TigerGraph

Using the Linked Data Benchmark Council Social Network Benchmark Methodology to Evaluate TigerGraph at 36 Terabytes

WHITE PAPER

Disclaimer

Although the results described in this report are derived from Linked Data Benchmark Council (LDBC) standards, they should not be considered to be LDBC benchmark results, since the current results have not been audited and approved by LDBC yet.

About TigerGraph

TigerGraph is the only scalable graph database for the enterprise. Based on the industry's first Native and ParallelGraph technology, TigerGraph unleashes the power of interconnected data, offering organizations deeper insights and better outcomes. TigerGraph fulfills the true promise and benefits of the graph platform by tackling the toughest data challenges in real time, no matter how large or complex the dataset. TigerGraph's proven technology supports applications such as fraud detection, customer 360, MDM, IoT, AI and machine learning to make sense of ever changing big data, and is used by customers including Amgen, China Mobile, Intuit, Wish and Zillow.

The company is headquartered in Redwood City, California, USA.
Follow TigerGraph on Twitter at @TigerGraphDB or visit www.tigergraph.com

WP-BTGULDBCSNBSF30K-01182022

Executive Summary

We recently measured TigerGraph's performance using the respected Linked Data Benchmark Council (LDBC) Social Network Benchmark (SNB) Scale Factor 30k dataset (36TB raw data, with 73 billion vertices and 534 billion edges).

To the best of our knowledge, this is the first benchmark test using the LDBC-SNB SF-30k BI workload on a distributed graph database.

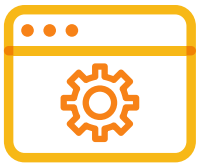
The study clearly demonstrates TigerGraph's ability to handle a big graph workload in a real production environment, where tens of terabytes of connected data with hourly or daily incremental updates is the norm—no other graph database vendor or relational database vendor has demonstrated equivalent analytical capabilities on such a large dynamic graph.

We focused on testing TigerGraph's performance on Business Intelligence (BI) workloads over a sequence of batch-refreshed big graphs using a distributed cluster of 40 machines in Google Cloud Platform (GCP). The BI workload included:

- **20 Read Queries**—the majority of OLAP-style iterative and deep-link graph queries were answered in a few minutes or less. Some queries require compute the edge weights first and then compute the top-k weighted shortest paths between two vertex sets to find the answer, which is very challenging.
- **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, and we run the BI queries after seven consecutive batches.

We did not include interactive complex (IC) and interactive short (IS) query workloads in this benchmark effort as the data generator is not ready yet for them in the latest LDBC SNB version. See here for more information.

TigerGraph's ability to handle a big graph workload in a real production environment, where tens of terabytes of connected data with hourly or daily incremental update is the norm



Benchmark Setup

This document describes an implementation of the LDBC Social Network Benchmark Scale Factor 30k (version 0.4.0) on a distributed cluster. The implementation used GSQL, a query language developed by TigerGraph. The queries were compiled and loaded into the database as stored procedures.

The data schema follows the property graph data model. We measured the loading time, storage size, and query latency of the 20 BI queries on a cluster of 40 machines. All benchmark scripts are publicly available on GitHub for reproducibility purposes. To the best of our knowledge, this is the first benchmark test using the LDBC-SNB SF-30k BI workload on a distributed graph database.

Machine Overview

We used 40 Google Cloud Platform (GCP) machines for benchmarking LDBC-SNB Scale Factor 30k. Table 1.1-1.3 shows the operating system and hardware used.

Number of Virtual Machines	40
Instance Type	m1-ultraman-40
Operating System	CentOS Linux 7
Kernel	linux 3.10.0-1160.41.1.el7.x86_64

Table 1.1: Overview of the testing cluster.

CPU Details

The following table provides the CPU and cores specifications.

Architecture	x86_64
Number of CPUs (threads)	40
Model Name	Intel Xeon E7 @ 2.20GHz
Socket(s)	1
Cores per Socket	20
Threads per Core	2
Cache Size per CPU	55MB
CPU Max (GHz)	3.3

Table 1.2: CPU details for each machine.

Memory Details

The memory size is 961GB for each machine (approximately 37.5TB for the complete cluster of 40 machines).

Disk and Network Details

The following table provides the disk information.

Disk	GCP regional persistent disk
Device Size	4TB
IO Speed	30 IOPS/GB
Network Bandwidth	32 Gbps

Table 1.3: Disk details on each machine.

TigerGraph Details

We used TigerGraph Enterprise Edition 3.2.2 throughout the benchmark test.

Dataset Information

Data Schema

The data schema used in the benchmark test is shown in the following schematic.

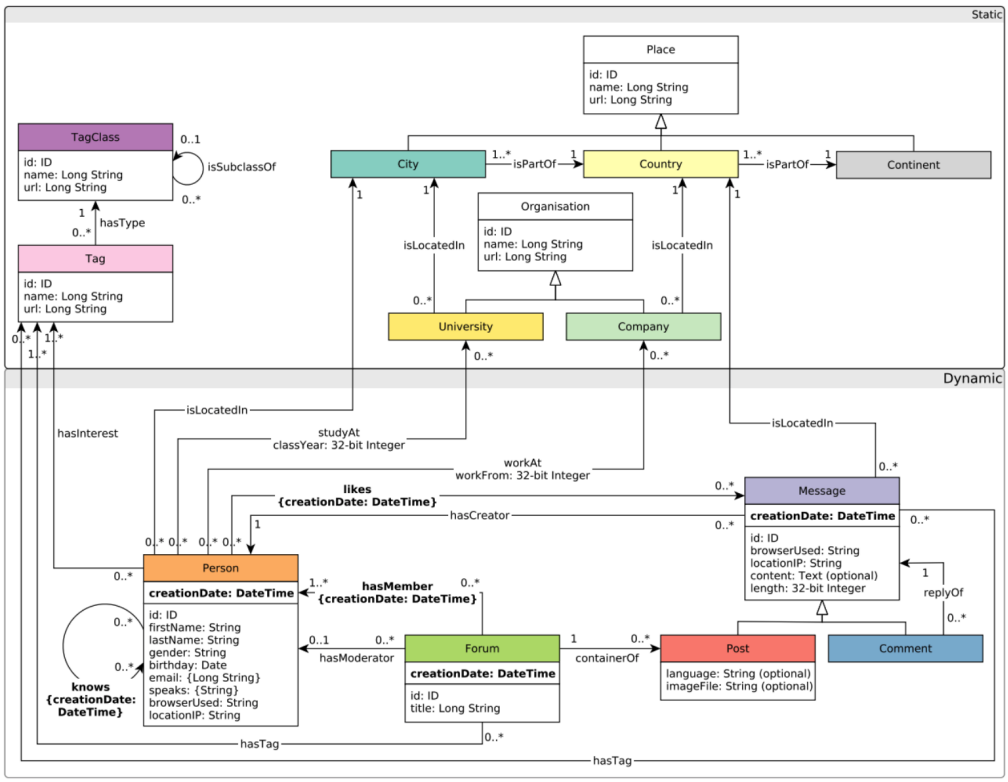


Figure 2.1: **Data schema from LDBC SNB Spec.**

Dataset Statistics

The statistics of the initial state for each vertex and edge type is shown in the following table. (The data size was 10-20% larger after a refresh operation, and is reported in subsequent sections):

DYNAMIC		STATIC	
VERTEX TYPE NAME	CARDINALITY	VERTEX TYPE NAME	CARDINALITY
	(# OF VERTICES)		(# 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 2.2: **Cardinality for each vertex type (total 72.62B vertices).**

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	60,105,627,162
KNOWS	11,468,940,044
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

Table 2.3: Cardinality for each edge type (total 533.5B edges).

Initial Data Loading

The schema DDL and loading scripts can be found on GitHub. The scripts are organized in the following structure:

- gcp/ - sets up the GCP cluster.
- LDBC_10T/ - downloads and preprocesses data for SF 1k, 10k and 30k.
- schema.gsql - defines the data schema.
- load.gsql - loads the initial snapshot of the social network.
- queries/ - runs queries in Interactive Workloads (14 IC and 4 IS queries) and Business Intelligence Workloads (20 BI queries).
- refreshes/ - performs micro-batches of insert and delete operations.
- driver.py - the utility script to run the whole benchmark in a single command.
- cypher/ - cypher scripts for cross validation at SF-1 (1GB).

The command for loading was:

./driver.py load all [data_dir]

In the 36.1TB benchmark, 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

Table below reports loading performance of TigerGraph on a 40-node cluster:

Loading Time	35 hour and 30 minutes
Raw Size	36.13TB
Loaded Data Size on each Machine	454.2GB on average (and 17.7TB in total)
Compression Ratio (size of input data / size of TigerGraph store)	2.037
Loading Speed/Machine	26.06 GB/hour/machine (36.13TB/35.5 hour = 1042GB/hour for 40 machines)

Table 3.1: Loading performance on a 40-node cluster.

The compression ratio is shown in the following figure:

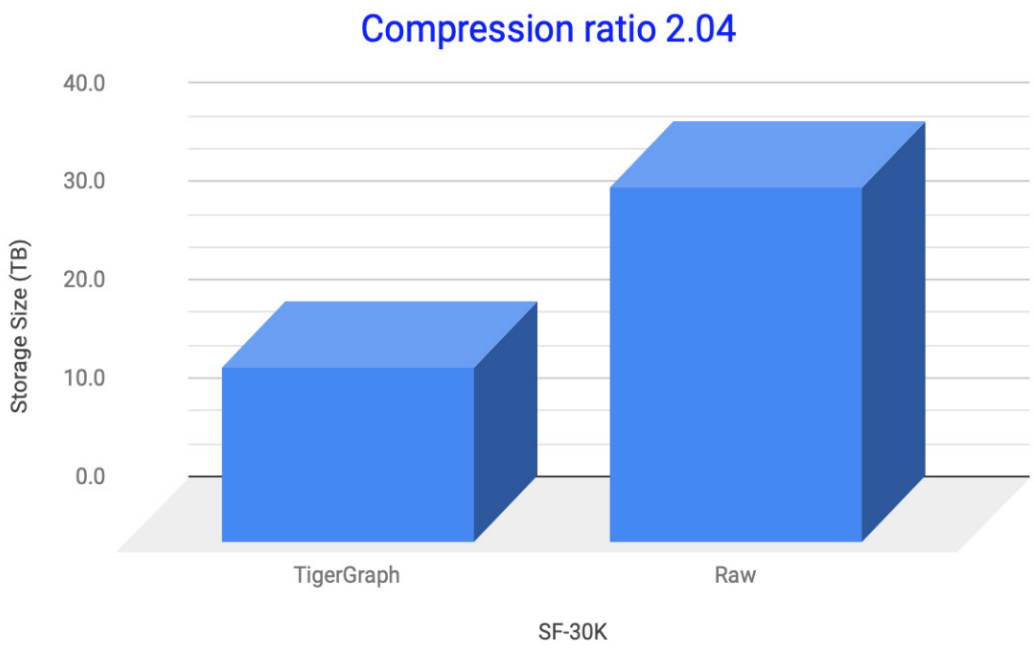


Figure 3.1: The disk space consumed by the dataset was 2.04 times smaller once loaded into TigerGraph.

As the figure shows, a 2.04x compression ratio was achieved when loading the raw data into TigerGraph native graph storage. And on average, we experienced a 26GB loading speed per hour per machine. The total end-to-end loading time was approximately 35.5 hours.

Cross-validation At SF-1

Before we conducted the LDBC SNB benchmark on SF-30k, we ran the same workloads using Neo4j on 1GB data (SF-1). The Neo4j Cypher queries are based on the implementation by LDBC (we made some modifications to support a different format of timestamp), which has been cross-validated against PostgreSQL. We used it as a sanity check to ensure different query language (GSQL, Cypher, SQL) implementations are logically equivalent and generate the same result on the same data set and query parameter input.

For both TigerGraph and Neo4j on SF-1, we completed the following workflow: first, we loaded the initial snapshot of the social network data dated 2012-09-13; next, we mutated the loaded graph by running insert and delete operations batched by each day.

- Insert operation is either insert a vertex or an edge
- Delete operation is either delete a vertex or an edge; vertex deletion triggers cascade deletion

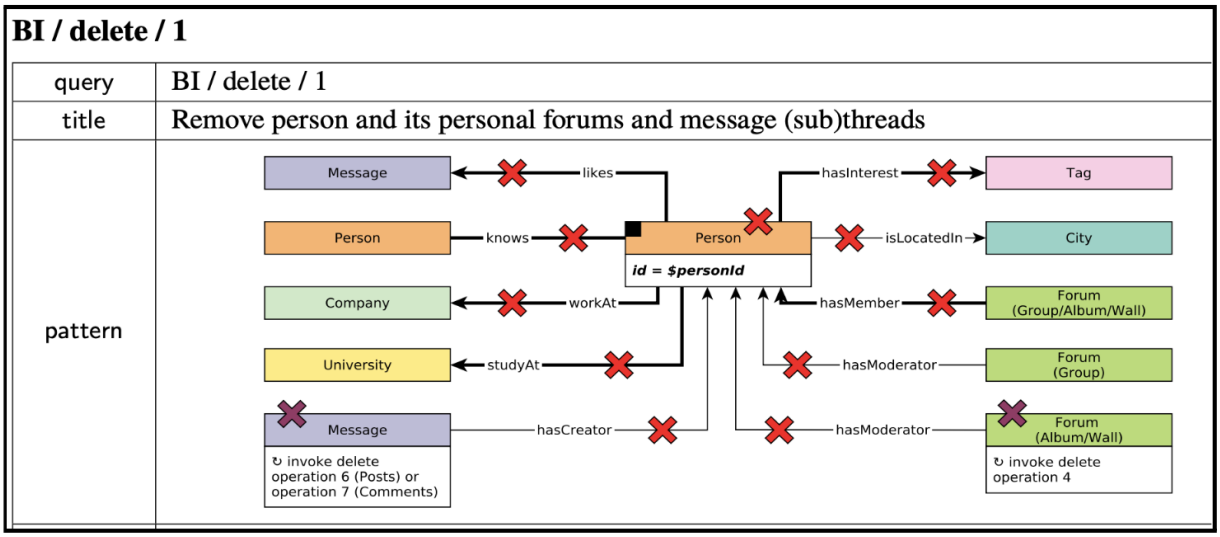


Figure 4.1: Deleting one Person vertex triggers the deletion of some connected vertices and edges according to the LDBC spec.

After every 30 batch updates, we cross-validated the query results and also reported the number of vertices and edges of the graph. The TigerGraph query results on four different dates are validated against the results using Neo4j. All results are the same across the two graph databases. We reported below the cross validation between TigerGraph and Neo4j.

- Validate the graph statistics - we cross validated the number of vertices and edges for some of the dynamic vertex and edge types
- Validate the query results - the results for all 20 BI queries are cross validated per 30-day sequential updates

SF1	TG3.2							
date	Comment	nPost	Forum	Person	HAS_TAG	LIKES	KNOWS	REPLY_OF
2012-09-13	1,116,485	999,664	90,227	9,884	2,471,902	1,265,881	120,835	1,116,485
2012-10-13	1,355,849	1,055,204	95,012	10,159	2,820,506	1,471,700	138,487	1,354,664
2012-11-12	1,645,123	1,111,077	99,994	10,422	3,194,742	1,715,430	159,946	1,642,590
2012-12-12	2,060,884	1,168,357	104,997	10,693	3,710,874	2,043,627		2,056,633
SF1	Neo4j							
date	Comment	Post	Forum	Person	HAS_TAG	LIKES	KNOWS	REPLY_OF
2012-09-13	1,116,485	999,664	90,227	9,884	2,471,902	1,265,881	120,835	1,116,485
2012-10-13	1,355,849	1,055,204	95,012	10,159	2,820,506	1,471,700	138,487	1,354,664
2012-11-12	1,645,123	1,111,077	99,994	10,422	3,194,742	1,715,430	159,946	1,642,590
2012-12-12	2,060,884	1,168,357	104,997	10,693	3,710,874	2,043,627	188,572	2,056,633

Table 4.2: The cardinality of selected vertex and edge types after each batch update, cross validated between TigerGraph and Neo4j at SF-1.

Micro-batch Insert and Delete Performance At SF-30K

The initial snapshot of the graph is dated 2012-11-29 (the data set's timestamp). BI queries for read operations were performed after every seven-day (or seven batch) updates of insert and delete operations.

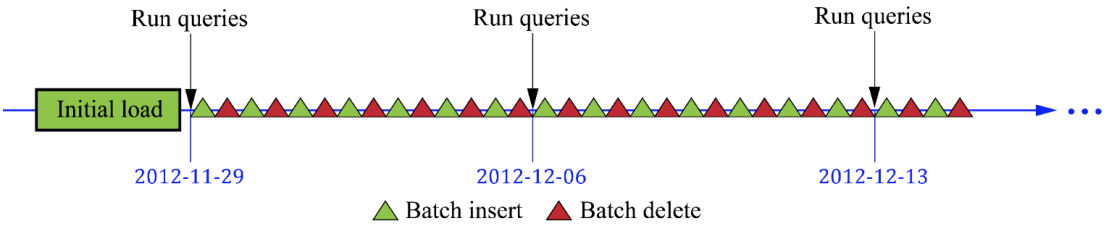


Figure 5.1: Workflow of LDBC SNB BI workloads with batch inserts and deletes

Insert/Delete Implementation

In the LDBC SNB benchmark, BI workloads also included the micro-batches of insert and delete operations. The insert and delete data was batched for a time period of one day. The initial data set was the graph snapshot dated on 2012-11-29. Then, after each day, we inserted a batch of vertices and edges and then deleted a batch of vertices and edges. After seven batch operations, we evaluated the query results. The reported insert and delete times shown in Table 5.1 are the total times of seven batch operations of insert and delete, respectively.

The inserted data was equally distributed on each machine and the insert operations were executed by running a loading job. This loading job was similar to the one that loads the initial snapshot, but only loaded the dynamic vertices and edges files.

The vertex delete was specified in the LDBC SNB v0.4.0 section 6.2.2. When a vertex (Person, Forum, etc.) was to be deleted, the related vertices were also required to be deleted (cascading delete). Vertex deletion was accomplished using GSQL queries. The vertex data was

loaded into the queries using "LoadAccum" function in GSQL and did not support distributed loading. We maintained the same copy of the delete data on all the nodes. The edge deletion was accomplished by running a loading job. Edge deletion operation was performed in a distributed way.

In summary, the insert operations and edge delete operations were distributed but the vertex delete specified in LDBC specification was more complicated (see Figure 4.1) and was not distributed in the current implementation.

The delete data set was much smaller than the insert data set and, therefore, the graph size increased after each batch update.

DATE	COMMENT	PERSON	LIKES	KNOWS	INSERT TIME(S)	INSERT TIME(S)
2012-11-29	58,666,958,815	74,689,437	123,425,491,642	11,468,940,044	0	0
2012-12-06	60,357,657,600	75,152,321	125,962,466,995	11,737,818,931	2693.37	2693.37
2012-12-13	62,146,221,013	75,621,563	128,626,905,655	12,035,032,350	3102.28	3102.28
2012-12-20	64,044,718,098	76,087,874	131,355,659,487	12,360,433,612	3326.13	3326.13
2012-12-27	66,129,264,865	76,551,431	134,356,124,333	12,770,202,615	3643.33	3643.33
			Total Time (s)		12765.11	7141.06
			Speed		25.99 GB/hr/ machine	138,806 operations/ sec

Table 5.1: Selected vertex and edge type cardinality at checkpoint dates; the batch insertion and deletion aggregate time at each checkpoint dates; and the total average speed of insertion and deletion. Note that since delete vertex triggers cascading delete, we are showing the number of delete operations per second (not counting the cascading operations). The first row insert and delete time is zero, since this was immediately following the initial loading, no insert and delete operations were performed.

Query Performance at SF-30k

Query Implementation

After cross validation of the GSQL query implementation of the BI benchmark queries, we ran them across the SF-30k data. The driver automatically generated the valid input parameters for the benchmark queries. The queries were written in GSQL v2 syntax using multi-hop pattern matching style. All of the queries ran in a distributed manner to obtain a scalable performance on the 40-node cluster.

We only ran BI workloads in the LDBC-SNB version 0.4.0. We are able to run IC and IS queries but the data generator developed by LDBC was not ready yet for IC and IS workloads for scale larger than 1000.

Since the graph is updated after the insert and delete operations, we developed a parameter generator to get valid query input parameters. The parameter generator used GSQL queries to get candidate values for

input parameters (country names, tag names, Person IDs and city IDs, etc.). We then chose one value from the candidates at random and passed it to the BI queries. The parameters (stored in JSON format) can be passed to driver.py with "-p" options. This allows the repeatability of queries with the same parameter as the previous runs.

Query Performance

The following table shows the query elapsed time at batch update completion points.

QUERY TIME (S)	2012-11-29	2012-12-06	2012-12-13	2012-12-20	2012-12-27
BI 1	38.44	142.76	127.78	135.59	153.28
BI 2	13.04	74.57	89.73	70.68	144.19
BI 3	44.12	179.29	874.02	75.97	86.03
BI 4	139.45	244.46	339.52	375.07	359.66
BI 5	6.87	8.64	12.37	13.66	14.6
BI 6	27.76	31.07	32.98	34.88	33.78
BI 7	8.88	10.76	11.48	13.43	15.18
BI 8	9.6	10.33	11.11	14.62	16.51
BI 9	89.42	108.2	133.84	105.15	138.55
BI 10	333.17	588.69	869.56	473.58	485.22
BI 11	9.21	14.45	15.84	15.71	15.33
BI 12	10.76	33.05	23.55	25.42	27.02
BI 13	65.3	88.71	89.11	104.6	505.27
BI 14	614.47	1039.34	1344.96	2127.02	4128.6
BI 15	68.46	673.23	709.47	722.63	693.71
BI 16	205.13	233.01	246.13	258.18	271.73
BI 17	40.42	39.83	41.39	44.75	44.7
BI 18	7.21	6.94	7.11	7.09	7.31
BI 19	1373.93	1997.92	882.47	2608.47	1320.73
BI 20	43.92	35.55	41.36	47.18	49.86
Total Average Time (s)	**236.06**	**273.71**	**372.11**	**369.36**	**293.54**

Table 6.1: The time spent by the BI read queries in the LDBC BI SF-30k benchmark in seconds. The BI 19 query is a new query pattern which is very expensive at SF-30k.

Remarks:

- All the queries were written in the distributed mode and in v2 syntax
- Input parameters of queries were different for different checkpoints
- BI 19 is a heuristic approximation query. This is a new query pattern, which required us to compute top-k weighted shortest paths between two vertex sets using derived edge weights. At SF-30k this query is too expensive. We used a heuristic approximation and only searched for paths within length 2—this provided the correct results for LDBC SNB 36TB data, but may not achieve the correct results for smaller data size (e.g. SF-1 and SF-100). The exact solution of BI 19 took about one minute for SF-100 data, but yielded "out-of-memory" errors at SF-30k.

Conclusion

The benchmark test described in this report demonstrates TigerGraph's capability in handling large scale updatable connected data with a set of demanding graph benchmark queries.

The LDBC SNB new version (0.4.0) BI workloads included two new challenges:

- Micro-batch of insert and delete operations to mutate the current graph
- New query patterns that require compute top-k cheapest paths between two vertex sets, where the edge weights are derived from the graph

TigerGraph can run deep-link OLAP style queries on this mutable big graph of 72 billion vertices and 533 billion edges, returning results on data-intensive queries in a few minutes or less.

The study clearly demonstrates TigerGraph's ability to handle big graph workload in a real production environment, where tens of terabytes of connected data with hourly or daily incremental update is a norm. No other graph database vendor or relational database vendor has demonstrated equivalent analytical and operational capabilities on this large scale updatable graph to the best of our knowledge.

TigerGraph can run deep-link OLAP style queries on this mutable big graph of 72 billion vertices and 533 billion edges, returning results on data-intensive queries in a few minutes or less.

About TigerGraph

TigerGraph is the only scalable graph database for the enterprise. Based on the industry's first Native and ParallelGraph technology, TigerGraph unleashes the power of interconnected data, offering organizations deeper insights and better outcomes. TigerGraph fulfills the true promise and benefits of the graph platform by tackling the toughest data challenges in real time, no matter how large or complex the dataset. TigerGraph's proven technology supports applications such as fraud detection, customer 360, MDM, IoT, AI and machine learning to make sense of ever changing big data, and is used by customers including Amgen, China Mobile, Intuit, Wish and Zillow.

The company is headquartered in Redwood City, California, USA.
Follow TigerGraph on Twitter at @TigerGraphDB or visit www.tigergraph.com

WP-BTGULDBCSNBSF30K-01182022