Evaluation of contemporary Graph Databases

Aparna Vaikuntam
Samsung R&D Institute India - Bangalore
#2870, Outer Ring Road
918067820000
a.vaikuntam@samsung.com

Vinodh Kumar Perumal
Samsung R&D Institute India - Bangalore
2870, Outer Ring Road
918067820000
vinodh.p@samsung.com

ABSTRACT
Graph databases attempt to make the modelling and processing of highly interconnected data, easier and more efficient by representing a system as a graph-like structure of nodes and edges. The fundamental premise of Graph Databases, unlike relational, is explicit and distinct definition of relationships and direct, non-index based access of related nodes from a given node. Growth of graph databases happened in large part with a need for complex processing of interconnected documents in the WWW and the surge in social networking. What was initially proprietary research has now transformed into a plethora of commercial and open source products, increasingly being adopted outside the internet services industry. This paper attempts to evaluate several such contemporary Graph Databases from a subjective feature-based and empirical performance-based perspective.

Categories and Subject Descriptors
H.2.4 **[Database Management]**: Systems, E.1 **[Data Structures]**: Graphs and Networks

General Terms
Measurement, Documentation, Performance, Experimentation.

Keywords
databases, empirical, evaluation, feature based, graph databases, open source, performance based, subjective, survey

1. INTRODUCTION
The graph data structure has been researched for more than 300 years now. A graph is a set of vertices and edges, the edges being directed or undirected [1]. Additionally, a weightage may be associated with the edges. While this structure is sufficient for algorithmic graph processing, it is inadequate for representing real world entities, which are characterized by properties and may be connected by different types of relationships. The property graph model bridges this gap.

Compute '14, October 09 - 11 2014, Nagpur, India

http://dx.doi.org/10.1145/2675744.2675752

[6] Defines a property graph as follows:
a)A property graph is made up of nodes, relationships, and their properties.
b)Nodes represent properties as <key, value> pairs. The keys are strings and the values are arbitrary data types.
c)Relationships connect nodes. A relationship always has a direction, a label, and a start node and an end node.
d)Like nodes, relationships can also have properties.

More rigorous mathematical definitions can be found in [1] and [8].Graph databases support the property graph model. Nodes and relationships can both have properties and relationships are always associated with a label, thus allowing them to be heterogeneous [1], as is required for real world models.

In the broader technical landscape, Graph databases fall under the loose category of NoSQL (Not only Structured Query language) databases. NoSQL databases introduce flexibility in modeling data. They may not be as rigidly structured through a schema as relational databases and may settle for the BASE (Basic availability, Soft state and eventual consistency) model as opposed to the relational ACID model [14].

The major categories of NoSQL databases are [2]:

a)Wide column stores :- Cassandra and HBase
b)Document stores :- CouchDB and MongoDB
c)Key value stores or Tuple stores: - Amazon Dynamo.
d)Triple stores or RDF stores :-Apache Jena, Allegro Graph
e)Graph databases :- Neo4j, Titan

Triple stores, like Graph Databases, model data as an interconnected structure of nodes and relationships. However, these databases are typically used to model RDF (Resource Description Framework) statements in the subject-object-predicate, or triple form and are optimized for data in that format. For instance, triple stores do not support properties for edges. Several efforts have been made towards generalizing RDF stores to support property graphs and customizing Graph databases to model RDF data. The distinction between these categories is certainly blurring.

The scope of this paper is limited to analysis of open source Graph databases. We choose Neo4j, Aurelius Titan and OrientDB for the purpose of this study.

Of these, OrientDB is a multi-model database that combines the graph model with a document store [53]. Support for the crucial non-index based node access feature (discussed in detail in 3.2), makes it a viable contender for a Graph Database.

This paper is organized as follows: we discuss related work in section II, Section III details the feature based evaluation of databases and we do a performance based evaluation in section IV, followed by conclusions in section V.

2. RELATED WORK AND MOTIVATION

This work builds on several Graph database evaluation efforts. One of the earliest such efforts was by Chad Vicknair et.al [3]. The authors compare Neo4j v 1.0x with the relational MySQL database in disk space, query performance and scalability. The paper discusses the security shortcomings of Neo4j in some detail. [3] Was followed by Marek Ciglan et.al [8] and Renzo Angles [10]. In [8], the authors evaluate graph database traversals in a memory constrained environment and propose a graph database benchmark. They emphasize the importance of non-biased testing with data resembling real world systems as much as possible. In [10], the author performs a subjective analysis of graph databases from a data modelling perspective to help designers evaluate products based on support for their application domain.

Like [3], Shalini Batra et.al in [4] compare graph and relational databases but arrive at a starkly different conclusion. While [3] raises several red flags on the use of Neo4j in production environments, [4] recommends it. Florian Holzschuher et.al in [9], evaluate graph databases in the same lines as [8], but take a query language rather than performance perspective. The most recent work is by Robert McColl et.al [11]. Open source graph databases are analyzed qualitatively (based on user experience, developer experience etc.) and empirically.

In our work, we refer to and build on several of the efforts mentioned above. This paper makes two important contributions. Firstly, we present a rigorous feature based comparison of open source graph databases. Indications of presence or absence of features are accompanied by annotations and references. Secondly, we perform read and write performance evaluations under massive loads. Earlier work like [9] and [11] focuses on traversal query performance, which is not a sufficient indication of performance on insertions or read write patterns. We attempt to bridge these gaps in the current literature with our work.

3. FEATURE BASED EVALUATION

3.1 Procurement

Table 1 has some interesting facts that highlight the differences in procurement of the database products, even in the open source versions. Neo4j's community version is open source, but is not publicly hosted. Contributions are possible only with a copyright sharing agreement. Titan is available with an Apache 2.0 license, but the storage backend specific licenses also apply. BerkeleyDB among them has the more restrictive AGPL license.

3.2 Graph Database Features

Table 2 provides an in-depth analysis of the Graph Databases based on technical features like distributed storage, query processing, indexing, replication and transaction management. We note that the security concerns of [3] regarding Neo4j are still valid. Neo4j, being an embedded database, treats the application as the sole user of the DB and leaves role based access control to the application. Titan too, has no built-in security features. Authentication and HTTPS are available only through Rexter. It is possible to use Neo4j with Rexter in theory, but it has been met with little success by the community. OrientDB is the only Graph Database in our study with built-in security features. Apart from database monitoring, query optimizations etc., OrientDB has made all the listed features available in its community version also. The same cannot be said for Neo4j. The Enterprise version of Neo4j additionally offers distributed query processing [15] and replication [16].

3.3 Graph Database Interfacing

Table 3 analyzes how applications, developers and users can interact with these databases. Neo4j and OrientDB Enterprise versions are not included in this comparison as they do not offer significantly different interfacing features. An exception must be made for monitoring. Neo4j Enterprise [17] and OrientDB [18] enterprise both offer monitoring features. In Neo4j, support comes through JMX. Transaction management, memory mapping, locking, caching, kernel, store file sizes etc. can be monitored.

3.4 Support and Maturity

Table 4, deals with the highly subjective issue of maturity and support levels of the Graph Databases. Neo4j, being a pioneer in the field of Graph Databases, has been in the market longer than Titan and OrientDB. There is more support for Neo4j on developer forums. The mailing group run by Titan, though, has fairly good turnaround time for queries. Note that comments on documentation are based on our experience during this study.

Collectively, for all databases, detailed documentation for important features like sharding across nodes of a distributed system, support for distributed query processing, disk storage format etc. is severely lacking. Though subjective, there can be no doubt about the importance of these criteria in selecting a graph database.

4. PERFORMANCE BASED EVALUATION

In this section, we evaluate the graph databases using a series of experiments that exercise the database with parallel request patterns.

Experimental setup:

a) Neo4j (community v2.0.1), Titan (v0.4.2) and Orient DB (community v1.7-rc1) were evaluated.

b) Performance evaluation was conducted on a Linux, Ubuntu 12.04 LTS machine with 4 GB RAM (entire 4 GB allocated to evaluation process) and 3.10 GHz quad core processor.

c) Single node setup is used for all databases. Performance in case of distributed backend is not tested.

d) Unless otherwise mentioned, all test cases were run 10 times. Of the 10 values obtained, the 2 highest and lowest were eliminated. Therefore, the final figure for each test case is an average of 6 runs.

We tested with the embedded Java API for all Graph databases.

Table 1. Procurement Details

	Neo4j - Community v2.0.1	Neo4j -Enterprise	Titan -v 0.4.2	OrientDB-community-1.7-rc1	OrientDB-enterprise
Open source	Yes. But contribution requires a Contributor license agreement (CLA) [19] and copyright sharing	No	Yes. Code is hosted in Github [20] and project can be forked.	Yes. Code is hosted in Github [21] and a patch can be applied by forking the project, applying changes and issuing pull request.	No
Pricing	Free.	Free for personal use and startups with 3 or fewer employees & <$100,000 revenue. >$10k otherwise.	Free. Commercial subscription on request [22]. Titan + Graph analytics engines(Fulgora, Faunus)	Free	Enterprise edition: 1000 pounds p.a, 1 server (Additional 500 pounds for every extra server)
License	GNU General Public License	Commercial subscription for commercial use. AGPL for open source projects	Apache 2.0 Backend specific licenses are also applicable. BerkeleyDB: AGPLv3 Cassandra & HBase: Apache 2.0	Apache 2.0	Commercial license

Table 2. Graph Database Features

	Neo4j-community	Titan	Orient-DB-community
Property Graph Model [6] Directed, edge-labeled, attributed multi-graphs.	Yes.	Yes	Yes
TinkerPop integration. TinkerPop is an open source Graph DB software stack. It is easier to plug-and-play the backend, if Graph databases integrate with it.	Yes. Not a part of the Neo4j API, but Blueprints has a Neo4j specific implementation. Gremlin can also be used with Neo4j [23].	Yes [24]. Blueprints, Gremlin, Frames and Rexter are supported. Blueprints API is directly implemented without adapters. Rexter can be configured for use with Titan and is available in the titan-server distribution. Gremlin is the only query language supported by Titan.	Yes [25]. Blueprints API is supported and Rexter can be configured for use with Orient DB. Gremlin querying is also supported.
Native Graph Storage & Index free adjacency [6]. Disk storage is optimized for graph processing. O(1) retrieval cost must be possible for adjacent vertices/edges [1] of a node.	Yes. Each node maintains direct pointer access to adjacent nodes. Query time is independent of the total size of the graph, but proportional only to the degree of a node.	Yes. Stores graph in adjacency list format on storage backend like Cassandra and HBase, which support the BigTable format [26].	Yes. Direct pointer based access of connected nodes [27].

Distributed storage Support of sharding across nodes of a cluster for partionability and scalability.	No.	Yes. Data partitioning is supported by both Cassandra and HBase [28].	No.
Distributed query processing. Processing of queries happens in parallel over the machines of a cluster.	No	Yes. Not officially documented in Titan manuals. But data can be distributed by random partitioning and a query can be processed in parallel by several instances in a cluster. This is tested in Cassandra by the Titan DB team [29].	No
Transaction management	Explicit and mandatory transaction blocks for all DB operations. Fully ACID compliant [30].	Implicit transaction block, Commit or rollback is required to close transaction. ACID or eventual consistency depending on chosen backend. HBase and Cassandra are eventually consistent. BerkeleyDB is ACID compliant [31].	Fully ACID compliant. Transaction types supported are: none (no transaction, all operations executed instantly) and optimistic (concurrent reads and writes on a record with integrity check on commit) [32].
Indexing. Although all Graph DBs are schema free, adding indices for properties is supported for faster lookups. E.g. index on name property for faster name based lookups.	Yes. Graph can be configured to automatically index properties. Manual index creation is also possible. Apache Lucene is embedded with Neo4j. A separate spatial index plugin is also available [33].	Yes. Standard and external indexing are supported. Standard (titan) indices match property keys exactly. External indexing backend can provide full text search, numeric range and GIS based capabilities. Currently Elastic search and Apache Lucene are both embedded with Titan [34].	Yes. Full text search is natively supported. Spatial index is not available. External indexing backend are not supported [35].
Security	Yes. User authorization and built in support for SSL with self-signed certificate is provided. For production deployments, a proxy front is recommended. No support exists for multiple users [36].	Yes. HTTPS, simple and custom authentication are supported through Rexter. [37].	Yes. OrientDB server is available. Offers role based and record level security. Rexter can also be used. Multi user support is through Rexter [38].
Bulk Load	Yes. Provides a batch inserter. It is not thread safe, so single threaded or synchronized access is required. Batch insertion is non transactional [39].	Yes. Bulk loading option can be configured. User is required to ensure data consistency. Locking is disabled in this mode [40].	No bulk insert wrapper. 'No transaction' option recommended.
Replication (Availability). Data replication across nodes of a cluster to ensure high availability.	No. Available only in enterprise version.	Yes. Through storage backend Cassandra, HBase [28].	Yes. Through Hazelcast integration [41].

Table 3. Graph Database Interfacing

	Neo4j-Community	Titan	OrientDB-Community
Rest API	Yes. Built in support [42].	Yes. Through Rexter [37].	Yes. Built in support [43].
Native API	Yes. Native API available for Java, Ruby, PHP, and .NET [44].	Yes. Only Java API available, through Blueprints [45].	Yes. Java, JavaScript, PHP, Node.js and Scala API [46].
Query Language	Yes. Neo4j's Cypher is recommended. Gremlin is also supported [47].	Yes. Only Gremlin is supported [48].	Yes. Support for SQL, JavaScript and Gremlin [46].
GUI	Yes. Built-in support for web user interface.	Yes. Through Rexter integration [37].	Yes. Through OrientDB Studio app [49].
Graph Visualization	Yes [50].	Yes. Through Rexter integration [37].	No. Open source tool Gephi is recommended.
Monitoring	No.	Yes [51].	No
Custom Workbench	Yes [52]. Neoclipse (a standalone app) is available.	No	No

Table 4. Support and Maturity

	Neo4j-Community	Neo4j-Enterprise	Titan	OrientDB-Community	OrientDB-Enterprise
Documentation	Yes. Well documented.	Yes. Details of enterprise level features available on website.	Yes. Not very well documented.	Yes. Reasonably well documented.	No. Enterprise version details are not available.
Professional support	No	Yes.	Yes. With paid subscription.	No.	Yes.
Community support	Yes. Very active community.	No. Professional support must be used.	Yes. Google group and stackoverflow support is there, but not as active as Neo4j community. No events or training programs like OrientDB and Neo4j.	Yes. Issue fixing is active. Not well supported on Stackoverflow yet.	No. Professional support must be used.
Maturity	Development active since 2000. Parent company is Neotechnology. First stable release in Feb 2010 (v 1.0).	Info not available.	Parent company Aurelius. First stable release in Sept 2012(v 0.1)	Parent company Orient Technologies, incorporated in 2011. First release, Dec 2013 (v 1.6.3)	Info not available.

4.1 Parallel Writes

Time taken for writes/inserts to the DB (BerkeleyDB for Titan) was tested in parallel, using 200 - 1000 threads. Table 5 (Graph in Fig1) contains results for writing nodes and Table 6 (Graph in Fig2) for relationships. Time is in milliseconds. For node creation, performance of all databases is comparable for small to medium graphs, up to 500k nodes. But Neo4j shows a significant increase in response time for a 1 million node graph. OrientDB, on the other hand, shows a stark increase in response time as more relationships are added. Neo4j persisted less than 100% of relationships as the graph size grew larger. This was due to deadlock exceptions, in spite of each thread adding relationships for nodes in an exclusive id range.

No properties were added for nodes or relationships. Each thread added relationships between nodes selected randomly using ids in an exclusive range. E.g. [0-49], [50-99] etc. All three databases issue id's in an incremental pattern, which was used in random selection.

Nodes are usually accessed based on their properties and an index is provided to speed up the queries. We wanted to measure only write performance, independent of property based access and indices. Hence, the above approach. In both node and relationship creation, threads were synchronized and commit was issued at thread completion. In effect, through multiple threads and a final commit statement in each thread, we tested the performance of the databases under the load of increasing number of users.

Table 5. Parallel Write Performance - Nodes

	10k 200 ×50	100k 200 ×500	200k 400 ×500	500k 500 ×1000	1million 1000 × 1000
Neo4j	7242	8048	16029	23808	197057
Titan	8276	9627	18661	27632	55573
OrientDB	2594	5533	10924	26341	52827

Time taken by each database in milliseconds for parallel writes of nodes. A ×B: A threads in parallel, each creating B nodes. Total of N = A ×B nodes are created.

Table 6. Parallel Write Performance – Relationships

	10k 200 × 50	100k 200 ×500	200k 400 × 500	500k 500 ×1000	1million 1000 ×1000
Neo4j	6824	8205	16936	28844	239718
Titan	10766	13521	23754	35540	72253
OrientDB	800	24980	46371	133245	293941

Time taken by each database in milliseconds for parallel writes of relationships. A ×B: A threads in parallel, each creating B relationships. Total of N = A ×B relationships are created.

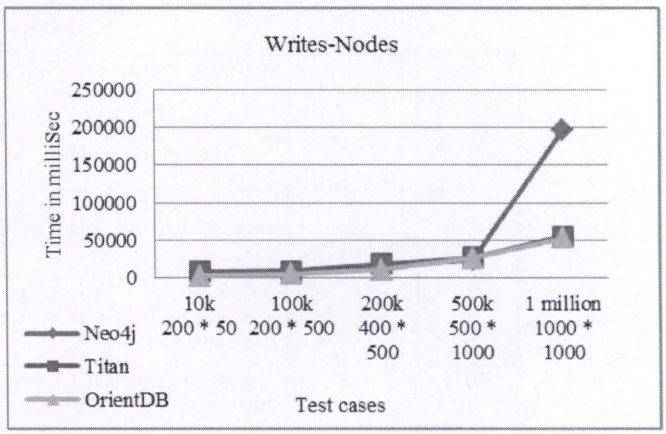

Fig. 1. Parallel insertion of nodes into the databases.

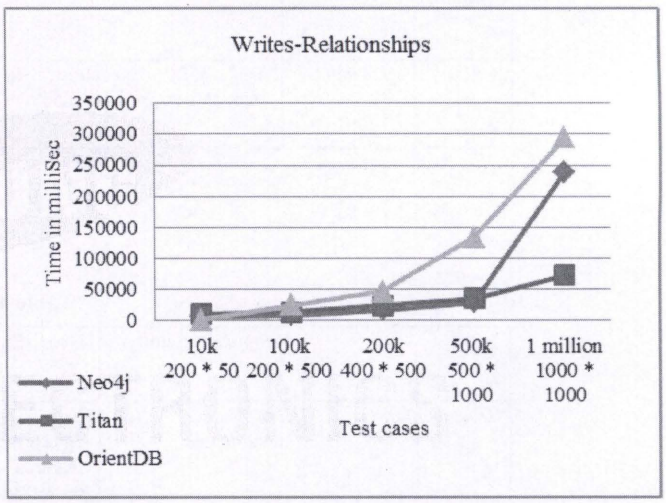

Fig. 2. Parallel insertion of relationships into the databases.

4.2 Parallel Reads

This test evaluates the performance of databases on parallel reads of nodes and their outgoing relationships in a connected graph. Every thread randomly selects nodes based on their ids. A read is a node access and read of all outgoing relationships of the node. Experiment results are in table 7 and graph in Fig. 3.

The reads were conducted in connected graphs, with relationships 1.1 times the number of nodes. Titan shows a sharp increase in response time as the graph size increases beyond 200k. OrientDB shows a more linear curve. Neo4j showed the best performance for parallel reads.

Table 7. Parallel Read Performance

	10k 200 * 50	100k 200 * 500	200k 400 * 500	500k 500 * 1000	1million 1000 * 1000
Neo4j	887	3740	5131	8328	11051
Titan	774	4034	10335	39508	110165
OrientDB	2572	5028	10762	28537	61071

Time taken by each database in milliseconds for parallel reads.
A ×B: A threads in parallel, each reading B nodes and their relationships

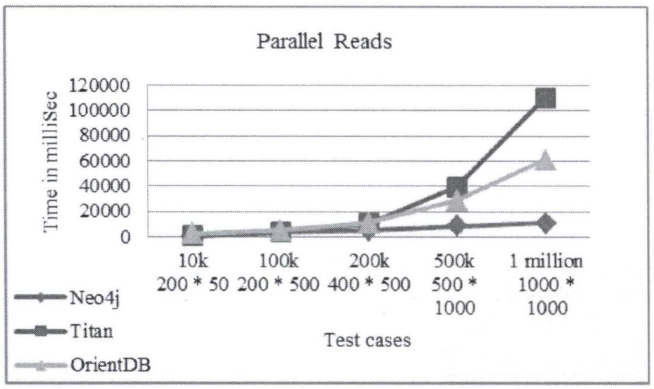

Fig. 3. Parallel reads of nodes and their outgoing relationships

4.3 Interleaved Reads and Writes

This test evaluates the performance of the databases on interleaved reads and writes. For instance, reads followed by writes, followed by reads again. This is also an evaluation of property based access of nodes and relationships. For each database, an integer property was set for every node and relationship created. An index was created on this property for edges and vertices. This index was used to access the nodes and relationships for reads based on the property.
Again, commit is issued at thread completion. Table 8 is a record of the experiment results; Fig 4 provides a graphical view.

The zigzag pattern corresponds to the higher load of WWRR2 and WRWR2 patterns. Neo4j shows a sharp increase in response time in the heavier interleaved load. Neo4j and Titan both could not successfully commit all transactions. There were deadlock conditions in Neo4j and lock timeout exceptions in Titan for property based access of nodes through an index. For this reason, figures for Titan and Neo4j are an average of 3 of 10 runs, where more than 80% of nodes and relationships were persisted. This is to mitigate any unfair latency figures. Only Orient DB completed execution without any issues in each interleaved test case.

4.4 Friend of Friend Query

The friend of a friend query performance was tested for a depth of 5 on a connected graph with 1 million nodes and 1.4 million relationships. Gremlin was used for Neo4j, Titan and OrientDB.

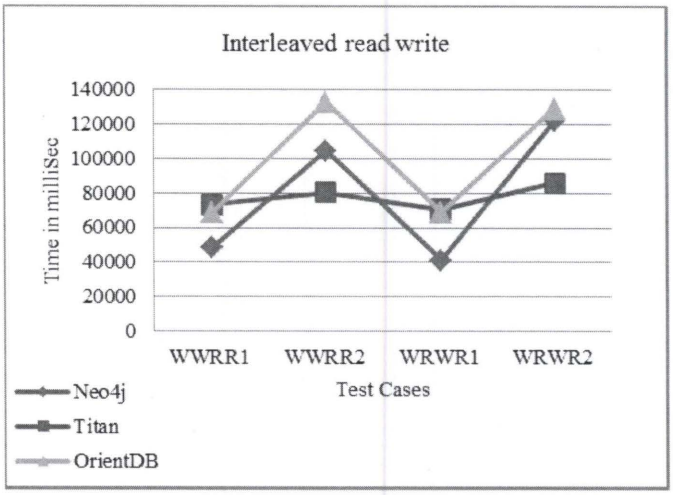

Fig. 4. Interleaved and parallel read and write operations

Using Gremlin for Neo4j ensured consistency of the experiment and factored-out any customization related performance benefits Cypher may have. In all Graph Databases, around 50% of the queries returned results. Each thread performs FoF on a randomly selected node. Experiment results are in Table 9, graph in Fig 5.

Table 9. Friend of Friend Query Performance

	50 threads	100 threads	300 threads	500 threads	1000 threads
Neo4j	76	83	143	261	449
Titan	327	579	1300	2618	11376
OrientDB	161	193	358	483	762

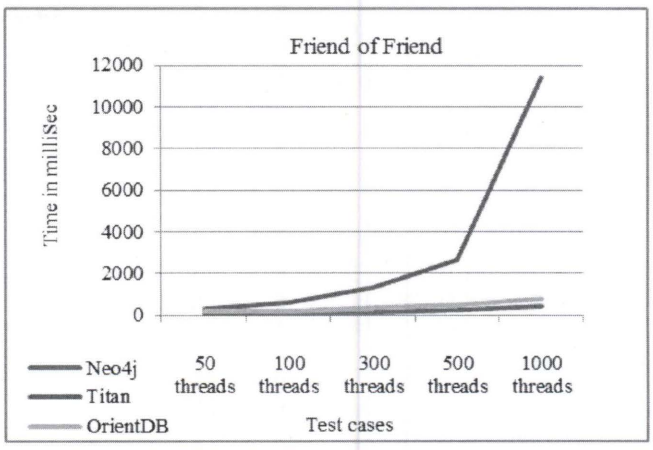

Fig. 5. Parallel friend of friend queries.

Table 8. Interleaved read and write

	WWRR1 200 threads, each: create 500 nodes create 500 relations read 500 nodes read 500 relations	WWRR2 400 threads, each: create 500 nodes create 500 relations read 500 nodes read 500 relations	WRWR1 200 threads, each: create 500 nodes read 500 nodes create 500 relations read 500 relations	WRWR2 400 threads, each: create 500 nodes read 500 nodes create 500 relations read 500 relations
Neo4j	48172	104744	41188	122030
Titan	73262	80566	70590	86065
OrientDB	68308	132951	68940	129320

5. Conclusion

The choice of a graph database is dependent on several factors. There is the application level consideration; whether regular updates are expected or very rare updates and intensive querying. How much scaling is required? Are the graphs going to get so large that there is no alternative to physical sharding? There is the technological aspect. If the organization has a well-established distributed storage system in place, the priority would be to integrate the Graph DB with existing distributed backend. There is also, finally, a mix of several subjective factors like the support and maturity of the product, cost and learning curve for developers etc.

Neo4j shows good parallel read and query performance, but its write performance is inferior to Titan and OrientDB, especially on larger graph sizes. This may not be a consideration for a query intensive application, where the graph is batch loaded and hardy modified after that. For highly parallel and interleaved reads and writes, OrientDB is a good choice. The general professional and community support does seem to be better for Neo4j, but it must be singled out for not supporting a distributed backend in its free community version. Titan and OrientDB must be commended for their much higher open source support. In general, we urge developers faced with a choice to study the implementation details and capabilities of Graph Databases and then exhaustively consider all criteria on which they want to base their decision before choosing any one database. No one Graph Database is uniformly better than the rest in every criteria, feature or performance based.

6. Future Work

The performance evaluation of Graph Databases in this work is done in a single node setup. It would be interesting to evaluate based on the same criteria in a distributed environment. Such a study would help us understand how features like sharding, replication and distributed query processing impact the performance of distributed Graph Databases on writes, reads and traversal queries.

7. References

[1] Marko A. Rodriguez and Peter Neubauer. The Graph Traversal Pattern. *Graph Data Management: Techniques and Applications, 2011, arXiv: 1004.1001 [cs.DS]*

[2] Bogdan George Tudorica and Cristian Bucur. A comparison between several NoSQL databases with comments and notes. In *Roedunet International Conference (RoEduNet)*.Lasi, 2011, Pages 1-5.

[3] Chad Vicknair et.al. A Comparison of a Graph Database and a Relational Database. In *ACMSE '10.* April 15-17, 2010, Oxford, MS, USA

[4] Shalini Batra and Charu Tyagi. Comparative analysis of relational and Graph databases. *C. T, International Journal of Soft Computing and Engineering (IJSCE)*, 2(2).

[5] Renzo Angles and Claudio Gutierrez. Survey of Graph Database Models. *ACM Computing Surveys, Vol. 40, No. 1, Article 1.*

[6] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases.* 1st ed, June 2003 [Online]. Available: http://graphdatabases.com/.

[7] Salim Jouili and Aldemar Reynaga. imGraph: A distributed in-memory graph database. Unpublished.

[8] Marek Ciglan, Alex Averbuch and Ladialav Hluchy, Benchmarking traversal operations over graph databases. *978-0-7695-4748-0/12 © 2012 IEEE.*

[9] Florian Holzschuher and René Peinl. Performance of Graph Query Languages. *ACM 978-1-4503-1599-9/13/03 © 2013.*

[10] Renzo Angles. A Comparison of Current Graph Database Models. *978-0-7695-4748-0/12 © 2012 IEEE.*

[11] Robert McColl et.al. A Performance Evaluation of Open Source Graph Databases. *ACM 978-1-4503-2654-4/14/024.*

[12] Salim Jouili and Valentin Vansteenberghe. An empirical comparison of graph databases. Unpublished.

[13] D. Dominguez-Sal et.al. Survey of graph database performance on the hpc scalable graph analysis benchmark. In *Proceedings of the 2010 international conference on Web-age information management, WAIM'10.* Berlin, Heidelberg: Springer-Verlag, 2010, pp. 37–48.

[14] E. A. Brewer. Towards robust distributed systems. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*. New York, USA, 2000.

[15] Neo4j. Neo4j Scales for the Enterprise. [Online]. http://www.neotechnology.com/neo4j-scales-for-the-enterprise/.

[16] Neo4j. How Neo4j HA Operates. [Online]. http://docs.neo4j.org/chunked/stable/ha-how.html.

[17]Neo4j. Monitoring. [Online]. http://docs.neo4j.org/chunked/stable/operations-monitoring.html.

[18] Orient technologies. OrientDB Enterprise. [Online]. http://www.orientechnologies.com/orientdb-enterprise.

[19] Neo4j. Contributor License Agreement. [Online]. http://docs.neo4j.org/chunked/stable/cla.html#_summary.

[20] Aurelius Titan. Github Code Base. [Online]. https://github.com/thinkaurelius/titan/wiki/Release-Notes.

[21] Orient Technologies. Github Code Base. [Online]. https://github.com/orientechnologies/orientdb/wiki/Contribute-to-OrientDB.

[22] Aurelius Titan. Aurelius Subscription. [Online]. http://thinkaurelius.com/services/subscription/.

[23] Neo4j. Tinkerpop Blueprints Neo4j implementation. [Online] https://github.com/tinkerpop/blueprints/wiki/Neo4j-Implementation.

[24] Aurelius Titan. Tinkerpop Integration.[Online]. https://github.com/thinkaurelius/titan.

[25] Orient Technologies. Tinkerpop Integration. [Online] https://github.com/orientechnologies/orientdb/wiki/Graph-Database-Tinkerpop.

[26] Aurelius Titan. Titan Data Model. [Online]. https://github.com/thinkaurelius/titan/wiki/Titan-Data-Model.

[27] Orient Technologies. Concepts. [Online]. https://github.com/orientechnologies/orientdb/wiki/Concepts.

[28] Aurelius Titan. Storage Backed Overview. [Online]. https://github.com/thinkaurelius/titan/wiki/Storage-Backend-Overview.

[29] Aurelius Titan. Cassandra with Titan. [Online]. https://groups.google.com/forum/#!msg/gremlin-users/bs_zZIQ6b7U/kjR3kr1MeIIJ

[30] Neo4j. Transaction Management. [Online] http://docs.neo4j.org/chunked/stable/transactions.html.

[31] Aurelius Titan. Transaction Handing. [Online]. https://github.com/thinkaurelius/titan/wiki/Transaction-Handling.

[32] Orient Technologies. Transactions. [Online]. https://github.com/orientechnologies/orientdb/wiki/Transactions.

[33] Neo4j. Automatic Indexing. [Online] http://docs.neo4j.org/chunked/stable/auto-indexing.html.

[34] Aurelius Titan. Indexing Backend Overview. [Online]. https://github.com/thinkaurelius/titan/wiki/Indexing-Backend-Overview.

[35] Orient Technologies. Indexes. [Online]. https://github.com/orientechnologies/orientdb/wiki/Indexes.

[36] Neo4j. Securing access to neo4j server. [Online] http://docs.neo4j.org/chunked/stable/security-server.html.

[37] Tinkerpop. Rexter. [Online]. https://github.com/tinkerpop/rexster/wiki.

[38] Orient Technologies. Security. [Online]. https://github.com/orientechnologies/orientdb/wiki/Security.

[39] Neo4j. Batch Insertion. [Online] http://docs.neo4j.org/chunked/stable/batchinsert.html.

[40] Aurelius Titan. Bulk Loading. [Online]. https://github.com/thinkaurelius/titan/wiki/Bulk-Loading.

[41] Orient Technologies. Distributed Architecture. [Online]. https://github.com/orientechnologies/orientdb/wiki/Distributed-Architecture.

[42] Neo4j. REST API. [Online] http://docs.neo4j.org/chunked/milestone/rest-api.html.

[43] Orient Technologies. OrientDB REST. [Online]. https://github.com/orientechnologies/orientdb/wiki/OrientDB-REST.

[44] Neo4j. Contributed Software. [Online] http://neo4j.com/contrib/.

[45] Aurelius Titan. Blueprints. [Online]. https://github.com/thinkaurelius/titan/wiki/Blueprints-Interface.

[46] Orient Technologies. OrientDB Documentation. [Online]. https://github.com/orientechnologies/orientdb/wiki.

[47] Neo4j. Learn Cypher. [Online] http://www.neo4j.org/learn/cypher.

[48] Aurelius Titan. Gremlin Query Language. [Online] https://github.com/thinkaurelius/titan/wiki/Gremlin-Query-Language.

[49] Orient Technologies. OrientDB Studio. [Online]. https://github.com/orientechnologies/orientdb-studio.

[50] Neo4j. Graph Visualization. [Online] http://www.neo4j.org/develop/visualize.

[51] Aurelius Titan. Performance and Monitoring. [Online]. https://github.com/thinkaurelius/titan/wiki/Titan-Performance-and-Monitoring.

[52] Neo4j. Neoclipse. [Online] http://www.neo4j.org/develop/tools/neoclipse.

[53] Orient Technologies. Multi model database. [Online]. http://www.orientechnologies.com/why-orientdb/.