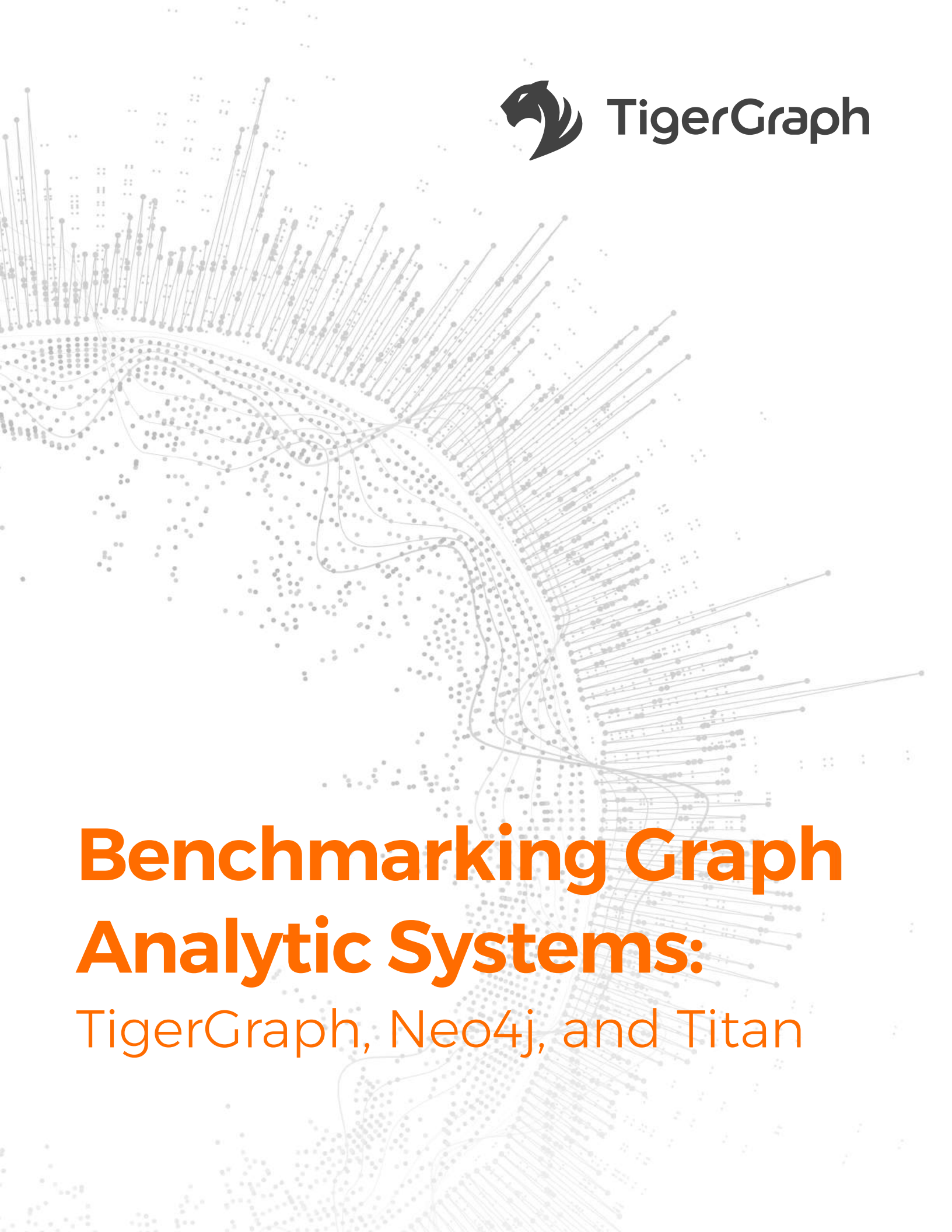

TigerGraph

Benchmarking Graph Analytic Systems:

TigerGraph, Neo4j, and Titan

Executive Summary: As the world's first Native Parallel Graph, the TigerGraph™ system loads, stores, and queries data faster than both Neo4j and Titan.

- TigerGraph has **4x to 100x faster graph traversal and query response times compared to both Neo4j and Titan**, when using a single machine.
- With TigerGraph, **query speed increases almost linearly with the number of machines.** Even on PageRank, requiring massive node-to-node communication, TigerGraph still achieves 4.7x speedup with 8x machines.
- TigerGraph's inherent parallelism enables it to **load data online 50x faster than Neo4j and 25x faster than Titan,** even when using a single machine.
- TigerGraph's high compression reduces storage to about **1/5 that of both Neo4j and Titan.**

This benchmark study examines the data loading and query performance of TigerGraph, Neo4j, and Titan. Benchmark topics include the following:

- Data Loading
  - Loading capabilities
  - Loading time and speed
  - Storage size of loaded data
- Query Performance
  - Query response times - for K-Neighborhood
  - Query response times - for Weakly Connected Components
  - Scalability of query response times

1. Benchmark Setup

Graph database/analytics systems tested:

- TigerGraph (formerly GraphSQL) 0.8
- Neo4j 3.1.3 Community Edition
- TitanGraph 1.0.0

Hardware platform

This benchmark uses an Amazon EC2 instance type. It is large enough for the test datasets and balances strong computational capability with economy.

- Virtual machine: Amazon EC2, C3.8xlarge instance type
  - 32 vCPUs, equivalent to 108 ECU
  - 60 GiB memory and 2 x 320 GB SSD disk
- Additional storage: 1TB Amazon EBS

Table 1 - Data sets

NAME	DESCRIPTION AND SOURCE	VERTICES	EDGES
graph500-22	Synthetic graph http://graph500.org	2.4 M	64 M
twitter_rv.net	Twitter user-follower directed graph http://an.kaist.ac.kr/traces/WWW2010.html	41.6 M	1.47 B

For each graph, the raw data are formatted as a single tab-separated edge list:

U1 U3

U1 U4

U2 U3

Vertices do not have attributes so there is no need for a separate vertex list.

Data Loading Tests

The data loading tests examined these three areas:

- Loading capabilities
- Loading time and speed
- Storage size of loaded data

LOADING CAPABILITIES

Before considering performance, we first compare functional capabilities. This is necessary to learn how to optimize the loading for each graph data platform.

TigerGraph composes a loading job in its native GSQL language thereby creating a RESTful endpoint. The job can then be run either through the GSQL shell or by submitting an HTTP request to the RESTful server. All loading takes place online while other data access services are running. The RESTful direct request is used for this test.

Neo4j offers two loaders: offline loading using neo4j-import[1] and online loading with the LOAD CSV Cypher command[2] . Their capabilities are quite different. Both methods were tested for this Benchmark.

Titan provides system settings and advice for performing bulk loading[3] , but it does not offer a native file-to-DB loading language. We used Cassandrathrift as the storage backend and wrote a custom loading procedure in Java using the Tinkerpop Graph Structure API.

[1] https://neo4j.com/docs/operations-manual/current/tutorial/import-tool/
[2] http://neo4j.com/docs/developer-manual/current/cypher/clauses/load-csv/
[3] http://s3.thinkaurelius.com/docs/titan/0.5.4/bulk-loading.html

Table 2 - Comparison of loading capabilities

FEATURE	TIGERGRAPH	NEO4J-IMPORT	NEO4J-CYPHER	TITAN
Built-in loading language	YES	YES	YES	NO
Bulk online loading (100M to 100B+ edges)	YES	NO	NO	YES
Incremental data loading	YES	NO	YES	YES
Index built during loading	YES	NO	YES	YES
Vertex ID deduplication	YES	NO	YES	NO

✓ TigerGraph offers the most convenient loading environment, with bulk online loading, a built-in loading language, and built-in indexing.

LOADING TIME AND SPEED

To get good performance in Neo4j-import, a schema index must be created separately. The total loading time for Neo4j includes both index creation and data loading. The time for creating the index alone is shown in parentheses in Table 3. As the loading progressed, Neo4j Cypher's loading got slower and slower as it checked for duplicate vertex IDs. Neo4j Cypher did not complete the loading test within 24 hours, so the loading was halted and loading on a smaller dataset was attempted.

Table 3 - Data sets

NAME	TIGERGRAPH	NEO4J-OFFLINE	NEO4J-CYPHER	TITAN
graph500-22	139 s	(13s) 116 s	Did not complete in 24 hours	3002 s
twitter_rv.net	1815 s	(135s) 3739 s	Did not complete in 24 hours	44654 s

The chart below presents these loading times.

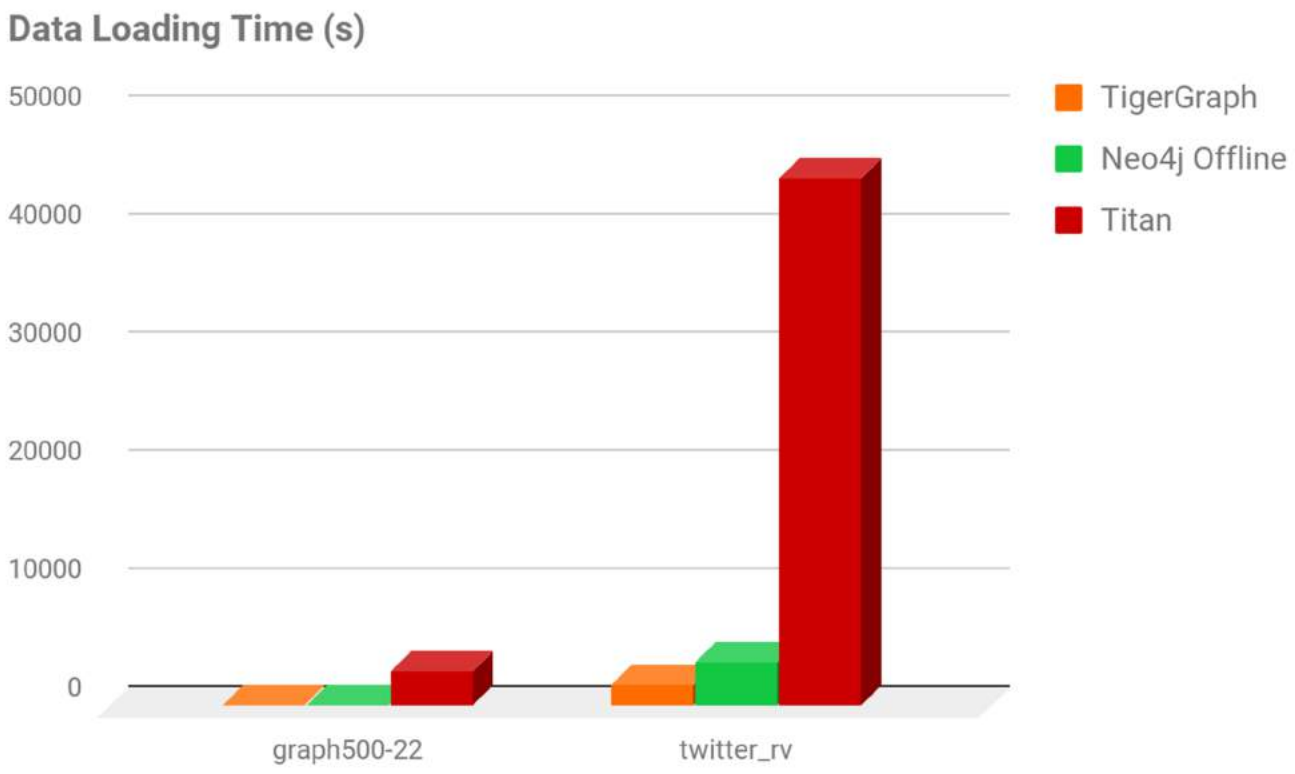

Figure 1 - Normalized data loading times

√ TigerGraph is 2x faster than Neo4j-offline and 25x faster than Titan at loading the large dataset twitter-rv.

By reducing the dataset size to 1 million lines, Neo4j Cypher's loading speed (number of data lines/second) became measurable. Periodic batch commit, with batch size = 100,000,

was used to optimize Neo4j's online loading performance. Table 4 includes the data loading speed results of both TigerGraph and Neo4j Cypher using this smaller dataset.

Table 4 - Data loading speed

DATASET	TIGERGRAPH	NEO4J CYPHER
twitter-rv (first 1M lines)	809K lines/sec	14K lines/sec

The chart below presents these loading times.

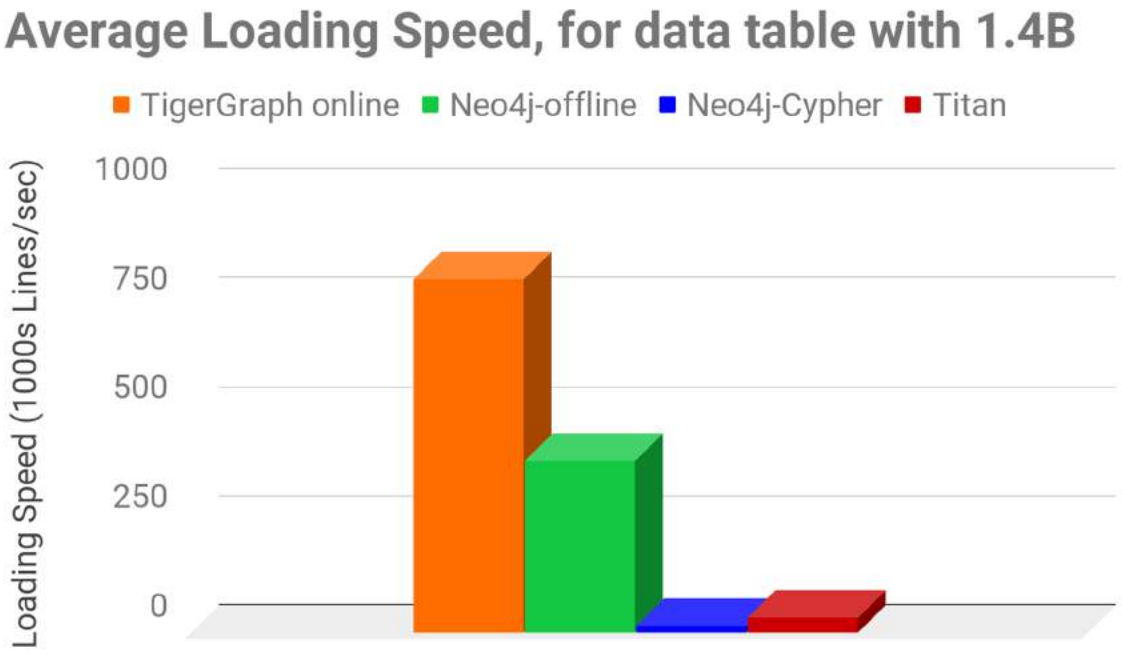

Figure 2 - Average Loading Speed

✓ TigerGraph loads 1M edges more than 50x faster than Neo4j Cypher.

✓ Neo4j Cypher did not load even the smallest data set (64M edges).

STORAGE SIZE OF LOADED DATA

The size of the loaded data is an important consideration, especially for in-memory

database systems. TigerGraph automatically encodes and compresses data, reducing the raw data size for both data sets to less than half its original size. In contrast, both Neo4j and Titan actually expand and more than double the original data size. While the two Neo4j loading methods operate differently, the resulting stored data is the same.

Table 5 - Loaded sata storage size

DATASET	RAW DATA	TIGERGRAPH	NEO4J	TITAN
graph500-22	967 MB	454 MB	2,369 MB	2,515 MB
twitter_rv.net	24,375 MB	11,038 MB	51,047 MB	50,028 MB

✓ TigerGraph stores large datasets in approximately 1/5 the space used by both Neo4j and Titan.

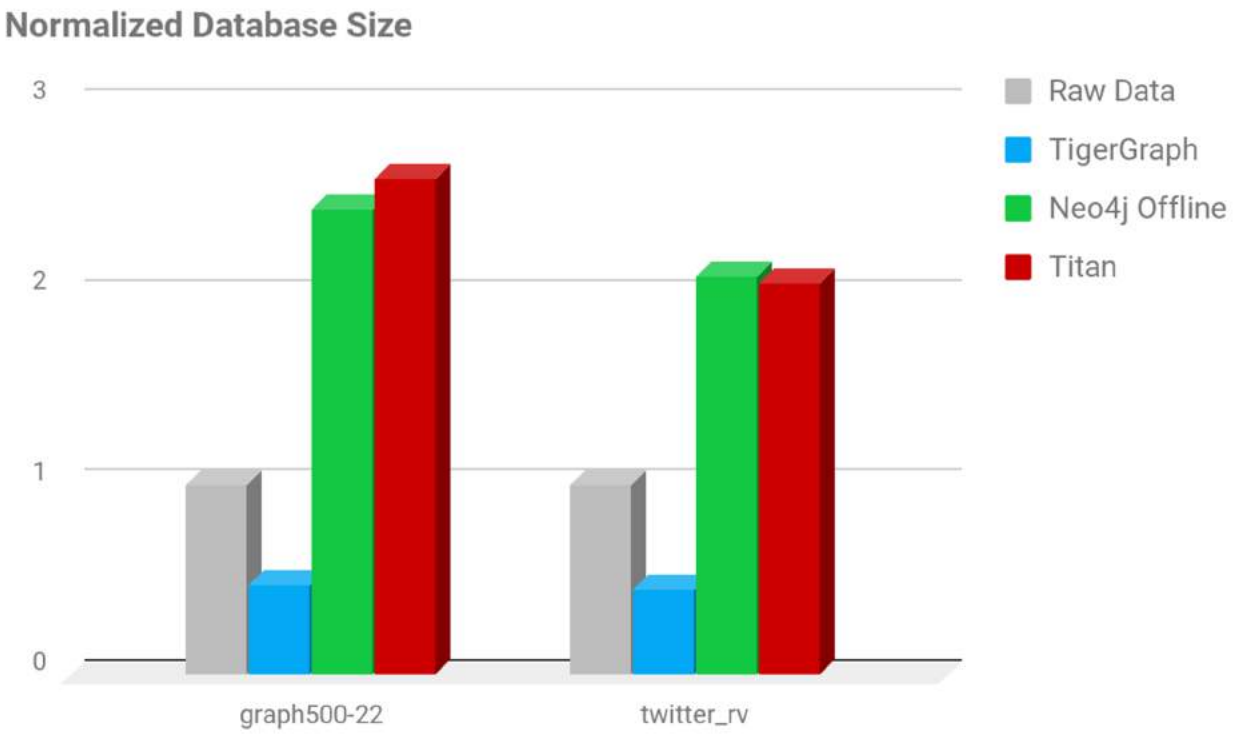

Figure 3 - Normalized database size after loading

DATA LOADING SUMMARY

✓ Only TigerGraph can do online data loading for large-scale data.

✓ TigerGraph's high compression reduces storage to about 1/5 that of both Neo4j and Titan.

3. Query Performance Tests

The Query performance tests examined these three areas:

- Query response times - for K-Neighborhood
- Query response times - for Weakly Connected Components
- Scalability of query response times

QUERY RESPONSE TIMES - FOR K-NEIGHBORHOOD

The k-neighborhood query, which asks for all the vertices which are within k hops of a starting vertex, is a good measure of graph traversal performance. The one-neighborhood of a vertex is simply its set of adjacent vertices. We compute the average query time for 1000 randomly selected starting vertices in each dataset. Each starting vertex has a different neighborhood size N; we report the average N. Query execution time is measured from the client side (clients run on the same machine as the database), which includes the network transfer time. To focus on the graph traversal and to minimize the network output time, we output only the size of the k-neighborhood, instead of the complete list of vertices.

Both TigerGraph and Neo4j can implement the k-neighborhood query efficiently. TigerGraph allows the user to adjust the number of threads and in these tests 16 threads were used. Titan uses the Gremlin query language and the only reasonable way we found to implement k-neighborhood was by using the subgraph API. The tests reveal that the subgraph API is very resource-heavy, making it very difficult to run the 2-neighborhood query to completion. Instead, Titan performs noticeably better when k-neighborhood is computed using the set of vertices that are exactly k steps from the source. The below tables include the query times for TigerGraph, Neo4j, Titan using true k-neighborhood, and Titan using the k-shell approximation.

Table 6 - One-Neighborhood query time

DATASET	AVG N	TIGERGRAPH	NEO4J	TITAN (K-NEIGH)	TITAN (K-SHELL APPROX)
graph500-22	4082	3.3 ms	14.4 ms	3064.9 ms	14.2 ms
twitter_rv.net	12271	7.5 ms	55.0 ms	34.1 ms	39.8 ms

Table 7 - Two-Neighborhood query time

DATASET	AVG N	TIGERGRAPH	NEO4J	TITAN (K-NEIGH)	TITAN (K-SHELL APPROX)
graph500-22	457400	0.19 s	19.0 s	Out of mem	11.1 s
twitter_rv.net	1747130	0.77 s	21.2 s	61.6 s	58.7 s

The one-neighborhood query time for Titan for the graph500-22 dataset has been omitted from the chart below. The response time was so high (3064.9 ms) and far out of range of the other values that it would make the chart unreadable.

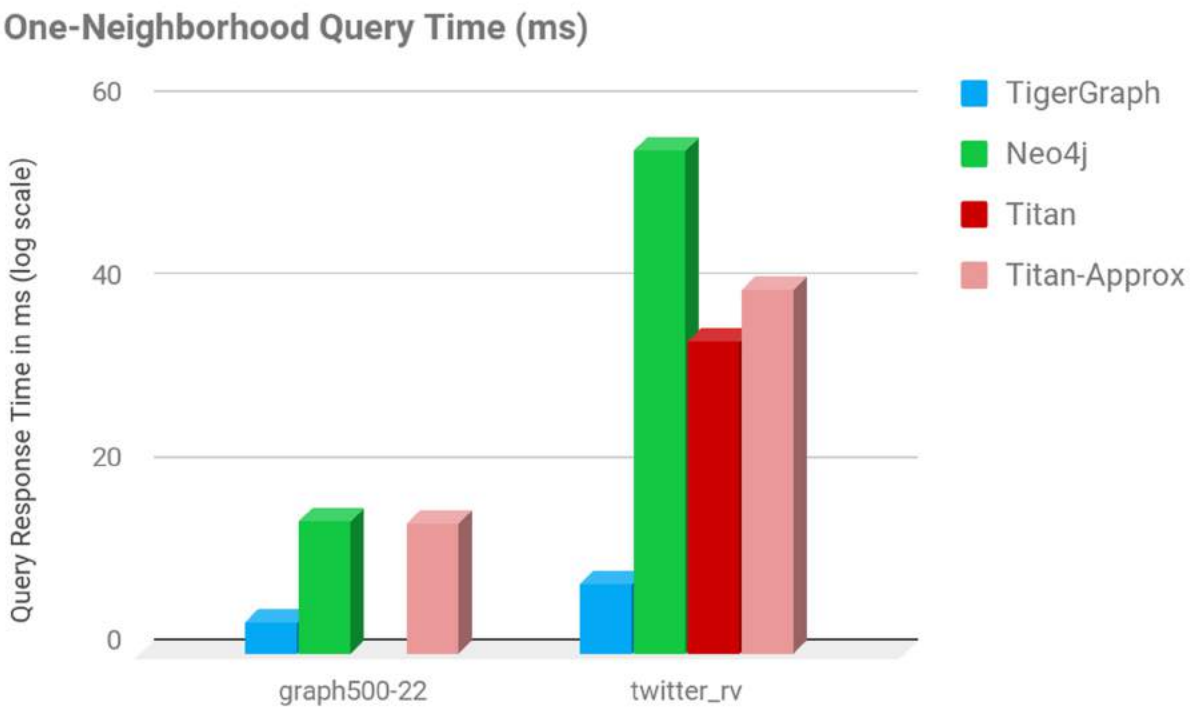

Figure 4 - One-neighborhood query time

In the two-neighborhood case, Titan ran out of memory on the graph 500-22 dataset.

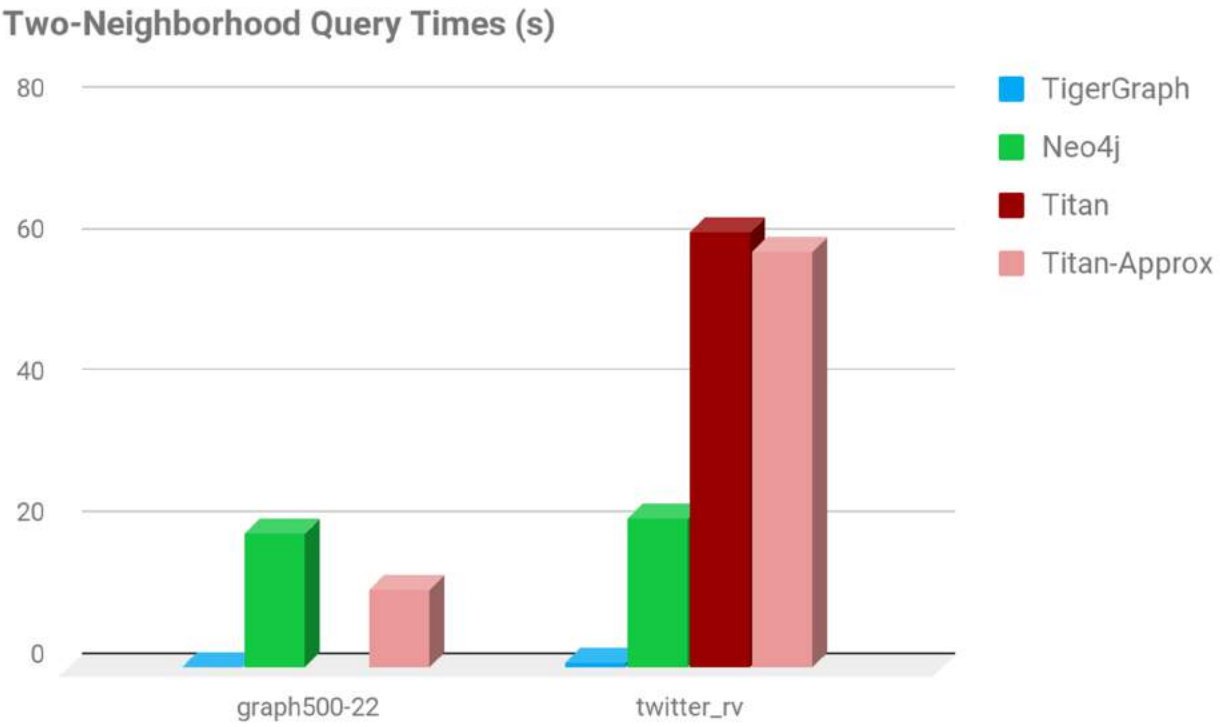

Figure 5 – Normalized Two-neighborhood query time

✓ TigerGraph is 4x to 7x faster than Neo4j on the one-neighborhood query and 27x to 100x faster than Neo4j on the two-neighborhood query.

✓ Titan is 900x slower than TigerGraph on the one-neighborhood query, and could not finish the two-neighborhood query.

QUERY RESPONSE TIMES - FOR WEAKLY CONNECTED COMPONENTS

A weakly connected component (WCC) is the maximal set of vertices and their connecting edges which can reach one another, if the direction of directed edges is ignored. The WCC query finds all the WCCs in a graph. This query requires that every vertex and every edge be traversed.

We compared TigerGraph to Neo4j. TigerGraph's GSQL query language is flexible enough to let us try several approaches. We implemented WCC two ways. First, WCC-global makes every vertex a starting point and expands from each to find WWCs in parallel; second, WCC-BFS runs a breadth-first search algorithm from an unlabeled vertex, assigns it a new WCC id, and labels all reachable vertices with the same WCC id. WCC-global has better parallelism, while WCC-BFS has lower algorithm complexity. TigerGraph's WCC-BFS was about 2x faster than WCC-global, so only the WCC-BFS times are reported.

For Neo4j, we use the Neo4j native WCC implementation provided in the Neo4j AOPC procedures (release 3.1.3.6). The APOC's implementation of WCC adopts an algorithm similar to WCC-BFS.

To minimize the network overhead time, we designed the queries to return only the number of weakly connected components, instead of reporting the full membership of each component.

Table 8 - Weakly connected component query time

DATASET	TIGERGRAPH-BFS-16	NEO4J
graph500-22	8.7 s	73.0 s
twitter-rv	217.8 s	Out of memory

✓ TigerGraph is 8x faster than Neo4j on the smaller dataset.

✓ Neo4j has a much larger memory footprint and ran out of memory when running on the larger twitter-rv dataset.

1 2
4 8

SCALABILITY OF QUERY RESPONSE TIMES

In our final test, we examine how the Native Parallel Graph architecture of TigerGraph enables the query response time to decrease as the number of machines used increases. For this test, we used a slightly different Amazon EC2 instance type (DB.R3.2xlarge). We used the twitter_rv dataset and the well-known PageRank query (rank every vertex) repeated 10 times. We tested with 1, 2, 4, and 8 machines in parallel. The graph was partitioned into equally sized segments across all the machines used.

Table 9 - Scalability of query response times

NO.OF MACHINES	1	2	3	4
average query time	81.4 s	51.8 s	28.9 s	17.2 s
Speed up	1.0	1.57	2.81	4.73

PageRank is an iterative algorithm which traverses every edge during every iteration. This means there is much communication between the machines, with information being sent from one partition to another. Despite this communication overhead, TigerGraph's NPG architecture still succeeds in achieving a 4.7x speed up with 8 machines.

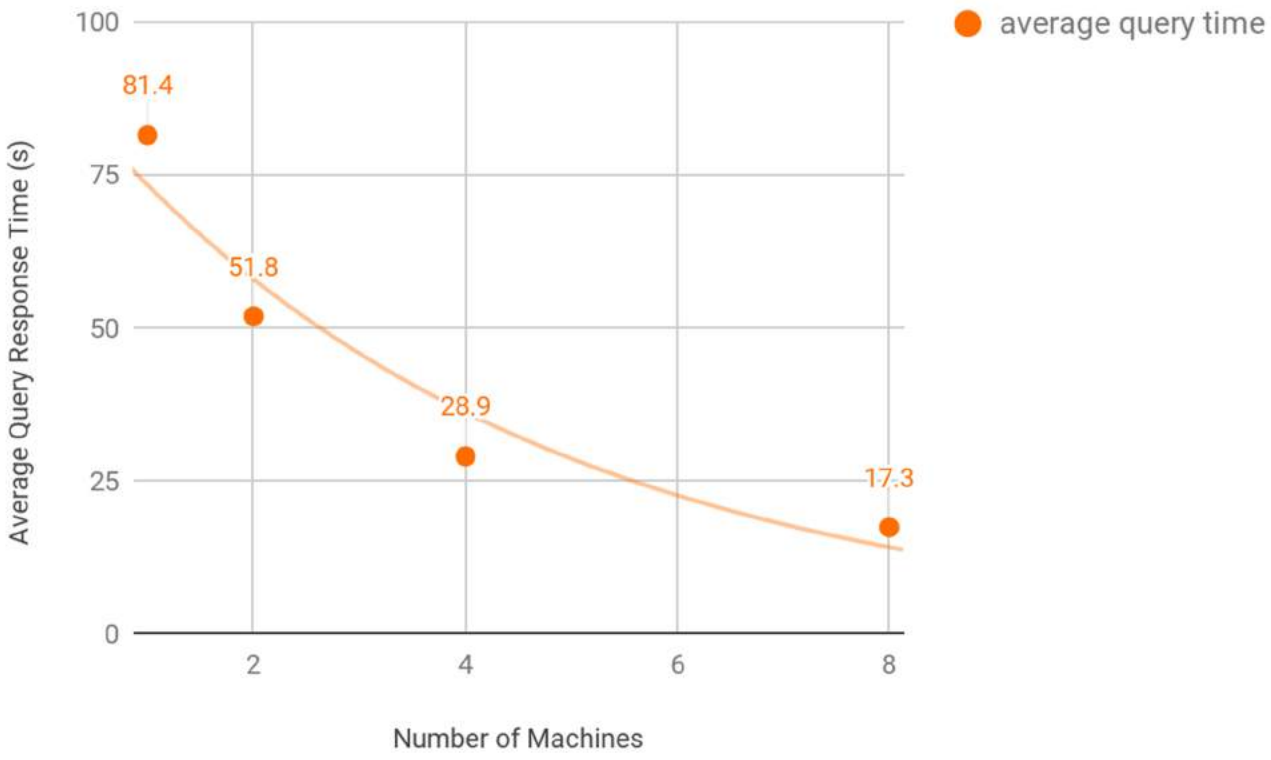

Figure 6 - Query response time vs. Number of machines

Neo4j is a single server and not a distributed system. Thus Neo4j cannot partition a graph across multiple machines and handle big graph data. TigerGraph is the first and only truly distributed native parallel graph system. A graph in TigerGraph can be partitioned and

stored on multiple machines.

QUERY PERFORMANCE SUMMARY

✓ TigerGraph has 4x to 100x faster graph traversal and query response times compared to both Neo4j and Titan.

✓ With TigerGraph, query speed increases as additional machines are added, achieving a 4.7x speed increase with 8 machines.

About TigerGraph

TigerGraph is the world's first Real-Time Graph Analytics Platform powered by Native Parallel Graph (NPG) technology. TigerGraph fulfills the true promise and benefits of the graph platform by supporting real-time deep link analytics for enterprises with complex and colossal amounts of data. TigerGraph's proven technology is used by customers including Alipay, VISA, SoftBank, State Grid Corporation of China, Wish and Elementum.

Founded by Yu Xu, Ph.D. in 2012, TigerGraph is funded by Qiming VC, Baidu, Ant Financial, AME Cloud, Morado Ventures, Zod Nazem, Danhua Capital and DCVC. TigerGraph is based in Redwood City, CA. Learn more at www.tigergraph.com.