ORACLE
BIG DATA

# Oracle Big Data Spatial and Graph
# Property Graph: Features and Performance

ORACLE TECHNICAL WHITEPAPER | DECEMBER 2017

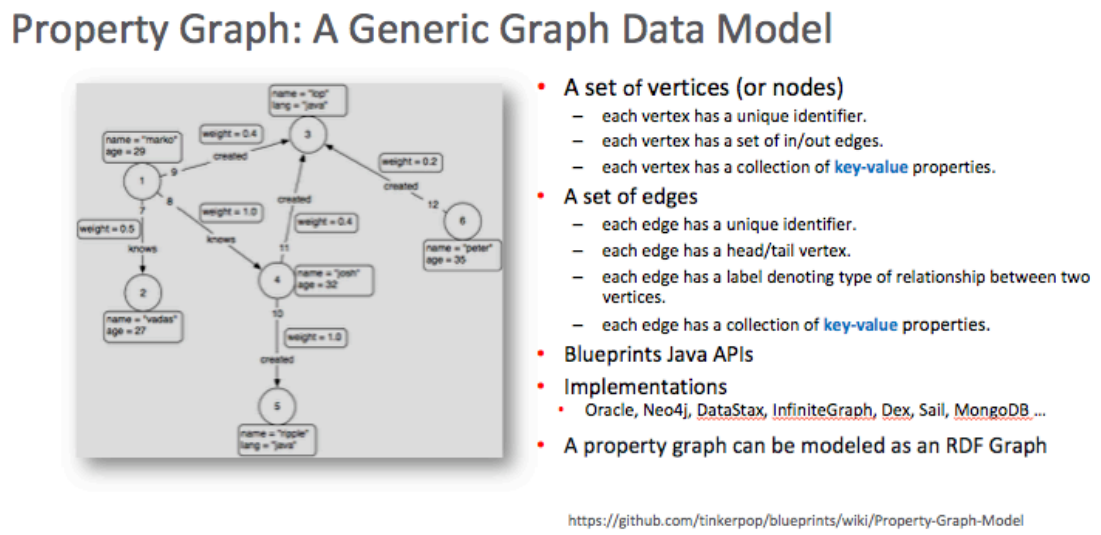
ORACLE

**Table of Contents**

## Introduction

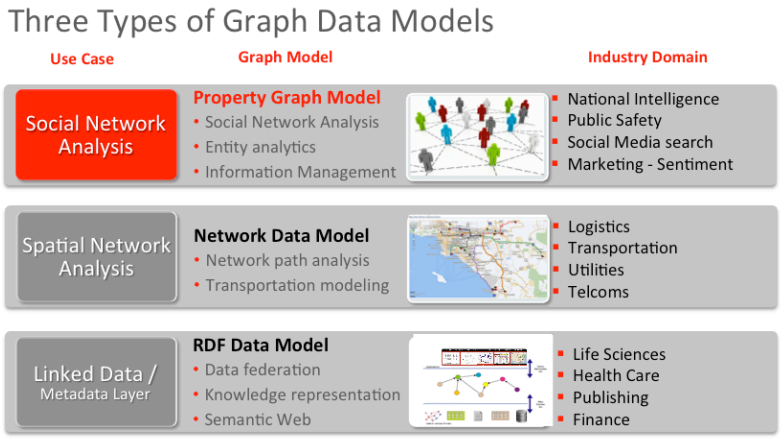
Much of the Big Data generated these days contains inherent relationships between the collected data objects. For example, important relationships and patterns are found in social network data from Facebook, a listener's music preferences from an online music service like Spotify, online shopper behavior on eBay, and bloggers and their relationship to followers and other bloggers. These relationships can be readily structured as a graph – a set of linked objects that represent these relationships.

Property graphs can be used to model these linked object relationships using data structures called

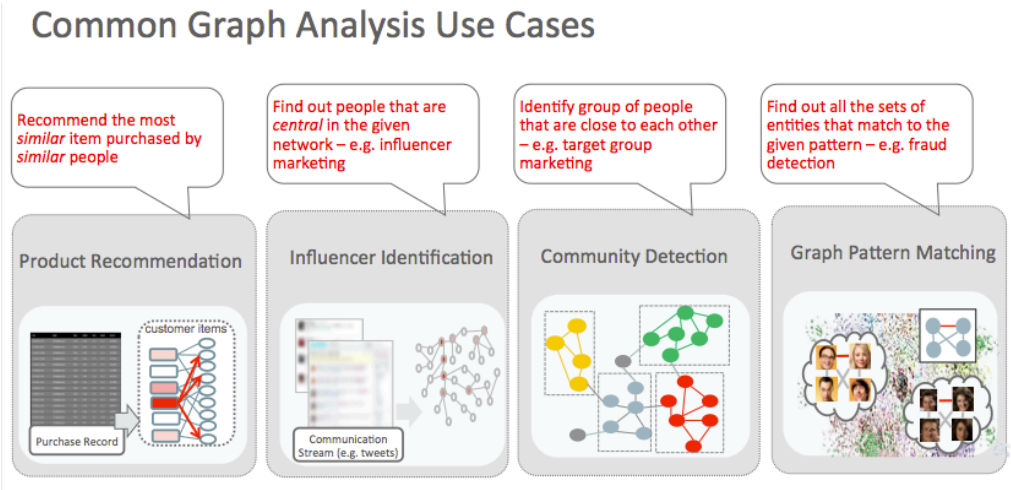



vertices and edges that have associated properties or attributes. These data structures store entities, relationships and their attributes, respectively.

Some graphs (such as roads, telecommunications, water, and utility networks) have an inherent spatial component. Other types of graphs (such as relationships among entities in the Internet of Things, and within biological pathways and social networks) may not.




Graphs are easy to represent, and readily intuitive to visualize. More importantly, it's possible to apply a variety of analytic processes to better understand underlying relationships and connections in large graphs. The recent interest in graph databases is due to the rich insight provided by various forms of graph analysis. Property graphs excel at answering social network questions about communities, influencers, recommendations, and patterns within the network of relationships.




» **Recommendations** can be derived in multiple ways, such as from a personal profile of preferences and an item's description (content-based filtering), or from patterns of past behavior, activities and preferences, and similarity to other users (collaborative filtering). Graphs store relationships as first class entities with vertices and properties, facilitating this kind of analysis.

» **Influencers** in a social network can play an important role in overall information flow by bridging different groups of people. Graph analytics can identify influencers, for instance for a marketing campaign.

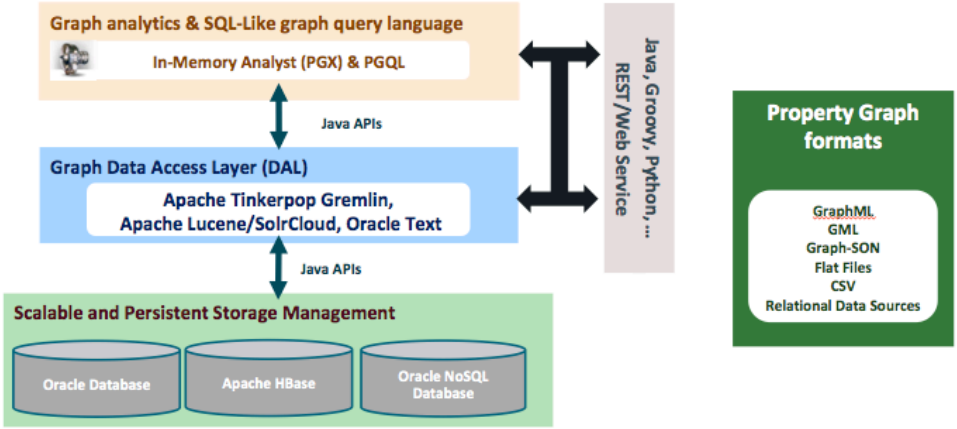
» **Community** can be a social group that interacts on the basis of shared behaviors, likes and interests or on the basis of affiliation or membership in an organization. The strength of a community can be described in terms of the proportion of actual to maximum possible links among the members or the average number of relationships each member has. Finding communities can be a step in further analyses, such as making product recommendations.

» **Pattern matching**, such as finding anomalies like fraud can be challenging in large data sets. However, when the data set is represented as a graph, certain abnormal subsets can become discernable as distinguishing patterns of linkages.

**Oracle Big Data Spatial and Graph** and **Oracle Spatial and Graph** for **Oracle Database** provide a comprehensive, massively scalable, fast, and flexible graph database with advanced property graph analytic capabilities. They provide a secure, distributed property graph that includes an in-memory analyst (PGX) with over forty built-in, high-performance analytics and a SQL-like graph query language called PGQL. They include text indexing for fast entity and relationship searches. Parallelism enhances the performance of property graph database operations, and in-memory analytics.

**Oracle Big Data Spatial and Graph** supports the Apache Hadoop and Oracle NoSQL Big Data platforms, Apache HBase and Oracle NoSQL Database, respectively running on the Oracle Big Data Appliance engineered system and on other Hadoop and Oracle NoSQL Database clusters.

**Oracle Spatial and Graph** brings advanced features for management and analysis of property graphs, spatial data, and RDF linked data applications to Oracle Database. A technical overview is provided in the Oracle whitepaper: *Spatial and Graph Analytics with Oracle Database 18c*. Oracle Spatial and Graph is included with Oracle Database Cloud Service High Performance and Extreme Performance packages, and Oracle Exadata Cloud Service and Exadata Cloud at Customer Extreme Performance packages. It is also an option for Oracle Database Enterprise Edition on-premise licenses.



Property graph features include:

- An in-memory analyst (PGX) with over forty built-in parallelized analytics. It can be deployed either as part of a Java application, scheduled via Hadoop YARN or deployed in shared fashion with a Web application server, Oracle WebLogic Server or Apache Tomcat.
- A SQL-like graph pattern-matching query language called PGQL.
- Ease of development with Java APIs based upon the Tinkerpop stack, and a Groovy shell.
- Support for Apache Spark allows running Spark jobs against graph data stored in HBase and reading data from Spark DataFrame into the in-memory analyst (PGX)
- A Zeppelin Notebook interpreter
- Distributed text indexing that can be automatic or selective (manual) and text search for graph elements using Apache Lucene and SolrCloud.
- A secure, distributed graph database. Property graph support works with Kerberos authentication to secure graphs in Apache HBase, and with both secure (recommended) and non-secure Oracle NoSQL Database installations.
- Spatial filtering to enhance graph analysis
- Fast parallel loading and export that supports an optimized Oracle-defined flat file format as well as the public formats: GraphML, GML, and GraphSON.
- SQL-based analytics with Oracle Advanced Analytics that can be used to access the property graph from Oracle Database using Oracle Big Data SQL.
- Support for graph visualization, including Tom Sawyer Perspectives and the open source Cytoscape graph visualization tool.

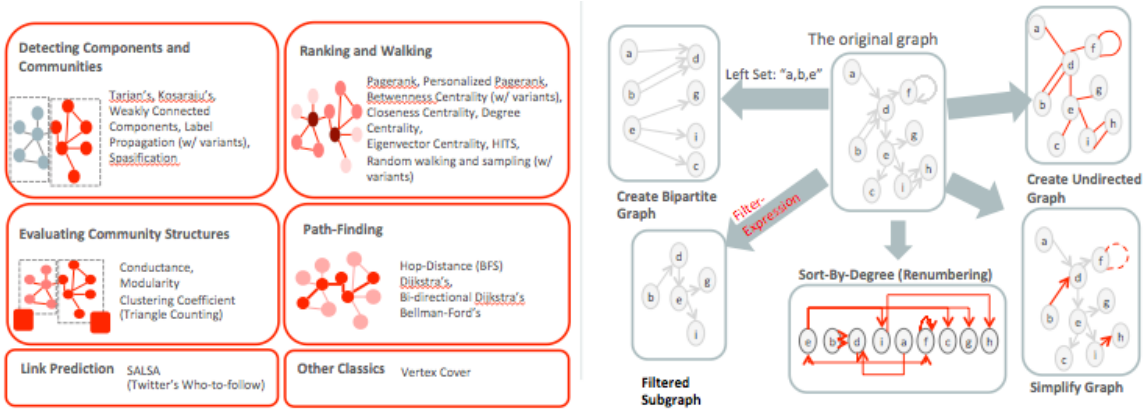
Oracle is committed to open source technology to offer choice, flexibility, and lower cost of computing to users. By investing significant resources in developing, testing, optimizing, and supporting open source technologies for Oracle Big Data Spatial and Graph, such as Apache Hadoop, Lucene, SolrCloud, Tomcat and Groovy; Tinkerpop Blueprints, Pipes, and Gremlin; Eclipse Jetty; Berkeley DB underpinning Oracle NoSQL Database; and open scripting languages, including Python Software Foundation, Oracle is embracing and offering leading open source solutions as a viable choice for development and deployment.

The Feature Overview that follows is organized into three sections. The first section describes the major capabilities of the property graph feature of Oracle Big Data Spatial and Graph. The second section discusses performance, including a comparison to other graph analytics packages, Apache Spark GraphX and Dato GraphLab Create. Appendix A summarizes the built-in property graph analytics in Big Data Spatial and Graph.

## Oracle Big Data Spatial and Graph Property Graph Features

### In-memory Analyst (PGX)

The property graph feature has an in-memory analyst (PGX) with over 40 built-in, parallelized analytics that provide fast graph analysis, as the performance section of this paper describes. Appendix A provides a list of the supported analytics. The in-memory analyst (PGX) also supports operations to derive graphs that are a subset or otherwise modified from the original graph, also referred to as graph mutation. This is illustrated in the figure below.




Built-in analytics and derived graphs

The in-memory analyst (PGX) takes advantage of modern server architecture that parallelizes computation using multiple cores and sizeable memory configurations that enable fast non-sequential data access across a larger portion of a graph read into memory.

Parallelism is also supported when reading graphs into memory for analysis from Oracle NoSQL Database and Apache HBase. It can read a whole graph into memory from the database or part of a graph using an optional filter query. Reading from flat files is also supported. Concurrent sessions are supported with multiple users executing analytics on the same or different graphs.

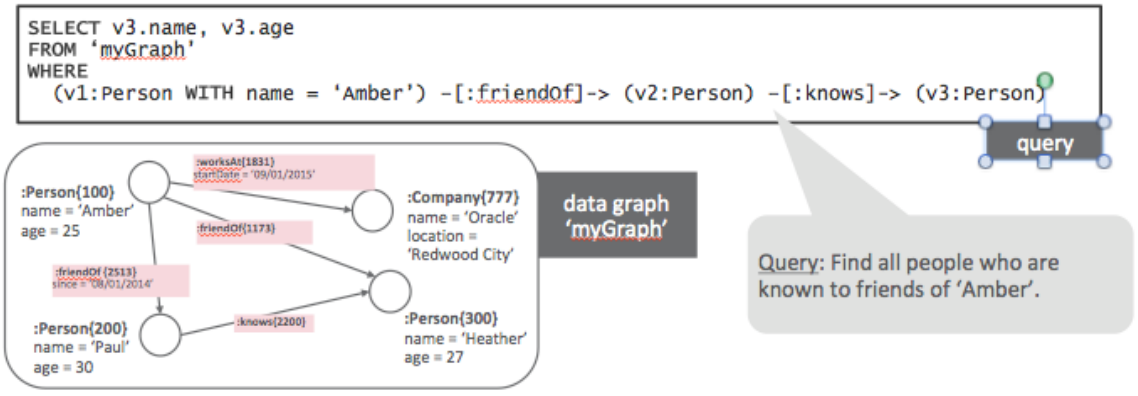
The in-memory analyst can be conveniently deployed depending on application requirements.

- **The Embedded approach**: This method embeds the analytics within the user's Java application. The analytics will execute the application's memory space. This is an efficient single user approach if sufficient memory is available.
- **The Remote approach**: If multiple sessions will be invoking the in-memory analyst, it may be more efficient to deploy it once in a Web container. The supported containers are Oracle WebLogic Server, Apache Tomcat, and Eclipse Jetty.
- **The Batch approach**: Finally, the analytics can be scheduled with Apache Hadoop YARN. YARN can find the necessary CPU and memory resources within the Hadoop cluster to deploy the in-memory analyst.

### Property Graph Query Language

A property graph can be queried using the declarative SQL-like PGQL graph pattern-matching query language. A PGQL query describes a graph pattern with vertices, edges, properties, and their relationships, When the query is evaluated against a property graph, the query engine finds all subgraph instances of the graph that match the specified query pattern. Then the query engine returns the selected data entities from each of the matched subgraph instances.

The following example finds all instances of a given pattern or template in the data graph




PGQL Query

PGQL includes support for grouping (GROUP BY), aggregation (e.g. MIN, MAX, AVG), sorting (ORDER BY) and many other familiar SQL constructs.

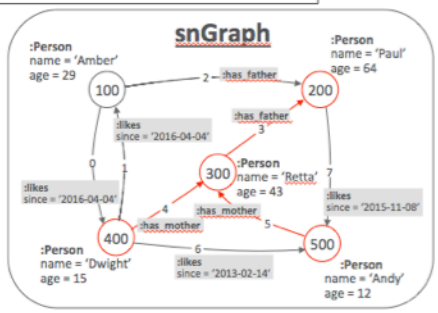
PGQL also supports regular path queries (recursion) for applications such as reachability analysis. A PGQL regular path query is simpler and more concise than a comparable SQL query would be.

The following example matches a pattern repeatedly by defining a PATH pattern at the top of a query, referring to it in the WHERE clause, and using the star symbol (*) for repeated matching.

```
PATH has_parent := (child) -[:has_father|has_mother]-> (parent)
SELECT x.id(), y.id(), ancestor.id()
WHERE
  (x:Person WITH name = 'Andy') -/:has_parent*/-> (ancestor),
  (y) -/:has_parent*/-> (ancestor),
  x != ancestor AND y != ancestor AND x != y
```

Result set

| x.id() | y.id() | ancestor.id() |
|---|---|---|
| 500 | 300 | 200 |
| 500 | 400 | 200 |
| 500 | 400 | 300 |




A regular path query

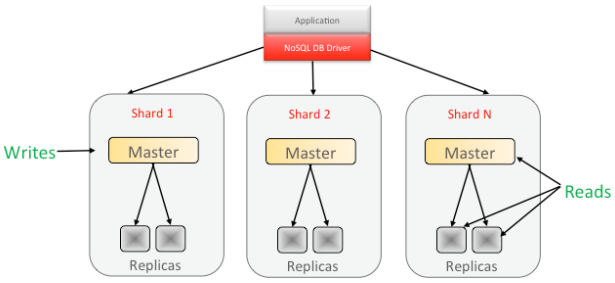
### Querying using programmatic APIs

The property graph can also be queried using Java APIs that implement Apache Tinkerpop Gremlin APIs. The Java APIs perform parallel scans on vertices and edges. Parallel retrieval takes advantage of the distribution of the data across shards and partitions.

## Optimized Schema for Oracle NoSQL Database and Apache HBase

### Oracle NoSQL Database

Oracle NoSQL Database is a distributed, highly scalable key-value store based on Berkley DB. It has ACID transactions, and supports tabular data models and secondary indexes that enable B-tree indexes on columns. It has application security with authentication and session-level SSL encryption. It supports storage nodes, flexible replication nodes and groups configurations, and elastic shards that can be split, added, and contracted for fault tolerance and performance as noted in the figure below.

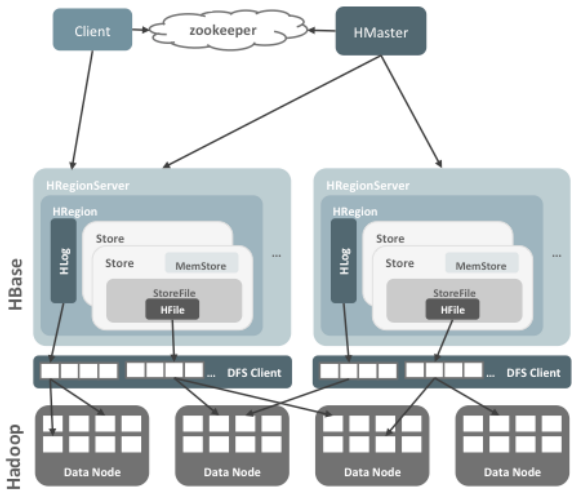



Oracle Big Data Spatial and Graph optimizes graph storage in Oracle NoSQL Database by creating three tables to store property graph vertices, edges and index metadata. It creates secondary indexes to speed up graph traversal. It uses the major features of Oracle NoSQL database, including parallel scan and customized scan, which scans at a finer level of granularity than the default parallel scan.

### Apache HBase

Apache HBase is a component in Apache Hadoop installations. It is a distributed column-oriented key-value data store built on top of HDFS and is based on Google's Bigtable model. HBase files, internally stored in HDFS, automatically organize data in Regions that correspond to a subset of a table's rows, like horizontal range partitioning. Data regions are handled by Region Servers that serve data for reads and writes using a log. A Master is responsible for coordinating the Region Servers, it assigns regions, detects failures, and more.

Apache HBase Database Architecture




Oracle Big Data Spatial and Graph optimizes graph storage in HBase by creating two tables in for vertices and edges. It uses a type of hashing called "salting" to distribute rows across the regions. Since HBase doesn't have

secondary indexes, Oracle Big Data Spatial and Graph maintains its own secondary indices and text index metadata to speed up key-value lookups.

## Text Indexing and Search with Apache Lucene and SolrCloud

### Apache Lucene

Text searching and indexing is enabled through Apache Lucene integration with Oracle Big Data Spatial and Graph that enables queries on property graph elements. For example, if you have Twitter feeds you can query for tweets containing the word “Oracle” using usual search syntax. Both manual (selective) and auto indexing of graph elements are supported.

Auto index is easy to use. It is a b-tree index that is created by specifying the index name and what to index, vertices or edges. It is automatically updated as graph elements are changed.

```
oraclePropertyGraph.createKeyIndex(“name”, Vertex.class);
oraclePropertyGraph.getVertices(“name”, “*Obama*”, true);
```

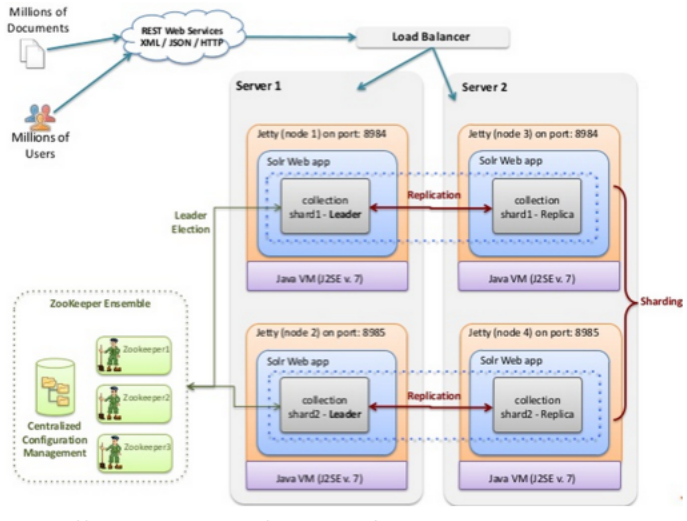
Manual (selective) index is a more flexible manual process. Index content is determined and loaded by the developer. This provides the flexibility to choose what graph elements will be indexed, making the index more focused.

```
oraclePropertyGraph.createIndex(“my_index", Vertex.class);
indexVertices = oraclePropertyGraph.getIndex(“my_index” ,  Vertex.class);
indexVertices.put(“name”, “Barack Obama”,  v);
```

### Apache Solr Cloud

A limitation of Apache Lucene is that its indexes can’t be shared directly among users and applications.

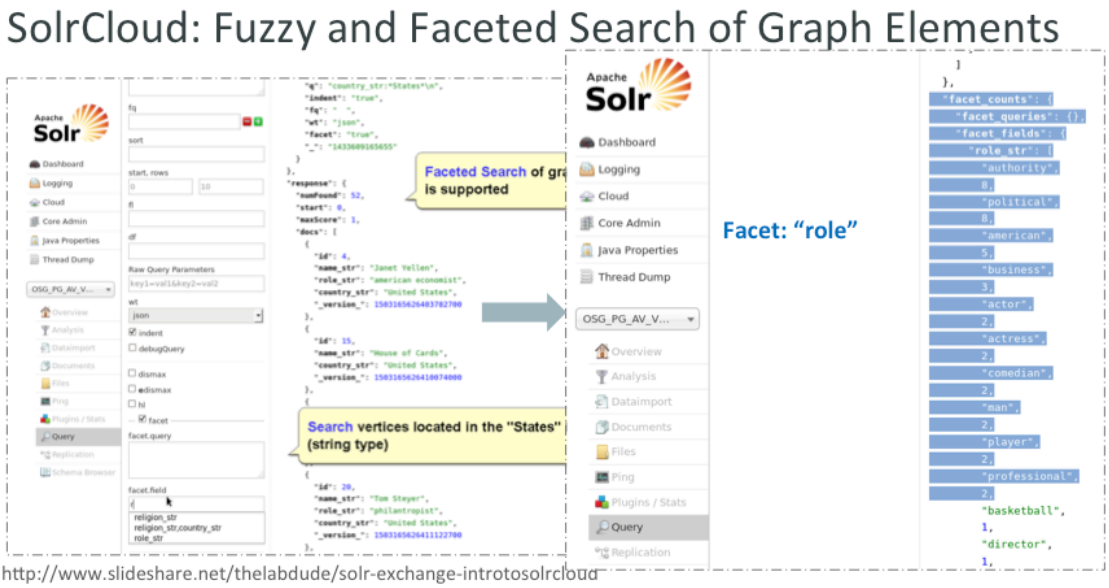
Multiple users and applications can share a SolrCloud index. SolrCloud in implemented with Lucene. A SolrCloud can be created across CDH Hadoop cluster.




http://www.slideshare.net/thelabdude/solr-exchange-introtosolrcloud

The SolrCloud load balancer enables multiple shards on different servers with index replication. Documents are sent to the HTTP interface, sharded, and the index is replicated if that is enabled. The query coordinator transparently queries the shards and assembles the final results.

Faceted Search is a powerful capability of SolrCloud. The left potion of the example below shows a query for vertices that are located in a place with a name containing "States" with "role" defined as a facet field. "Role" is one of the keys used in K/V pairs storing the properties of the vertices. The right side of the example lists the query results by this facet with eight matches that have "political" role, and two matches with "comedian" role.




## Open and Optimized File Formats

The property graph feature supports the serialization formats, GML, GraphML, and GraphSON. Recognizing the limitations of these formats, Big Data Spatial and Graph introduced an Oracle flat file format with better scaling, and more flexibility and data type support. The supported data types are String, Integer, Float, Double, Date (with time zone), Boolean and Java serializable object. The flat file format is uniquely flexible in that it allows a key to be associated with multiple data types. A key is part of a key-value pair that describes a property for a vertex or edge. The example below shows the key_name, "likes" is associated with the Date and String data types.

The flat file format is UTF8 based and stores data in two files, a vertex file with an ".opv" extension and an edge file with an ".ope" extension. Each line in the vertex and edge file is a record that describes a key-value property for particular vertex or edge. Vertices and edges can have multiple properties. The example below has four properties for vertex 1 and one for vertex 2. The fields are the vertex_ID, key_name (property name), data type, and key value in one of the last three fields, depending on the data type. Data type is encoded as 1 for string, 2 for integer, 5 for date, etc.

```
1,name,1,Barack%20Obama,,
1,age,2,,53,
1,likes,5,,,2009-01-20T00:00:00.000-05:00
1,occupation,1,44th%20president%20of%20United%20States%20of%20America,,
2,likes,1,basketball,,
```

## Parallel Load

The parallel data loader provides a simple Java API. Parallelism is achieved with Splitter threads that split vertex and edge files into multiple chunks. Multiple loader threads each load a chunk into the database using a separate database connection. Java pipes are used to connect Splitter and Loader threads, Splitter: PipedOutputStream and Loader: PipedInputStream.

This example loads graph data stored in vertices and an edges files in the optimized Oracle flat file format, and executes the load with forty-eight degrees of parallelism.

```
OraclePropertyGraphDataLoader opgdl = OraclePropertyGraphDataLoader.getInstance();
vfile = "/home/alwu/pg-bda-nosql/demo/connections.opv";
efile = "/home/alwu/pg-bda-nosql/demo/connections.ope";
opgdl.loadData(opg, vfile, efile, 48);
```

## Security

Oracle recommends using secure storage. The property graph feature supports database secure storage; it works with Kerberos authentication to secure graphs in Apache HBase, and with both secure and non-secure Oracle NoSQL Database installations.

Apache HBase can be secured in three steps:

- Modfy `hbase-site.xml` to use kerberos (hbase.security.authentication),
- Configure Apache HBase kerberos principals, and
- Configure Apache HBase servers and clients to authenticate with a secure ZooKeeper. For example:

```
Djava.security.auth.login.config=/etc/hbase/conf/zk-jaas.conf
```

Details can be found in the CDH documentation under "HBase Authentication."

Storing graphs in a secure Oracle NoSQL Database requires configuring a password store and creating a login properties file. For example:

```
Doracle.kv.security=/your_path_here/login_properties.txt
```

Details can be found in the Oracle NoSQL Database documentation, "Getting Started with Oracle NoSQL Database Tables" under "Using the Authentication APIs."

## Groovy Shell

The property graph feature includes a built-in Groovy shell for graph database access and in-memory analyst operations. With this command-line shell interface, you can explore the feature's Java APIs. Groovy scripts allow developers to test Java code snippets more easily without defining objects or compilation. The scripts `gremlin-opg-nosql.sh` and `gremlin-opg-hbase.sh`, are included for connecting to an Oracle NoSQL Database and an Apache HBase, respectively. The documentation "Oracle Big Data Spatial and Graph User's Guide and Reference" includes substantial scripting examples.

## Oracle Advanced Analytics with Big Data SQL, and Oracle Database

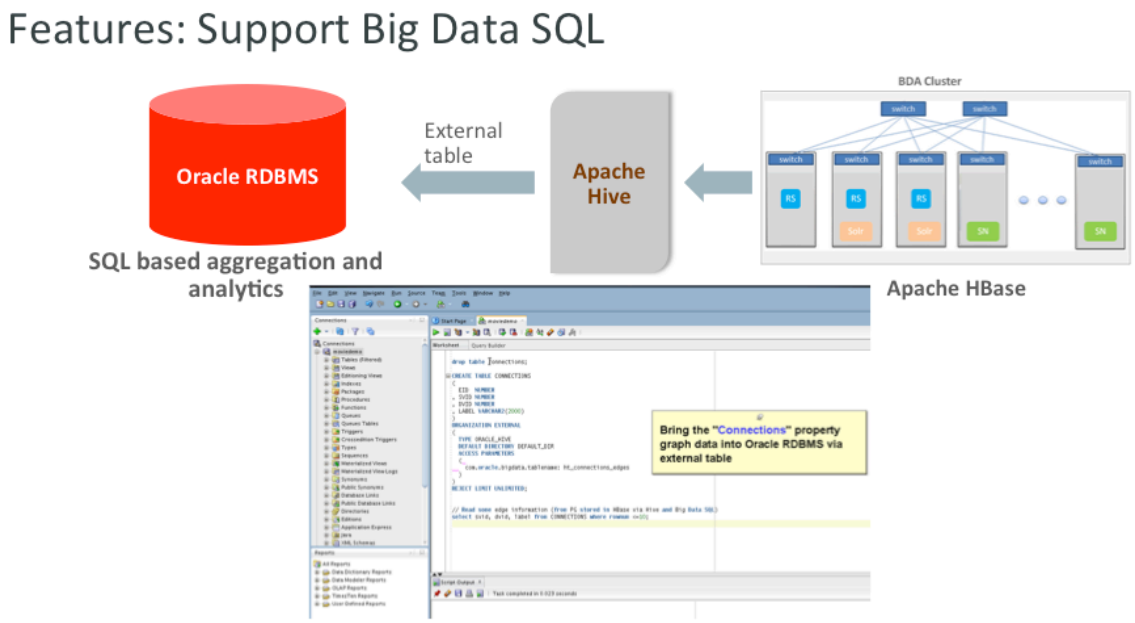
A relational view can be created in Oracle Database using Oracle Big Data SQL on property graph data stored in Oracle NoSQL Database (Release 12.1.3.5.2 or higher) or Apache HBase. The relational view can be used to apply Oracle SQL functions, aggregations, etc. to property graph data and/or to present the property graph data to relational tools for analysis, such as Oracle Advanced Analytics for data mining and R statistical analysis.

Establish a relational view in Oracle Database on the property graph data by:

» Defining a table in Oracle NoSQL Database or a Hive external table in Apache Hbase on the property graph edge table

» Defining an external table in Oracle Database using Oracle Big Data SQL to map to the NoSQL or Hive table.

An example using Hive and HBase is shown below.



# Performance Study

## Performance of Property Graph Operations and In-Memory Analytics

The property graph feature of Oracle Big Data Spatial and Graph was benchmarked on Oracle NoSQL Database and Apache HBase using the data sets Live Journal (69M edges), Twitter (1,5B edges) and YahooWeb (2.9B edges).

**The major findings**: Big Data Spatial and Graph makes use of parallelism to provide fast, scalable loading, scanning, text indexing and search, in-memory analytics, and sub-millisecond execution of graph operations.

### The Tested Environments

For NoSQL loading, scanning, and in-memory analyst tests

- 8-Node BDA Cluster - Oracle NoSQL Database Enterprise Edition 12.1.3.2.5
- 8 Storage Nodes
- 12 Replication Nodes/Storage Node
- 8.5GB Heap Size/Replication Node, Replication Factor 1
- 2 Clients: Heap Size typically 18-24GB for data access, maximum 48GB (not a dynamic setting)

For NoSQL graph operations tests

- 9-Node Cluster - Oracle NoSQL Database Enterprise Edition 12.1.3.2.5
- 9 Storage Nodes
- 4 Replication Nodes/Storage Node
- 31GB Heap Size/Replication Node, Replication Factor 1
- 1 Client: Heap Size 48GB maximum

For HBase tests

A 4-Node BDA Cluster - Cloudera CDH 5.4 w/ Apache HBase 1.0.0

- 3 HBase RegionServers + 1 node serving as the client
- 48 GB RAM per node, 24 Processors/Node
- 31 GB Heap Size

A 9-Node BDA Cluster - Cloudera CDH 5.4 w/ Apache HBase 1.0.0

- 8 HBase RegionServers + 1 node serving as the client
- 126 GB RAM per node, 72 Processors/Node
- 31 GB Heap Size

### The Tested Data Sets

LiveJournal (LiveJ) social network

- LiveJournal is an on-line community w/ near 10 million members
- Members maintain journals, individual and group blogs
- 68.9m edges (2.9 GB)
- 4.8m vertices (123.8 MB)
- http://snap.stanford.edu/data/soc-LiveJournal1.html

Twitter data about follower relationships

- 1.5B edges (68.9 GB)
- Vertices (1.1 GB)

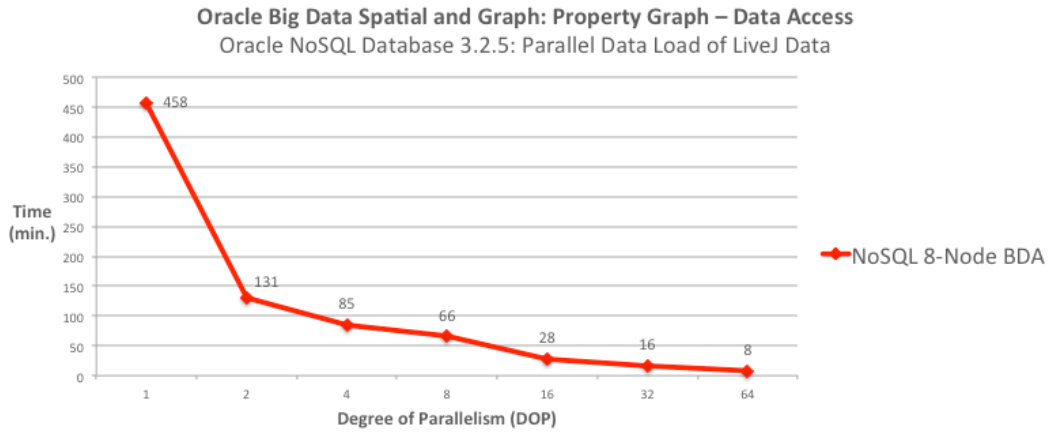
YahooWeb data about pages and their links

- 2.9B edges (140.3 GB)
- Vertices (2.1 GB)

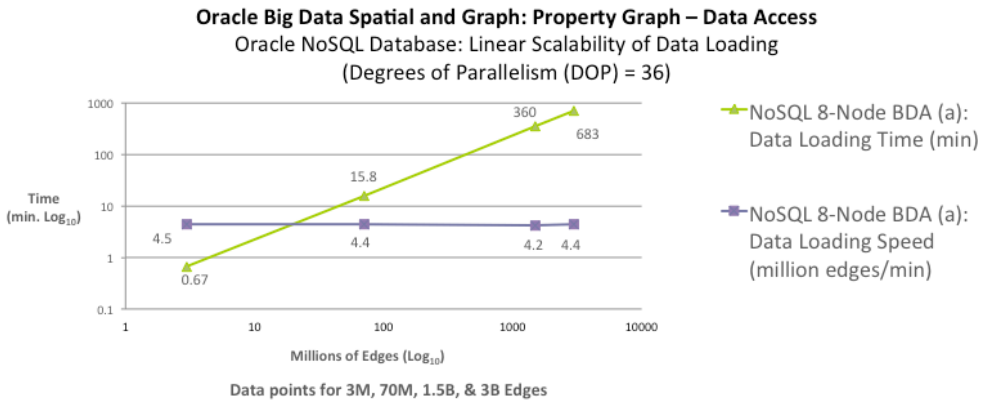
### Performance on Oracle NoSQL Database

Graph data loading, scanning, graph operations and in-memory analyst performance were benchmarked on Oracle NoSQL Database Enterprise Edition 3.2.5. The test results are as follows:

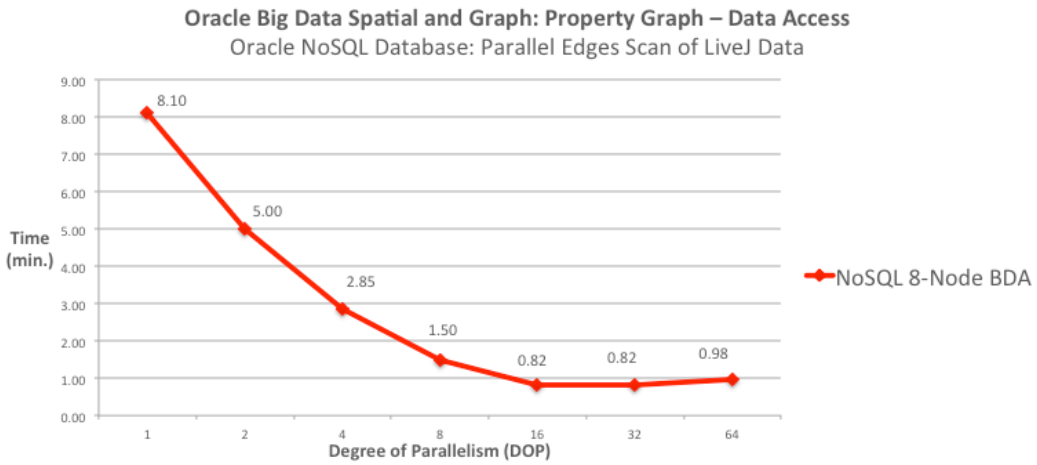
**8 Minutes to Load 68.9m Edges (2.9 GB) in NoSQL w/ Parallelism**



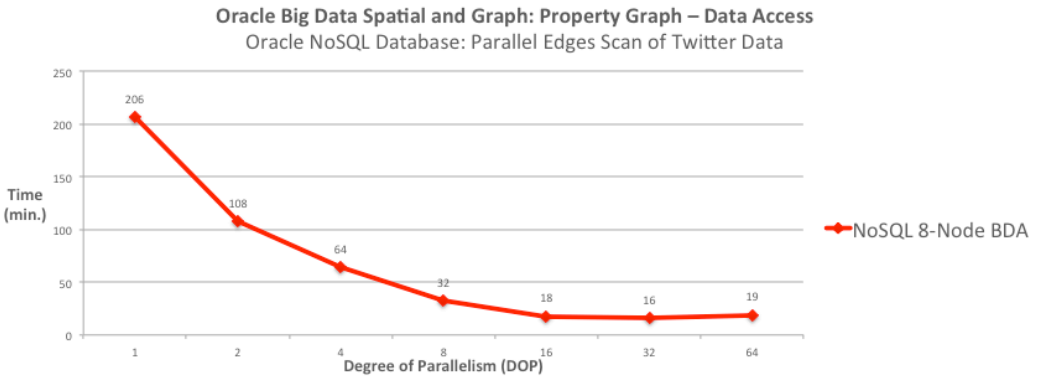

**Linear Scalability Loading in NoSQL w/ Parallelism**



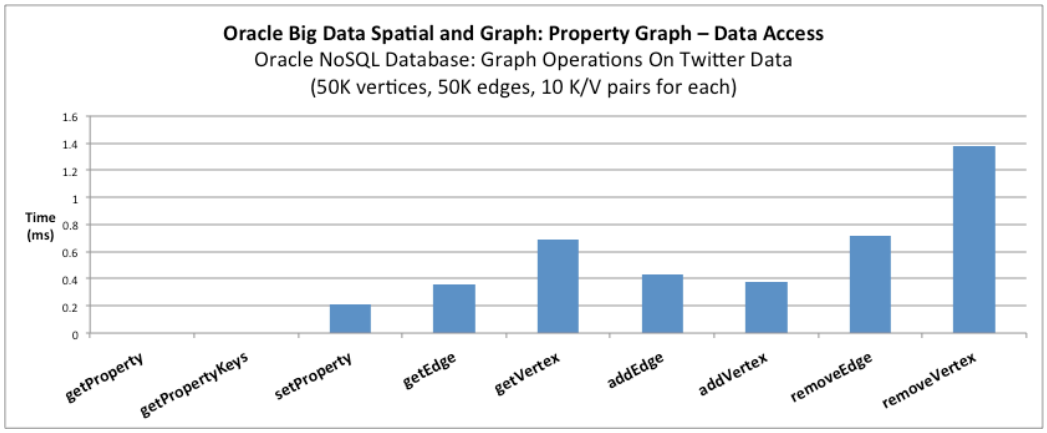

**49 Seconds to Scan 68.9m Edges (2.9 GB) in NoSQL w/ Parallelism**



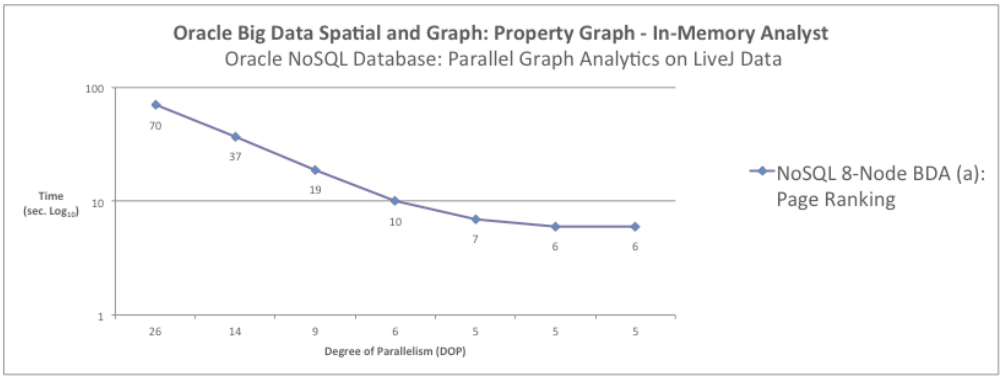

**16 Minutes to Scan 1.5B Edges (68.9 GB)in NoSQL w/ Parallelism**




**Sub-millisecond Performance for Graph Operations in NoSQL**



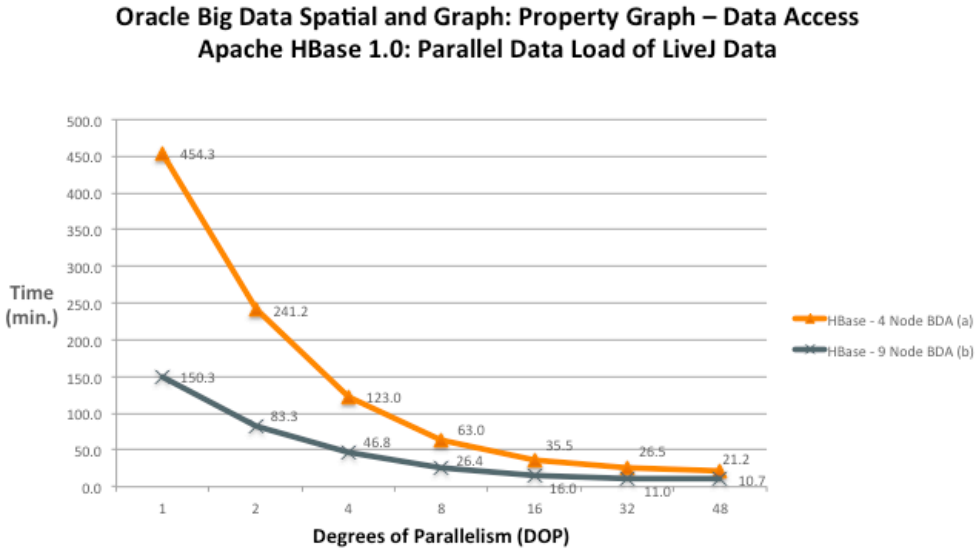

**6 Seconds to Page Rank 4.8m Vertices w/ 68.9m Edges (2.9 GB) w/ In-Memory Analyst**



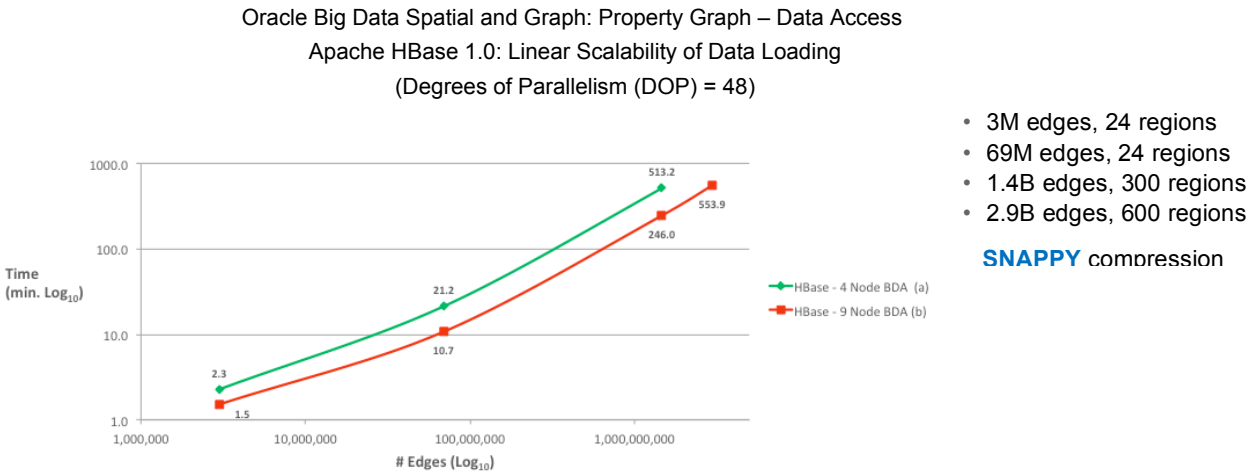

### Performance on Apache HBase

Graph data loading, scanning, text indexing and search, graph operations and in-memory analyst performance were benchmarked on Apache HBase. The test results are as follows:

**11 Minutes to Load 68.9m Edges (2.9 GB) w/ 8 Servers & Parallelism**



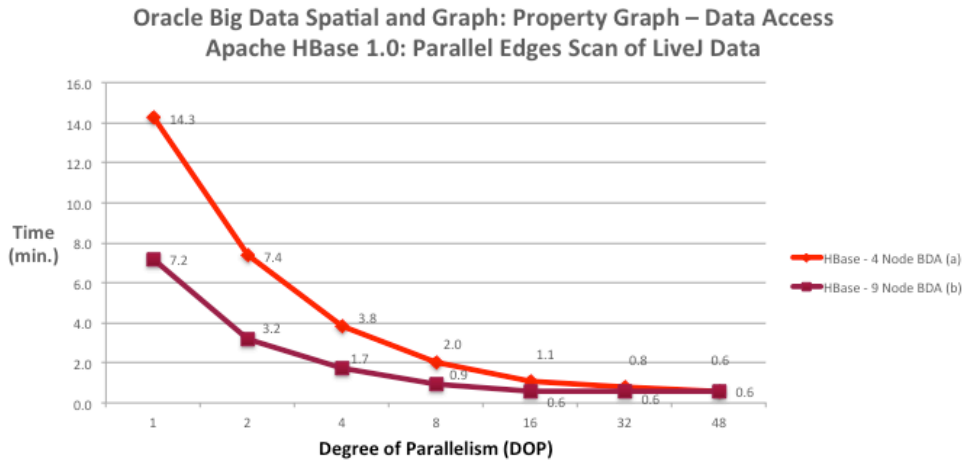
### Linear Scalability Loading in HBase w/ Parallelism



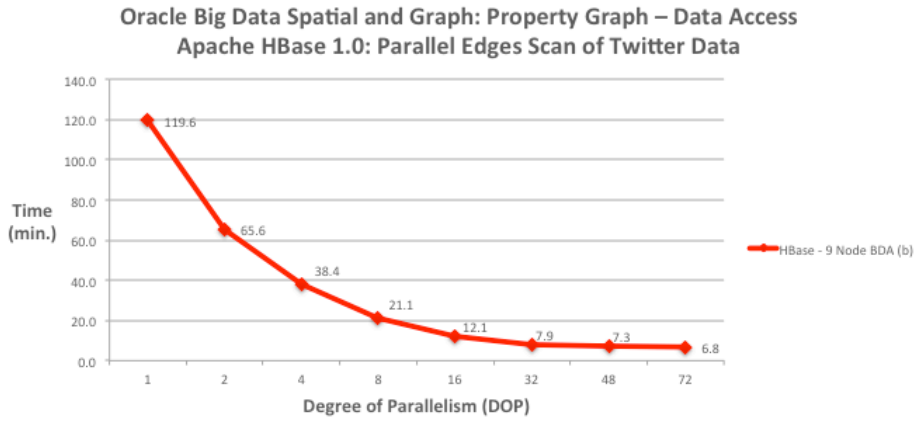

- 3M edges, 24 regions
- 69M edges, 24 regions
- 1.4B edges, 300 regions
- 2.9B edges, 600 regions

**SNAPPY** compression

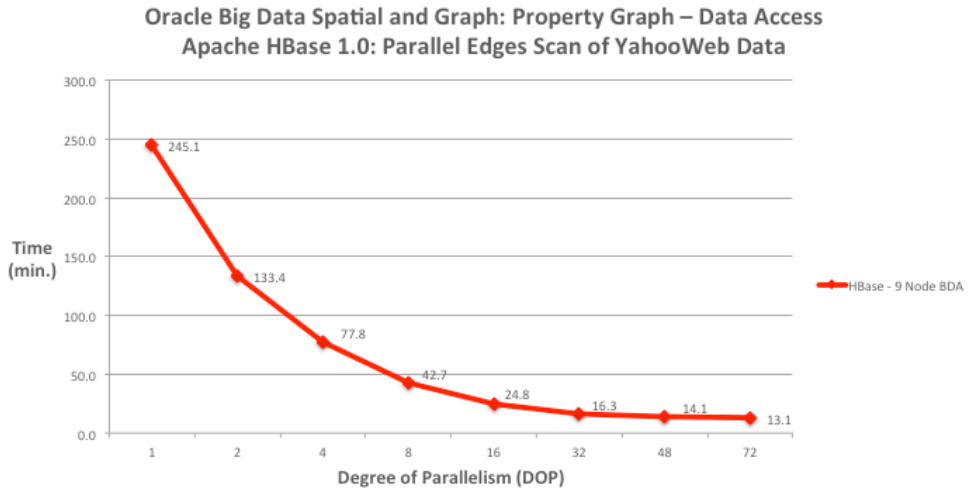
### 34 Seconds to Scan 68.9m Edges (2.9 GB) w/ 8 Servers & Parallelism



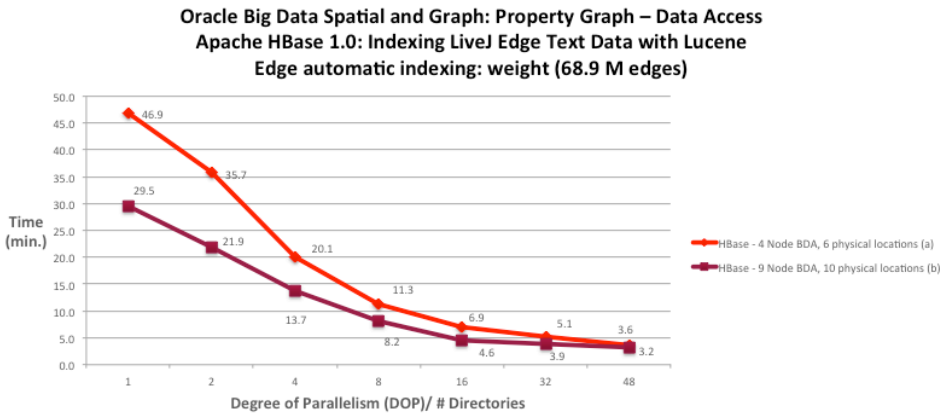

### 7 Minutes to Scan 1.5B Edges (68.9 GB) w/ 8 Servers & Parallelism



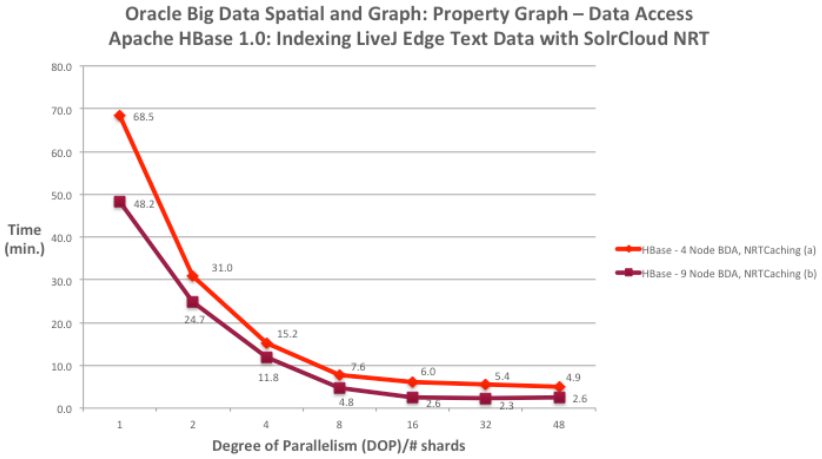
### 13 Minutes to Scan 2.9B Edges (140.3 GB) w/ 8 Servers & Parallelism



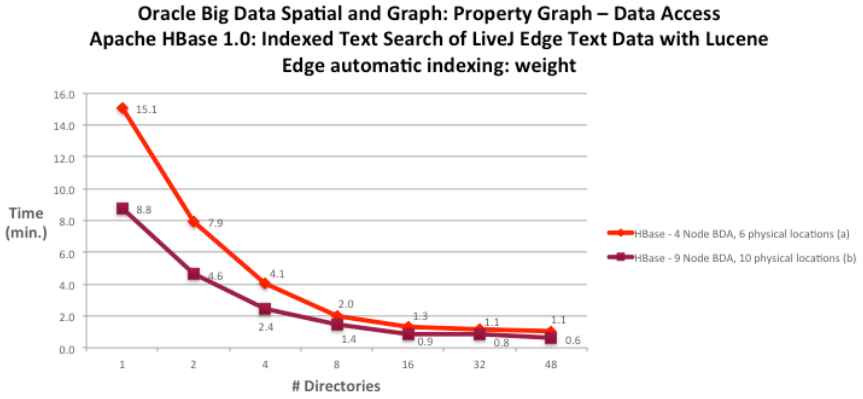

### 3.2 Minutes to Index Text in 68.9m Edges w/ 8 Servers & Parallelism



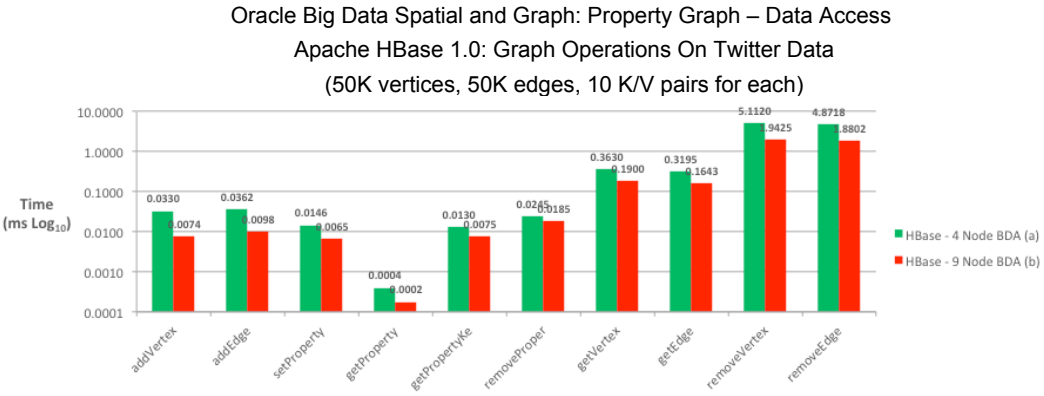

### 2.6 Minutes to Index Text for 68.9m Edges w/ 8 Servers & Parallelism



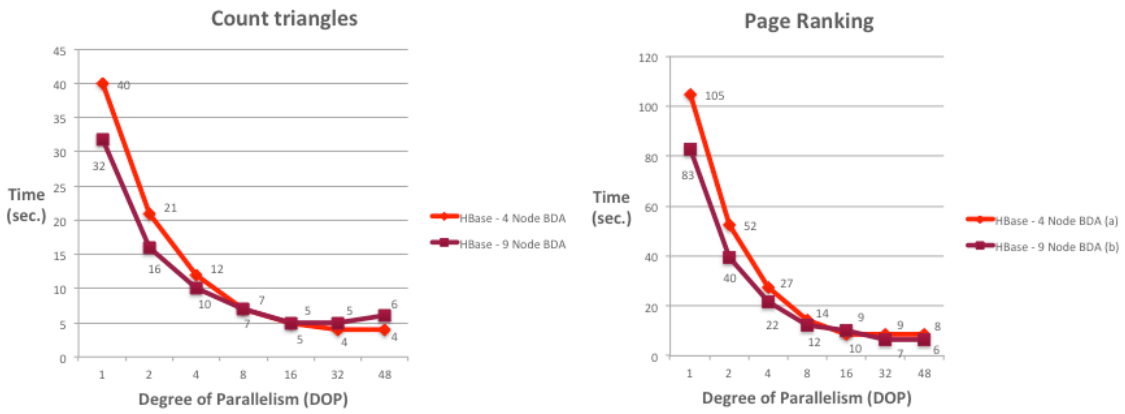
### 37 Seconds to Search Text for 68.9m Edges w/ 8 Servers & Parallelism




### Sub-millisecond Performance for Graph Operations in HBase




### 4-6 Seconds for Analytics on 4.8m Vertices & 68.9m Edges (2.9 GB) w/ In-Memory Analyst



## Performance of in-Memory Analyst Compared to GraphLab and GraphX

### In-memory Analyst Compared to Dato GraphLab Create

The in-memory analyst feature of Oracle Big Data Spatial and Graph property graph was benchmarked and compared to the performance of GraphLab for PageRank and Triangle Counting analytics. The in-memory analyst running on a single node with 32 degrees of parallelism was up to ten times faster than GraphLab distributed on sixteen nodes.

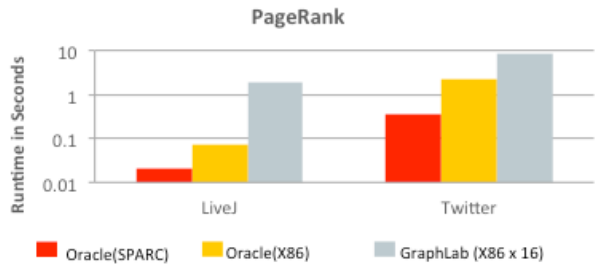
The performance advantage of Oracle Big Data Spatial and Graph is due in large part to its efficient in-memory graph representation and parallelized computation. GraphLab on a single node was over two times slower than GraphLab on sixteen nodes. GraphLab's slower performance on sixteen nodes compared to Big Data Spatial and Graph is due, in part, to GraphLab's messaging overhead. GraphLab partitions the graph and stores partitions on the sixteen nodes. Messages to coordinate operations among the sixteen nodes were a significant source of overhead.

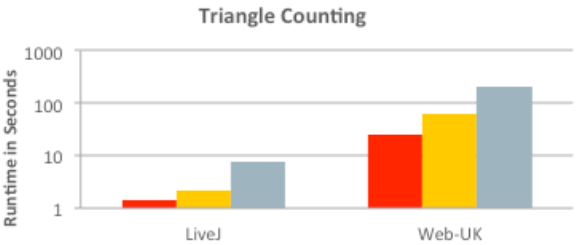
**Details**: Three large public graph instances were used for the benchmark, LiveJournal (LiveJ) with 69 million edges, Twitter data about follower relationships with 1.4 billion edges and Web-UK data with 2.9 billion edges.

Big Data Spatial and Graph in-memory analyst performance was tested on x86 and SPARC (T5) systems with the in-memory analyst running in the same memory space as the Java application that invoked the analytics.

- The x86-based system was an X4-2 Intel Ivy Bridge with 8 cores, 2 sockets and 256GB RAM.
- The Sparc-based T5 had 4TB RAM and 3.6GHz CPU with128 cores and 1024 threads.

GraphLab ran on sixteen instances of the same x86-based system and the systems were connected by Mellanox Connect-IB InfiniBand.




### In-memory Analyst Compared to Apache Spark GraphX

The in-memory analyst feature of Oracle Big Data Spatial and Graph property graph was benchmarked and compared to the performance of Spark GraphX using Spark version 1.1.0. The results show that Oracle Big Data Spatial and Graph significantly outperformed Spark GraphX by up to two orders of magnitude. Even multiple machines did not improve Spark GraphX performance in a significant way. The performance advantage of Oracle

Big Data Spatial and Graph is due, in large part, to its efficient in-memory graph representation and parallelized computation.

The benchmark tested four common graph algorithms. They are described in Table 1. The Spark 1.4.1 documentation[1] documents GraphX's built-in support for three algorithms - PageRank, Connected Components, and Triangle Counting. Oracle used Scala and GraphX APIs to implement the Single-Source Shortest Path, Hop-distance, and Eigenvector Centrality analytics.

**TABLE 1. THE GRAPH ALGORITHMS TESTED**

| Name | Description |
|---|---|
| Pagerank | A famous graph algorithm that computes relative importance between vertices in a graph from its link structure. This is an iterative algorithm where the same neighborhood traversal is repeated until convergence condition is met. |
| Single-Source Shortest Path | Given a starting vertex, the algorithm computes the length of shortest path to every other vertex in a graph. Specifically, the algorithm uses a method from Bellman-Ford. |
| Hop-distance | Similarly to single-source shortest path, the algorithm computes the minimum hop distance from a given vertex to every other vertex, disregarding edge weights. Note that this computation can be efficiently done through a single breadth-first search. |
| Eigenvector Centrality | This algorithm computes the greatest Eigenvalues of adjacency matrix using power iteration. |

Two large public graph instances were used for the benchmarking as described in Table 2. Random weights were generated on every edge with a uniform distribution for testing the Single-source Shortest Path algorithm. This was done because the original graph instances did not have edge weights.

**TABLE 2. THE CHARACTERISTICS OF THE GRAPH DATA INSTANCES**

| Name | Vertices | Edges | Description |
|---|---|---|---|
| Twitter[2] | 41,652,230 | 1,468,365,182 | The snapshot of followership between twitter users at 2010. |
| Web[3] | 77,741,046 | 2,965,197,340 | Link relations of web-pages in .UK domain |

### Environment

The benchmark was conducted on a homogeneous computing cluster with the following systems and network characteristics.

- **CPU**: Intel "Sandy Bridge", Xeon E5-2660, 2.20 GHz, 8 Cores (x 2 HT)
- **Memory**: 256 GB (DDR3 – 1600)
- **SSD**: 3 x 256 GB (combination of OCZ Vertex 4 and Samsung 840 Pro)
- **Network Card**: Mellanox Connect-IB (InfiniBand Adapter)
- **Switch**: Mellanox SX6512 (InfiniBand Switch)

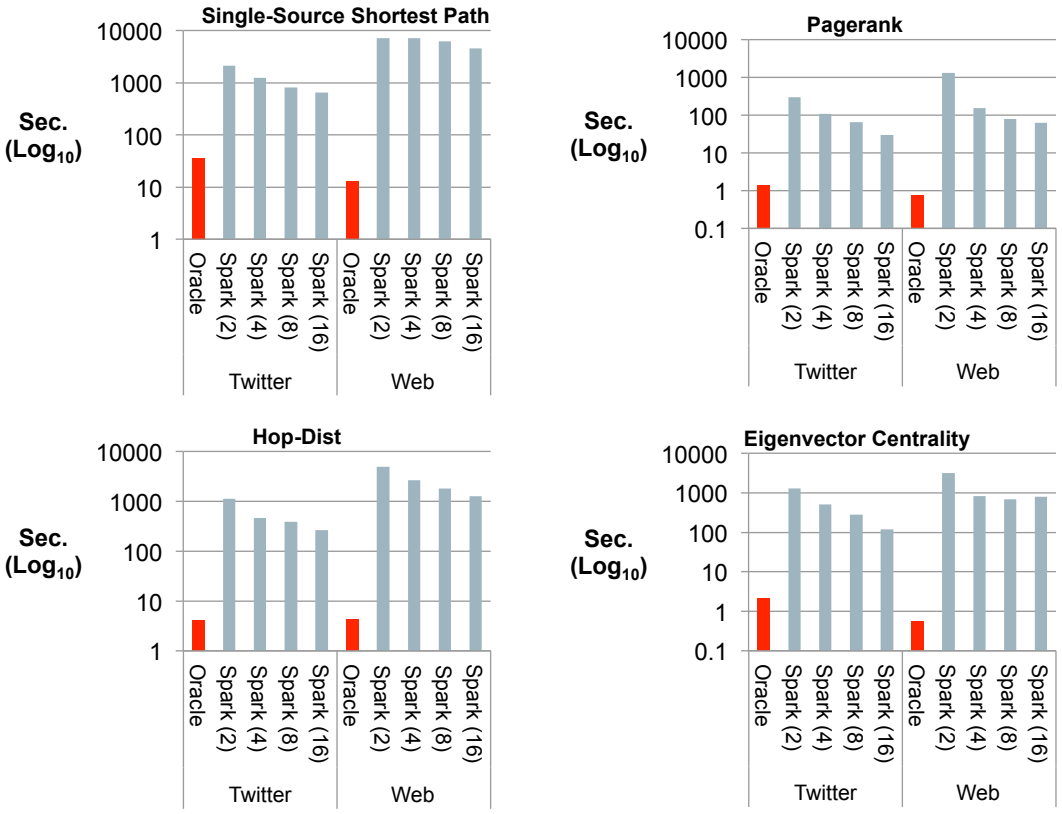
### Results and Discussion

The results of the four tests performed with Oracle Big Data Spatial and Graph in-memory analyst and Apache Spark GraphX on 2 to 16 machines[4] are summarized in the figures below. The execution times are in log-scale.

1 http://spark.apache.org/docs/latest/graphx-programming-guide.html#graph-algorithms

2 H.Kwak, et. al, "What is Twitter, a social network or a news media?" In WWW '10

3 http://webscope.sandbox.yahoo.com/




Total execution time is reported for Single-source Shortest Path and Hop-Dist. Per-iteration execution time is reported for Pagerank and Eigenvector Centrality. The per-iteration execution time is reported because these algorithms repeat the same computation until the arbitrary convergence condition is met; execution time per iteration is more relevant to evaluate the performance of the execution engine.

4 Spark execution was terminated after 2 hours (7200 seconds) while Spark(2) and Spark(4) configuration took more than 2 hours to run the SSSP algorithm on Web graph.

## Conclusion

Oracle Big Data Spatial and Graph provides a comprehensive, massively scalable, fast, and flexible graph database and analytics platform that incorporates and supports open source, including Apache, Java, Tinkerpop, Groovy, and other scripting languages, such as Python.

**Comprehensive**: Users can take advantage of secure databases, text indexing and faceted search, a parallel in-memory analyst with 35 built-in analytics and integration w/ relational & SQL-based analytics. Developers can build applications more quickly with open source Java APIs and scripting languages, and a Groovy console.

**Scalable**: Database storage, and text indexing and search are distributed. The parallel in-memory analyst supports concurrent and multiuser analytics. Filter queries are available to refine and reduce the amount of graph data read into memory for analysis.

**Fast**: Parallel everything: loading, querying, graph operations and the in-memory analyst take full advantage of parallelism in modern system architectures.

**Flexible**: Oracle Big Data Spatial and Graph can be deployed on-premise or in the Cloud, store data in Apache Hbase and Oracle NoSQL Database, search text in graph elements with Lucene and SolrCloud, and deploy the in-memory analyst in a Java program or with any of three Web servers or using Apache YARN. Finally, the extensible analytics architecture makes it easy for Oracle to respond to customer requests for additional built-in graph analytics.

## Appendix A: The In-Memory Analyst's Built-in Analytics

| Property Graph Parallelized Analytics | Description |
|---|---|
| (Classic - ConnectedComponents) sccKosaraju | Finds strongly connected components using Kosaraju's algorithm. This algorithm is parallelized. |
| (Classic - ConnectedComponents) sccTarjan | Finds strongly connected components using Tarjan's algorithm. This algorithm is sequential by nature. |
| (Classic - ConnectedComponents) wcc | Finds weakly connected components through label propagation. |
| (CommunityDetection) communitiesLabelPropagation | Detects communities using parallel label propagation. |
| (PathFinding) AllPaths | The paths from one source vertex to all other vertices. |
| (PathFinding) FattestPath | The Fattest Tree Algorithm computes the fattest path from a source node to all nodes in the graph. The fattest path between two nodes is the path with the highest possible minimum edge capacity. |
| (PathFinding) FindCycle | two algorithms for finding cycles: a robust version always scans the whole graph by performing several DFS traversals, a faster lightweight version performs one DFS traversal |
| (PathFinding) shortestPathBellmanFord | Computes single source shortest paths using Bellman & Ford algorithm time complexity. |
| (PathFinding) shortestPathBellmanFordReverse | Computes reverse single source shortest paths using Bellman & Ford algorithm. |
| (PathFinding) shortestPathDijkstra | Computes shortest path using Dijkstra's algorithm. |
| (PathFinding) shortestPathDijkstraBidirectional | Computes shortest path using a bi-directional Dijkstra variant. |
| (PathFinding) shortestPathFilteredDijkstra | Computes shortest path using Dijkstra's algorithm on a filtered graph The filter specified by the given filter expression is applied on each edge during traversal of the graph. If it evaluates to false, the edge is skipped. |
| (PathFinding) shortestPathFilteredDijkstraBidirectional | Computes shortest path using a bi-directional Dijkstra variant on a filtered graph. |
| (PathFinding) shortestPathHopDist | Computes hop-distance from given node to every other node. |
| (PathFinding) shortestPathHopDistReverse | Computes reverse hop-distance from given node to every other node. |
| (Ranking) approximateNodeBetweennessCentrality | Computes approximate node betweenness centrality (without considering edge length). The algorithm performs a BFS only from randomly selected k seed nodes instead of every node. |
| (Ranking) approximateNodeBetweennessCentralityFromSeeds | Computes approximate node betweenness centrality (without considering edge length). The algorithm performs a BFS only from the given seed nodes instead of every node. |
| (Ranking) closenessCentralityDoubleLength | Computes closed centrality. The algorithm computes bellman-ford from every node in the graph. |
| (Ranking) closenessCentralityUnitLength | Computes closed centrality. The algorithm computes BFS from every node in the graph. |
| (Ranking) degreeCentrality | Computes degree centrality. |
| (Ranking) eigenvectorCentrality | Computes eigenvector centrality using power iteration (with L1 norm). This algorithm is similar to Pagerank. |

| | |
|---|---|
| (Ranking) hits | This is the Hyperlink-Induced Topic Search (HITS) algorithm, also known as Hubs and Authorities. The implementation uses an iterative method. At each iteration step, the hub and the authority values of all the nodes in the graph are computed |
| (Ranking) inDegreeCentrality | Computes in-degree centrality. |
| (Ranking) nodeBetweennessCentrality | Computes node betweenness centrality (without considering edge length). The algorithm is implemented by performing BFS from every node in the graph. |
| (Ranking) outDegreeCentrality | Computes out-degree centrality. |
| (Ranking) pagerank | Classic pagerank algorithm. |
| (Ranking) Approximate Pagerank | a faster variant of Pagerank that can be used when less precision is acceptable. |
| (Ranking) personalizedPagerank | Evaluates relative importance of nodes in a graph with respect to a given node v. That is the probability of ending at each vertex u, when performing random walk from the given node v with probability of c to restart from the start node. The implementation directly emulates random walk via Monte-Carlo method. |
| (Ranking) Weighted Pagerank | Considers edge weights. |
| (Ranking) randomWalkWithRestart | Performs a ego-centric random walk on the graph starting at a source node. At each node visited, the walk will jump back to source with a given reset probability, or if the node degree is zero. If the walk does not jump back, it will choose a random outgoing edge and jump to the destination node. The walk finishes after a given length nodes have been visited (not necessarily distinct nodes). |
| (Recommendation) salsa | This is the link prediction algorithm used by Twitter's follower recommendation engine. The algorithm chooses a set of nodes for recommendation from a given bipartite graph. |
| (Recommendation) Personalized SALSA | Evaluates the relative importance of nodes with respect to a given set of hub nodes. |
| (Recommendation) whomToFollow | The Who to Follow recommendation algorithm by Twitter, Inc. This algorithm performs a random walk from v and extracts the top visited vertices to determine the circle of trust. It then creates a BipartiteGraph with the circle of trust as left set and runs the salsaAsync(BipartiteGraph, int, double, double, int) recommendation algorithm on that graph to compute the hubs and authorities. |
| (StructureEvaluation) conductance | Computes conductance. |
| (StructureEvaluation) countTriangles | Counts the number of 'triads' in the given undirected graph. |
| (StructureEvaluation) K-Core | Computes k-core decomposition of a graph to identify the layers in a graph, from external to more central vertices. |
| (StructureEvaluation) inDegreeDistribution | Computes the indegree distribution of the given graph and stores it in the map. |
| (StructureEvaluation) outDegreeDistribution | Computes the outdegree distribution of the given graph and stores it in the map. |
| (StructureEvaluation) partitionConductance | Computes average conductance among all partitions. |
| (StructureEvaluation) partitionModularity | Computes modularity of partitions. |
| (StructureEvaluation) PRIM | Finds minimum spanning trees in a graph. |
| (StructureEvaluation) sparsify | Computes a ranking for the graphs edges that ranks them according to their importance in the graph. The importance is basically measured by counting the number of triangles the edge is involved in. Also computes the number of edges to keep for each node using a given sparsification coefficient. The number of edges is the node's degree to the power of the coefficient. These two results can then be used to create a sparsified subgraph. |

| Property Graph In-Database Analytics | Description |
|---|---|
| (Pathfinding) getShortestPath | Computes shortest path using Dijkstra's algorithm. |

**Oracle Corporation, World Headquarters**
500 Oracle Parkway
Redwood Shores, CA 94065, USA

**Worldwide Inquiries**
Phone: +1.650.506.7000
Fax: +1.650.506.7200

ORACLE

CONNECT WITH US


blogs.oracle.com/oracle


facebook.com/oracle


twitter.com/oracle


oracle.com

**Hardware and Software, Engineered to Work Together**



ORACLE BIG DATA SPATIAL AND GRAPH PROPERTY GRAPH: FEATURES AND PERFORMANCE
December 2017
Bill Beauregard


Oracle is committed to developing practices and products that help protect the environment