem:            415188 kB
Slab:           11716932 kB
SReclaimable:   11314780 kB
SUnreclaim:       402152 kB
KernelStack:       72144 kB
PageTables:      1166284 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    517377412 kB
Committed_AS:   30520388 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:      593816 kB
DirectMap2M:    27299840 kB
DirectMap1G:    1023410176 kB
```

Listing A.4: Output of the `lshw -C memory` command

```
  *-firmware
       description: BIOS
       vendor: Amazon EC2
       physical id: 0
       version: 1.0
       date: 10/16/2017
       size: 64KiB
       capacity: 64KiB
       capabilities: pci edd acpi virtualmachine
  *-cache:0
       description: L1 cache
       physical id: 6
       slot: L1 - Cache
       size: 3MiB
       capacity: 3MiB
       clock: 1GHz (1.0ns)
       capabilities: pipeline-burst internal write-back unified
       configuration: level=1
  *-cache:1
       description: L2 cache
       physical id: 7
       slot: L2 - Cache
       size: 24MiB
       capacity: 24MiB
       clock: 1GHz (1.0ns)
       capabilities: pipeline-burst internal write-back unified
       configuration: level=2
  *-cache:2
       description: L3 cache
       physical id: 8
       slot: L3 - Cache
```

```
    32         size: 192MiB
    33         capacity: 192MiB
    34         clock: 1GHz (1.0ns)
    35         capabilities: pipeline-burst internal write-back unified
    36         configuration: level=3
    37    *-cache:0
    38         description: L1 cache
    39         physical id: 9
    40         slot: L1 - Cache
    41         size: 3MiB
    42         capacity: 3MiB
    43         clock: 1GHz (1.0ns)
    44         capabilities: pipeline-burst internal write-back unified
    45         configuration: level=1
    46    *-cache:1
    47         description: L2 cache
    48         physical id: a
    49         slot: L2 - Cache
    50         size: 24MiB
    51         capacity: 24MiB
    52         clock: 1GHz (1.0ns)
    53         capabilities: pipeline-burst internal write-back unified
    54         configuration: level=2
    55    *-cache:2
    56         description: L3 cache
    57         physical id: b
    58         slot: L3 - Cache
    59         size: 192MiB
    60         capacity: 192MiB
    61         clock: 1GHz (1.0ns)
    62         capabilities: pipeline-burst internal write-back unified
    63         configuration: level=3
    64    *-memory
    65         description: System Memory
    66         physical id: c
    67         slot: System board or motherboard
    68         size: 1TiB
    69       *-bank:0
    70            description: DIMM DDR4 Static column Pseudo-static Synchronous Window DRAM 3200 MHz
(0.3 ns)
    71            physical id: 0
    72            size: 256GiB
    73            width: 64 bits
    74            clock: 3200MHz (0.3ns)
    75       *-bank:1
    76            description: DIMM DDR4 Static column Pseudo-static Synchronous Window DRAM 3200 MHz
(0.3 ns)
    77            physical id: 1
    78            size: 256GiB
    79            width: 64 bits
    80            clock: 3200MHz (0.3ns)
    81       *-bank:2
    82            description: DIMM DDR4 Static column Pseudo-static Synchronous Window DRAM 3200 MHz
(0.3 ns)
    83            physical id: 2
    84            size: 256GiB
    85            width: 64 bits
    86            clock: 3200MHz (0.3ns)
    87       *-bank:3
    88            description: DIMM DDR4 Static column Pseudo-static Synchronous Window DRAM 3200 MHz
(0.3 ns)
    89            physical id: 3
    90            size: 256GiB
    91            width: 64 bits
    92            clock: 3200MHz (0.3ns)
```

B IO Performance

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

```
read_iops_test: (g=0): rw=randread, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio,
iodepth=256
fio-2.14
Starting 1 process
read_iops_test: Laying out IO file(s) (1 file(s) / 102400MB)
read_iops_test: (groupid=0, jobs=1): err= 0: pid=5191: Wed Nov 30 08:49:27 2022
  read : io=2781.7MB, bw=47468KB/s, iops=11862, runt= 60006msec
    slat (usec): min=1, max=4610, avg=81.47, stdev=204.32
    clat (usec): min=4788, max=28613, avg=21492.19, stdev=345.41
     lat (usec): min=5273, max=28615, avg=21574.97, stdev=346.36
    clat percentiles (usec):
     |  1.00th=[20864],  5.00th=[21120], 10.00th=[21120], 20.00th=[21120],
     | 30.00th=[21376], 40.00th=[21376], 50.00th=[21376], 60.00th=[21632],
     | 70.00th=[21632], 80.00th=[21632], 90.00th=[21888], 95.00th=[21888],
     | 99.00th=[22144], 99.50th=[22400], 99.90th=[23168], 99.95th=[24192],
     | 99.99th=[25984]
    lat (msec) : 10=0.01%, 20=0.05%, 50=99.98%
  cpu          : usr=1.59%, sys=4.18%, ctx=97147, majf=0, minf=1
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=0.1%, >=64=105.0%
     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=0.1%
     issued    : total=r=711841/w=0/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=256
Run status group 0 (all jobs):
   READ: io=2781.7MB, aggrb=47468KB/s, minb=47468KB/s, maxb=47468KB/s,
mint=60006msec, maxt=60006msec
Disk stats (read/write):
  nvme0n1: ios=747241/9058, merge=0/378, ticks=3690012/62044, in_queue=3680220,
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 SF30000 data set took 1hr27min when performed by the following command: `find /data/sf30000 -name "*.csv.gz" -print0 | parallel -q0 gunzip`

Listing C.1: Script to generate the SF30000 dataset for TigerGraph in AWS EMR. This dataset is used for the benchmark run

```
export SCALE_FACTOR=30000
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
```

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"
```

```
32 CREATE UNDIRECTED EDGE KNOWS (FROM Person, TO Person, creationDate INT, weight19 UINT, weight20 UINT
   DEFAULT
         10000)
33 CREATE DIRECTED EDGE LIKES (FROM Person, TO Comment|Post, creationDate INT) WITH
   REVERSE_EDGE="LIKES_REVERSE"
34 CREATE DIRECTED EDGE REPLY_OF (FROM Comment, TO Comment|Post) WITH REVERSE_EDGE="REPLY_OF_REVERSE"
35 CREATE DIRECTED EDGE STUDY_AT (FROM Person, TO University, classYear INT) WITH
REVERSE_EDGE="STUDY_AT_REVERSE"
36 CREATE DIRECTED EDGE WORK_AT (FROM Person, TO Company, workFrom INT) WITH REVERSE_EDGE="WORK_AT_REVERSE"
37
38 CREATE DIRECTED EDGE ROOT_POST (FROM Comment, TO Post) WITH REVERSE_EDGE="ROOT_POST_REVERSE" //FOR
   BI-3,9,17
39 CREATE DIRECTED EDGE REPLY_COUNT (FROM Person, TO Person, cnt UINT)
40
41 CREATE GLOBAL SCHEMA_CHANGE JOB addIndex {
42     ALTER VERTEX Country ADD INDEX country_name ON (name);
43     ALTER VERTEX Company ADD INDEX company_name ON (name);
44     ALTER VERTEX University ADD INDEX university_name ON (name);
45     ALTER VERTEX Tag ADD INDEX tag_name ON (name);
46     ALTER VERTEX TagClass ADD INDEX tagclass_name ON (name);
47 RUN GLOBAL SCHEMA_CHANGE JOB addIndex
48 CREATE GRAPH ldbc_snb (*)
```