TigerGraph

TigerGraph: Key Lessons from Running SNB BI Workload on Large-Scale Data Set

Mingxi Wu, SVP of Engineering

Aug 2024

Disclaimer: The 108TB benchmark is not official LDBC benchmark results, as it has not been audited.

Safe Harbor Statement

- This presentation does not represent an official LDBC Benchmark Results.

- Our work is derived from the LDBC SNB BI Benchmark Workload. The benchmark results presented herein are not audited, and we emphasize that these numbers do not constitute an official LDBC Benchmark test run.

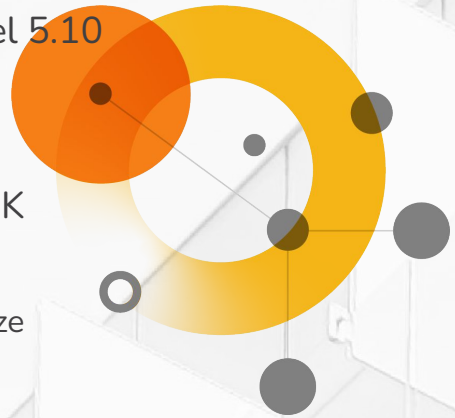
- The purpose of this presentation is to share our experiences and insights from conducting large-scale stress testing on TigerGraph's engine.

BI Workload

- **Read Query**
  - complex read queries
  - touching a significant portion of the data.
  - Choke-point based query design
    - Explosive and redundant multi-joins
    - Expressive path finding
- **Microbatches of refresh operations**
  - a set of insert and delete operations
  - batched for a given time period (e.g. a day)

Cluster Setup - Hardware

- **Number of Machines**: 72
- **Instance Type**: AWS r6a.48xlarge
- **Each Machine Configuration:**
  - **Operating System:** Amazon Linux 2 AMI (HVM) - Kernel 5.10
  - **CPU Type:** AMD® EPYC® 7R13 Processor @3.564 GHz
  - **Number of vCPUs:** 192
  - **CPU Cache**: L1d 32K, L1i cache 32K, L2 512k, L3: 32768K
  - **Memory:** 1536 G
  - **Disk Type**: AWS 6TB General Purpose SSD (gp3), ~4X of data size
  - **Disk Detail**: 16k Max IOPS; 1GB/s Max throughput
  - **Network Bandwidth**: 50GB/s
  - **EBS Bandwidth**: 40GB/s



Setup - Software

- TigerGraph Version: 3.7.0
- LDBC SNB Versions
  - Specification: 2.2.0
  - Data Generator: 0.5.0
  - Driver and implementations: 1.0.2

Schema and Data Inflation from SF-30K

Original Schema — Initial Snapshot(37T)

Dynamic (36T)		Static (<1T)	
Vertex	**Edge**	**Vertex**	**Edge**
Comment	CONTAINER_OF	Company	HAS_TYPE
Person	HAS_CREATOR	University	IS_LOCATED_IN
Post	HAS_INTEREST	City	IS_PART_OF
Forum	HAS_MEMBER	Country	IS_SUBCLASS_OF
	HAS_MODERATOR	Continent	
	HAS_TAG	Tag	
	IS_LOCATED_IN	TagClass	
	MESG_LOCATED_IN		
	KNOWS		
	LIKES		
	REPLY_OF		
	STUDY_AT		
	WORK_AT		

Dynamic x3

Triple Schema — Initial Snapshot(<109T)

3 x Dynamic (36T) = 108T		Static (<1T)	
Vertex	**Edge**	**Vertex**	**Edge**
Comment1	CONTAINER_OF’	Company	HAS_TYPE
Comment2	HAS_CREATOR’	University	IS_LOCATED_IN
Comment3	HAS_INTEREST’	City	IS_PART_OF
Person1	HAS_MEMBER’	Country	IS_SUBCLASS_OF
Person2	HAS_MODERATOR’	Continent	
Person3	HAS_TAG’	Tag	
Post1	IS_LOCATED_IN’	TagClass	
Post2	MESG_LOCATED_IN’		
Post3	KNOWS’		
Forum1	LIKES’		
Forum2	REPLY_OF’		
Forum3	STUDY_AT’		
	WORK_AT’		

Duplicate the dynamic group in original schema three times

Discussion

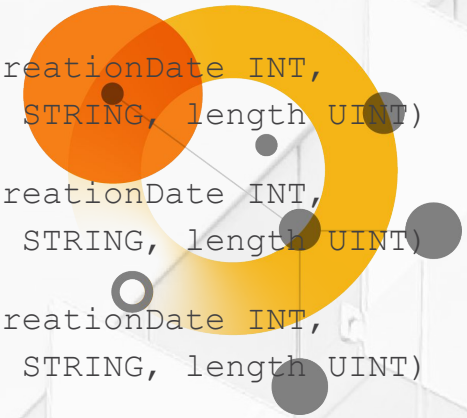
In essence, we duplicate dynamic vertex types, modify batch activation, and keep point activation intact.

- **If a query starts with all vertices of a dynamic type, we activate all**
  - SELECT p FROM (Post1|Post2|Post3):p WHERE p.language IN languages;
- **If a query starts with one dynamic vertex, we select one vertex**
  - CREATE QUERY bi10(VERTEX<Person1> personId, STRING country, STRING tagClass)
- **If a query starts with static vertex, we activate all dynamic vertices**
  - SELECT p
    FROM Country:cn -(<IS_PART_OF.<IS_LOCATED_IN)-
    (Person1|Person2|Person3):p
    WHERE cn.name == country AND p.creationDate < endEpoch;

Schema Setup for Dynamic Groups

- Dynamic Vertex: Vertex type is duplicated three times and indexed
  Example:
  - `CREATE VERTEX Comment1 (id UINT PRIMARY KEY, creationDate INT, locationIP STRING, browserUsed STRING, content STRING, length UINT)`
  - `CREATE VERTEX Comment2 (id UINT PRIMARY KEY, creationDate INT, locationIP STRING, browserUsed STRING, content STRING, length UINT)`
  - `CREATE VERTEX Comment3 (id UINT PRIMARY KEY, creationDate INT, locationIP STRING, browserUsed STRING, content STRING, length UINT)`

Note: Other dynamic vertex types are duplicated and indexed similarly.



Schema Setup for Dynamic Groups

- Dynamic Edge: FROM and TO vertex type are replaced with duplicated vertex types
  Example:
  - **CREATE DIRECTED EDGE** CONTAINER_OF (**FROM** Forum1|Forum2|Forum3 , **TO** Post1|Post2|Post3) WITH REVERSE_EDGE="CONTAINER_OF_REVERSE"
  - **CREATE DIRECTED EDGE** REPLY_OF (**FROM** Comment1|Comment2|Comment3 , **TO** Comment1|Comment2|Comment3|Post1|Post2|Post3) WITH REVERSE_EDGE="REPLY_OF_REVERSE"
  - **CREATE DIRECTED EDGE** LIKES (**FROM** Person1|Person2|Person3 , **TO** Comment1|Comment2|Comment3|Post1|Post2|Post3 , creationDate INT) WITH REVERSE_EDGE="LIKES_REVERSE"

Note: Dynamic vertex involved in edge definition are duplicated and indexed. Edge type name remains the same in queries.

Data Statistics

Vertex Cardinality (total: 217.86B)

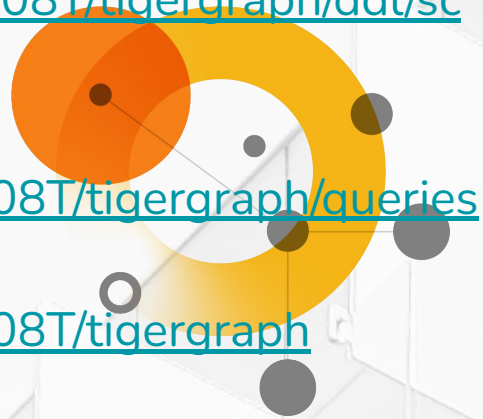
Vertex	Count
Comment1	58,666,958,815
Comment2	58,666,958,815
Comment3	58,666,958,815
Post1	13,148,296,221
Post2	13,148,296,221
Post3	13,148,296,221
Forum1	728,629,666
Forum2	728,629,666
Forum3	728,629,666
Person1	74,689,437
Person2	74,689,437
Person3	74,689,437
Company	4,725
University	19,140
City	4,029
Country	333
Continent	18
Tag	48,240
TagClass	213
Subtotal	217,855,799,115

Edge Cardinality (total: 1.62T)

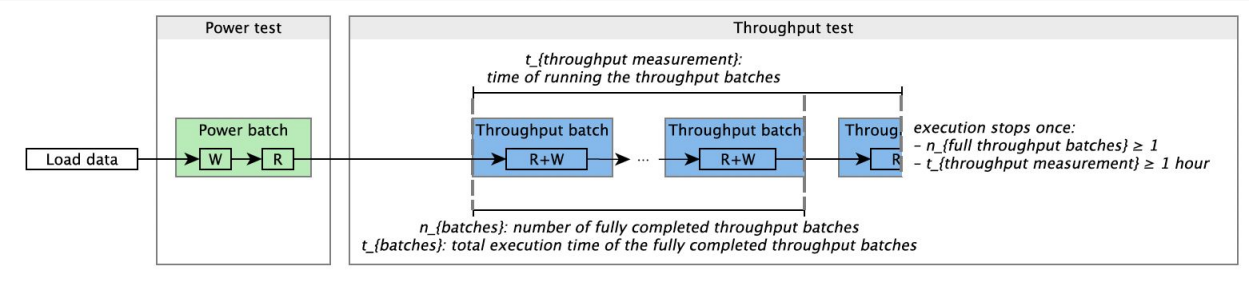
Edges	Count
CONTAINER_OF	39,444,888,663
HAS_CREATOR	215,445,765,108
HAS_INTEREST	5,243,002,503
HAS_MEMBER	271,956,270,042
HAS_MODERATOR	2,185,888,998
HAS_TAG	304,603,732,866
IS_LOCATED_IN	224,076,266
MESG_LOCATED_IN	215,445,765,108
KNOWS	17,203,410,066
LIKES	370,276,474,926
REPLY_OF	176,000,876,445
STUDY_AT	179,275,377
WORK_AT	487,556,766
HAS_TYPE	16,080
IS_PART_OF	1,454
IS_SUBCLASS_OF	70
Subtotal	1,618,697,000,738

Schema and Query Script

- Schema
  https://github.com/tigergraph/ecosys/blob/ldbc_108T/tigergraph/ddl/schema.gsql
- Query
  https://github.com/tigergraph/ecosys/tree/ldbc_108T/tigergraph/queries
- Driver and update
  https://github.com/tigergraph/ecosys/tree/ldbc_108T/tigergraph



Benchmark Workflow and Implementation



- The official benchmark consists of
  - load data,
  - power test
  - and throughput test
- This benchmark runs power and throughput test of 28 query variants with 5 substitution parameters, in total of 28 x 5 = 140 queries

Results -Loading

- **Loading time:** 12hr45min over 72 machines
- **Loaded topology data size on each machine:**
  - ~617 GB (about 44.4TB aggregate 72 machines)
- **Compress ratio**: 44.4/108*100%=41%
- **Loading Speed:** 117.6G/Hour/Machine

Performance Results

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/$
22.67 hr	106 034.76	9.78	26 398.01	2.43

Calculations based on LDBC Specifications

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

where $w = 15160$ 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 4.12).
The throughput score is calculated as

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

Performance Results

Operations in the **power test** for TigerGraph at 108TB dataset. Execution times are reported in seconds.

Operation	Time (hh:mm:ss)	Time (seconds)
Total read time (5 runs)	5:37:57	20,277.70
Total write time	4:12:40	15,160.01
Precomputation for Q4	0:06:22	382.08
Precomputation for Q6	0:15:59	959.31
Precomputation for Q14 and Q19	2:04:53	7,493.26
Precomputation for Q20	0:06:17	376.55

Detailed Results -Performance

Detailed power test results for TigerGraph at 108TB dataset. Execution times for five runs with different substitution parameters per query are reported in seconds.

Query	Sum (5 runs)	Max.	Min.	Mean
1	190.85	41.77	36.17	38.17
2a	1356.09	322.91	153.44	271.22
2b	688.63	161.29	125.84	137.73
3	1569.73	642.00	195.34	313.95
4	86.07	18.79	14.77	17.21
5	276.02	70.10	46.62	55.20
6	266.63	65.31	47.81	53.33
7	1068.72	223.63	199.79	213.74
8a	395.34	85.51	73.94	79.07
8b	227.08	50.60	41.31	45.42
9	790.82	162.24	150.31	158.16
10a	867.05	204.22	146.86	173.41
10b	353.49	100.48	32.47	70.70
11	211.01	45.90	40.27	42.20
12	759.12	168.89	136.31	151.82

Detailed power test results for TigerGraph at 108TB dataset. Execution times for five runs with different substitution parameters per query are reported in seconds.

Query	Sum (5 runs)	Max.	Min.	Mean
11	211.01	45.90	40.27	42.20
12	759.12	168.89	136.31	151.82
13	1780.49	362.29	351.08	356.10
14a	667.76	138.77	124.42	133.55
14b	284.12	66.48	46.09	56.82
15a	897.11	183.40	166.82	179.42
15b	3004.23	664.72	509.15	600.85
16a	1728.39	411.64	245.30	345.68
16b	418.36	86.47	79.16	83.67
17	548.34	113.44	104.09	109.67
18	1156.12	234.36	227.99	231.22
19a	262.11	55.64	49.01	52.42
19b	260.12	56.13	48.59	52.02
20a	77.12	36.20	8.87	15.42
20b	86.69	27.06	11.59	17.34

Conclusion

- TigerGraph efficiently handled deep-link OLAP queries on a graph with 217.9 billion vertices and 1.6 trillion edges. Eleven data-intensive queries returned results within one minute; the rest took 1 to 10 minutes.

- This benchmark showcases TigerGraph's capability to manage large-scale graph workloads with frequent incremental updates. To our knowledge, no other graph or relational database has demonstrated similar performance on such a scale.