15799

Final Project Presentation

Dec. 2nd, 2013

Qing Zheng & Atreyee Maiti

Goals

Graph Queries

- How different DBs handle large graph?
- What's the differences in performance?
- What DB to which for a specific use-case?

Datasets/System

Neo4j v.s. MySQL

- the most popular open-source DB for each community

Datasets/System

Neo4j v.s. MySQL

- the most popular open-source DB for each community

Wikipedia Datasets

- Reasonably big, easily accessible, and people are familiar with it

Experimental Settings

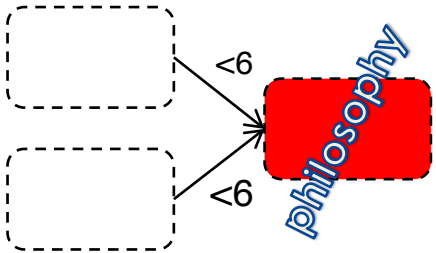
Amazon EC2

- Neo4j 1.9.5
- MySQL 5.5.34
- Ubuntu 12.04.3
- 1 CPU, 4GB RAM, 410GB Disk (m1.medium)

Benchmarks

Queries

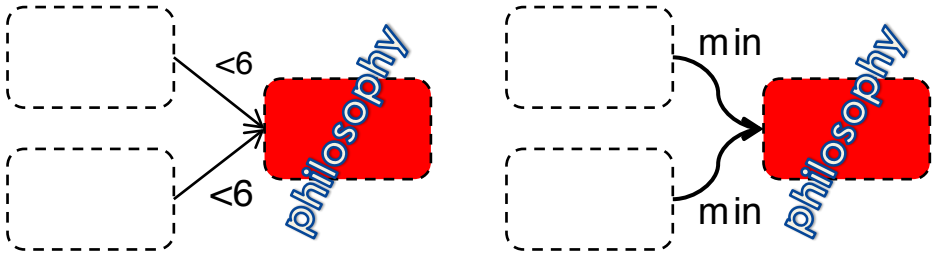
- Six-Degree



Benchmarks

Queries

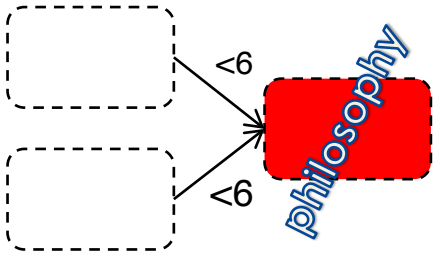
- Six-Degree
- Shortest Path

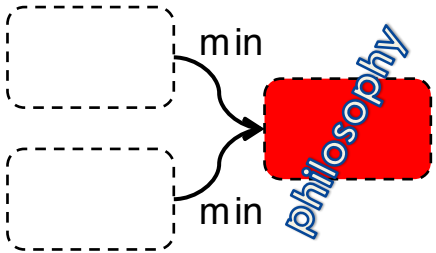


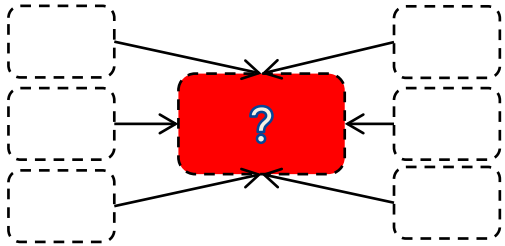
Benchmarks

Queries

- Six-Degree
- Shortest Path
- Most Cited Page







API

Client Interface

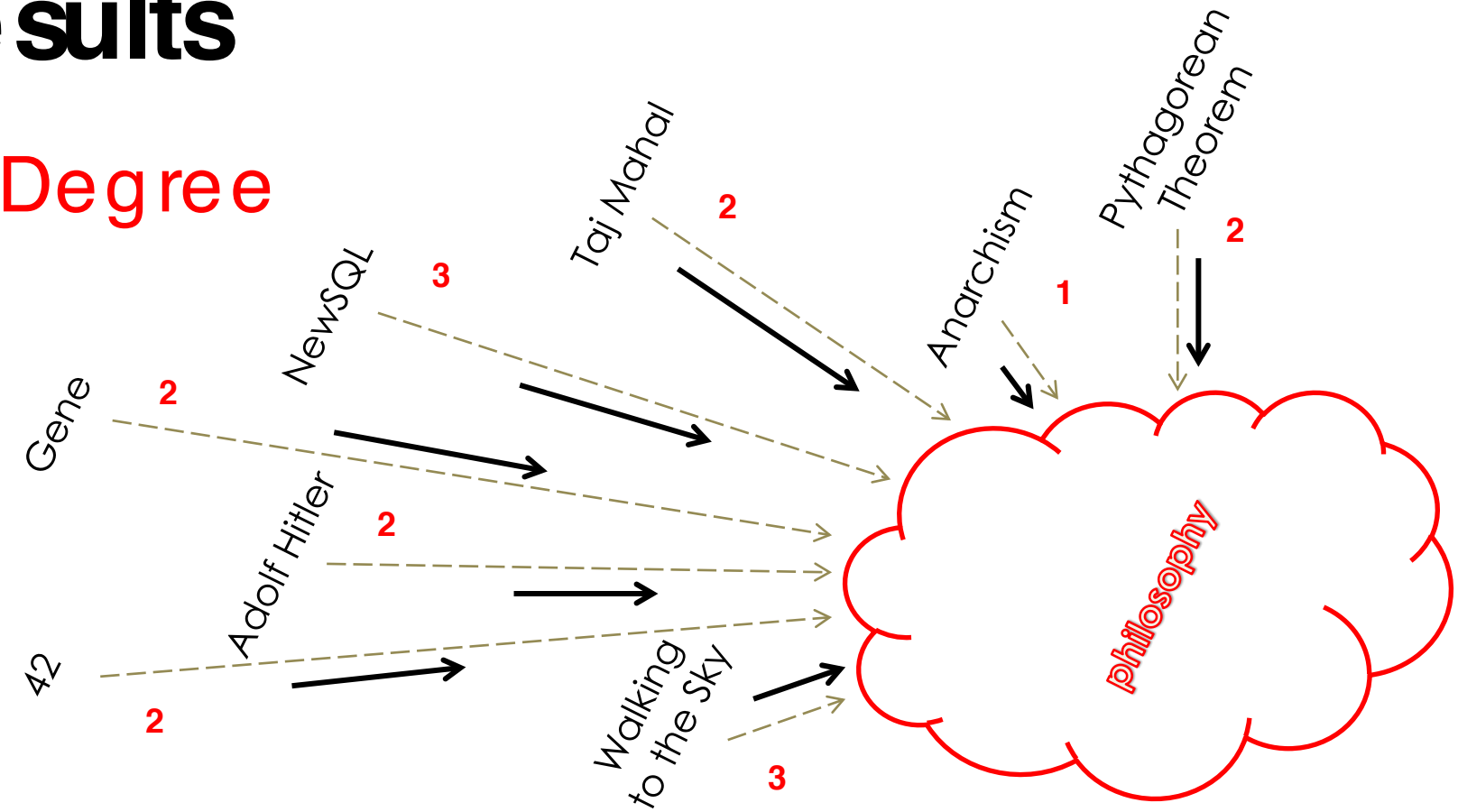
- SQL for MySQL
- Java API for Neo4j

Results

Six Degree

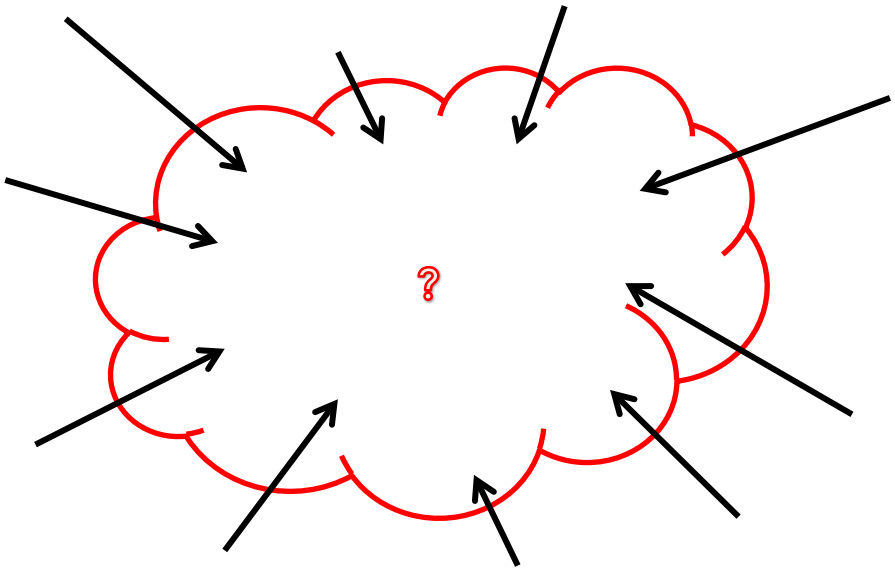
Results

Six Degree



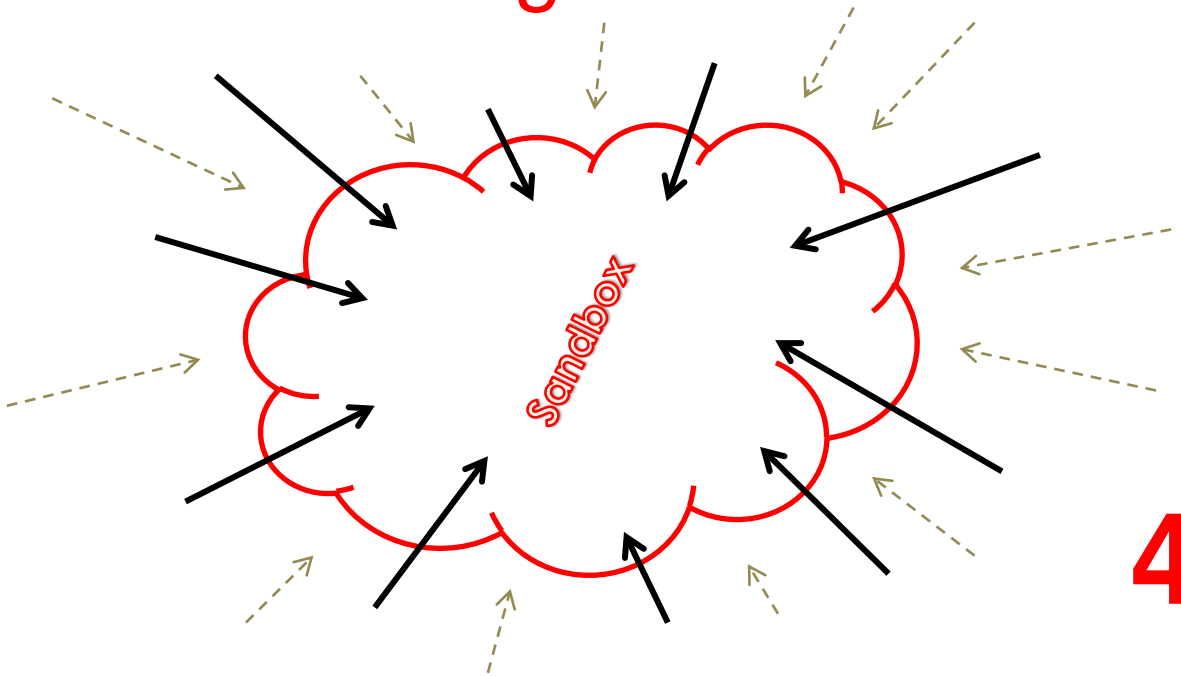
Results

Most Cited Page



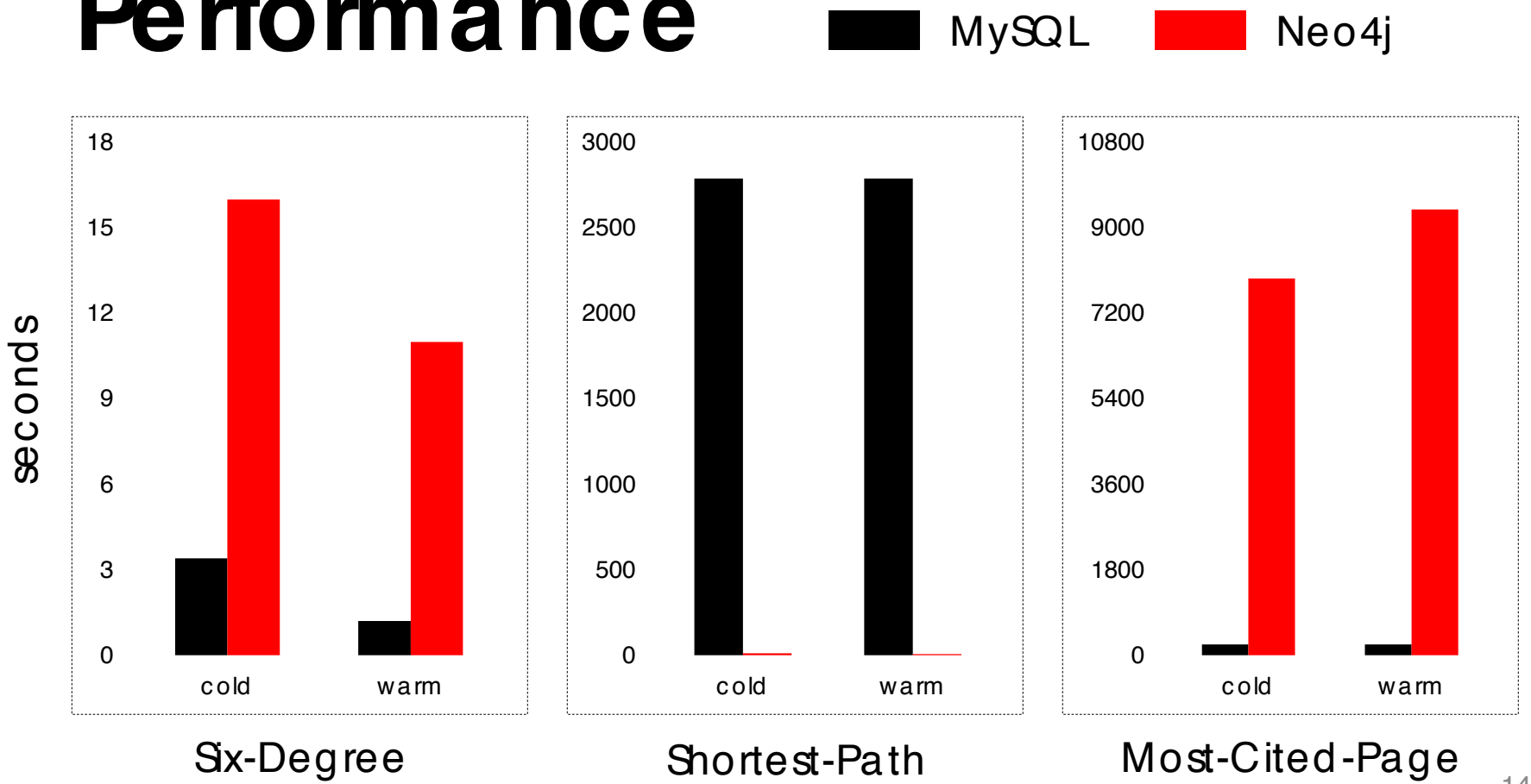
Results

Most Cited Page



4,673,396

Performance



MySQL outperforms Neo4j in most cases (apart from shortest path) even though Neo4j has a smaller dataset(10M nodes and 97M relationships)

The world's most popular open source database

ANALYSIS SYSTEM WISE

Storage Engine

INNODB

- Reliable, high-performance transactional engine

MYISAM

- Read-optimized, data-warehouse class engine
- Dedicated in-memory buffer for index blocks
- Uses OS page cache for buffering data blocks

Bulk Loading

Best Practices

- Convert SQL inserts into raw CVS Files
- Build indices after data is fully loaded
- Increase “MySIAM_Sort_Buffer_Size”
- Add more memory

Tuning

Optimizing for workloads

- Compression (total table size after compression: 26G)
- Resign table schemas
- Add/remove Indices
- Set index cache to 25% of the RAM
- Disable query cache (not for optimization)

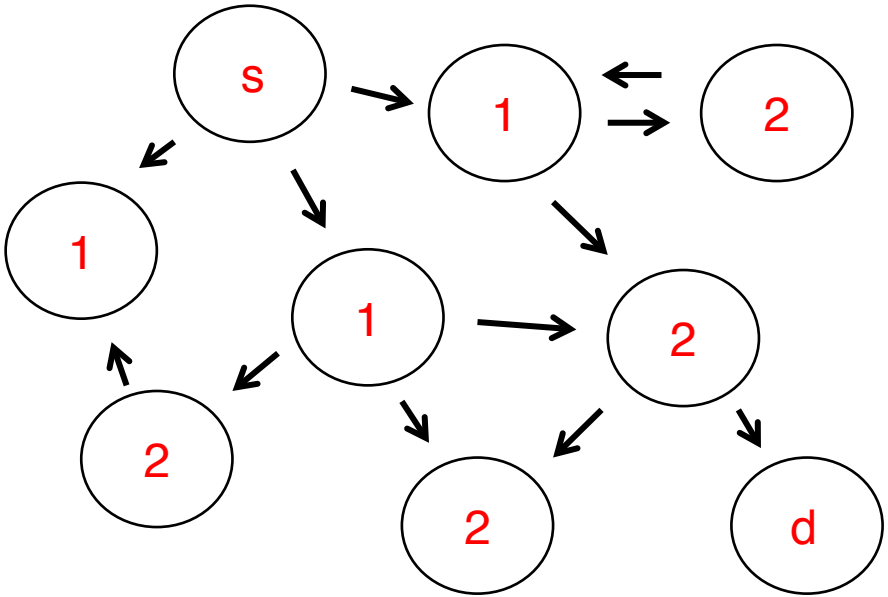
Schema Profile

Wiki Datasets

- **31**,293,738 pages
- **709**,804,739 links

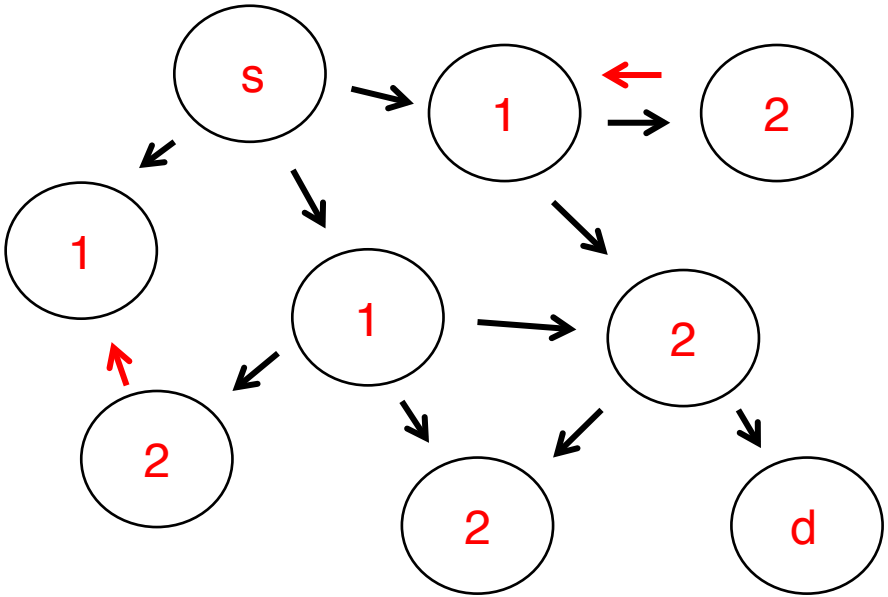
Six Degree Query

Breath-First Search



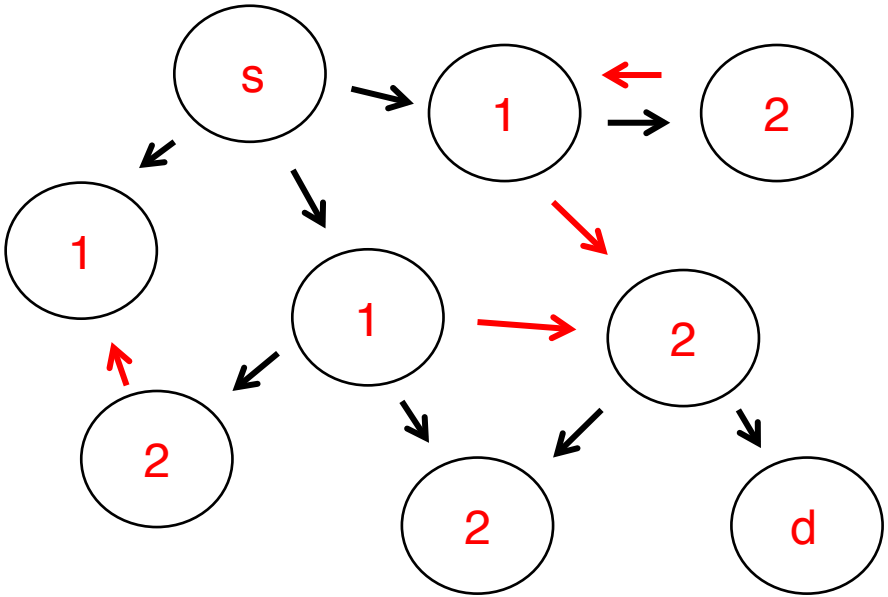
Six Degree Query

Breath-First Search



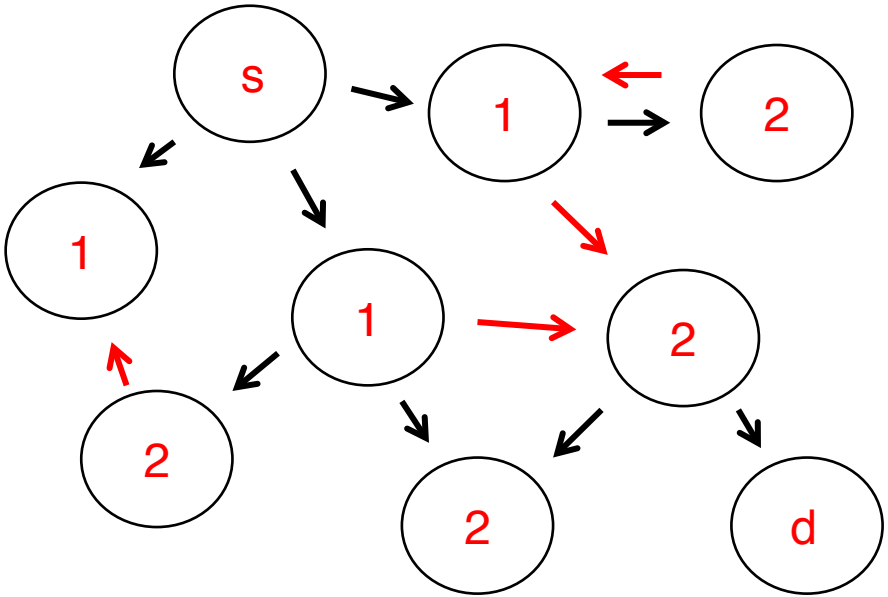
Six Degree Query

Breath-First Search



Six Degree Query

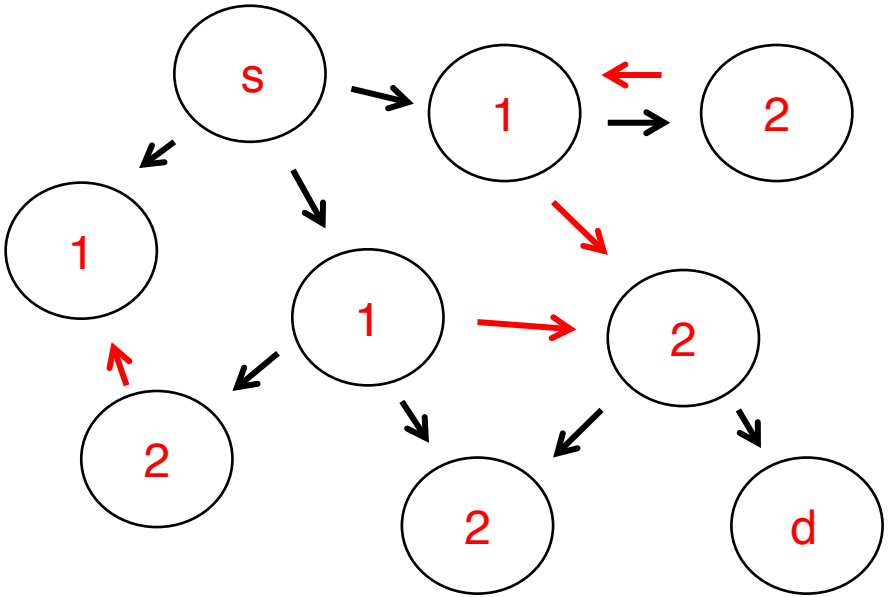
Breath-First Search



Group By / Subquery

Six Degree Query

Breath-First Search



Insert Ignore Into …

Group By … Subquery

Six Degree Query

Ignoring Breath-First Search

- 1/44th index block read requests
- No additional sorting
- 5x more rows in temporary tables

>>> 20x performance boost

Six Degree Query

Ignoring Breath-First Search

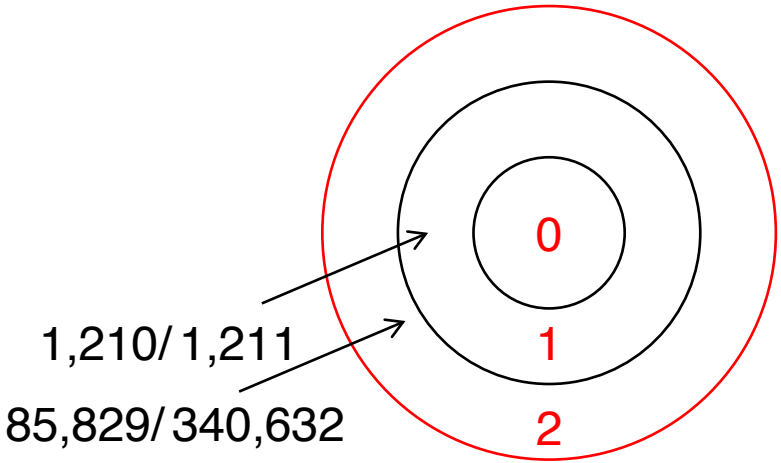
- 1/44th index block read requests
- No additional sorting
- 5x more rows in temporary tables

>>> 20x performance boost

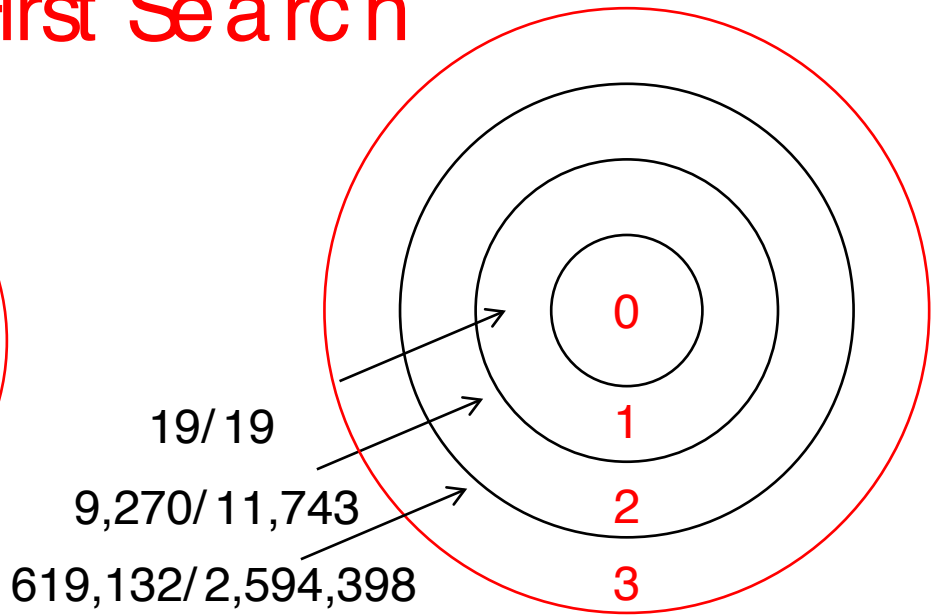
Need to keep temp table short!

Six Degree Query

Ignoring Breath-First Search



Adolf-Hitler



Walk-to-the-Sky

Six Degree Query

Bidirectional Breath-First Search

- 1/34th rows in temporary tables
- 1/386th index block read requests
- 1/5th index block write requests

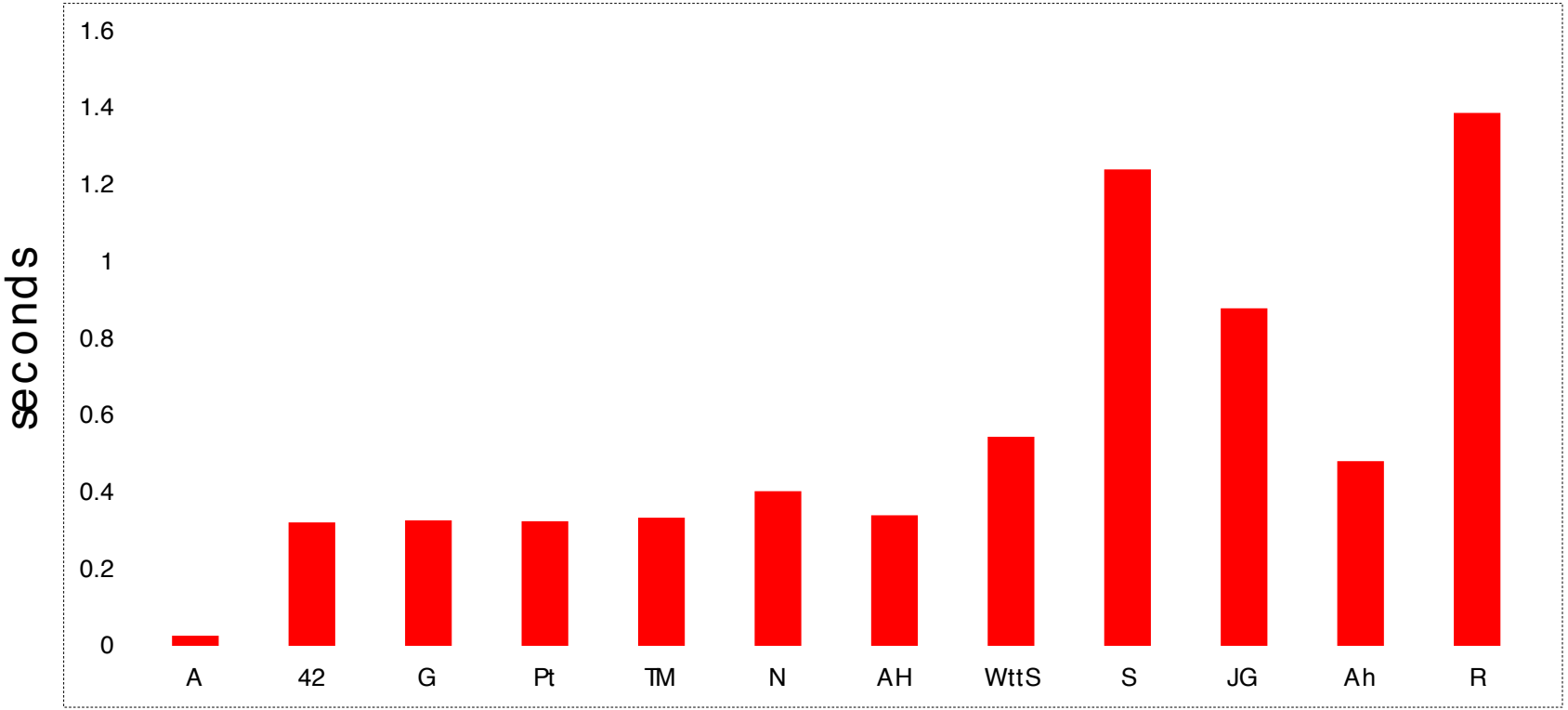
>>> 720x additional performance boost

Shortest Path Query

Bidirectional Batched Shortest Path

>>> 318x performance boost

Shortest Path Query



2,786 secs

W

Most Cited Page

Count(*) & Group-BY & Order-By & Limit

Most Cited Path

Count(*) & Group-BY & Order-By & Limit

```
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows      | Extra                                         |
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
|  1 | SIMPLE      | links | index | NULL          | REVERSE | 8       | NULL | 709804739 | Using index; Using temporary; Using filesort |
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
```

Most Cited Path

Count(*) & Group-BY & Order-By & Limit

```
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows      | Extra                                         |
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
|  1 | SIMPLE      | links | index | NULL          | REVERSE | 8       | NULL | 709804739 | Using index; Using temporary; Using filesort |
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
```

Most Cited Path

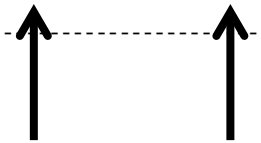
Count(*) & Group-BY & Order-By & Limit

```
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows      | Extra                                         |
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
|  1 | SIMPLE      | links | index | NULL          | REVERSE | 8       | NULL | 709804739 | Using index; Using temporary; Using filesort |
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
```

Most Cited Path

Count(*) & Group-BY & Order-By & Limit

```
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows      | Extra                                         |
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
|  1 | SIMPLE      | links | index | NULL          | REVERSE | 8       | NULL | 709804739 | Using index; Using temporary; Using filesort |
+----+-------------+-------+-------+---------------+---------+---------+------+-----------+-----------------------------------------------+
```



Most Cited Path

Sort Buffer

- 2MB: 33 merge passes
- 8MB: 8 merge passes
- 64MB: 1 merge pass

Most Cited Path

Sort Buffer

- 2MB: 33 merge passes
- 8MB: 8 merge passes
- 64MB: 1 merge pass

>>> 0x performance improvements

Most Cited Path

Sort Buffer

- 2MB: 33 merge passes
- 8MB: 8 merge passes
- 64MB: 1 merge pass

>>> 0x performance improvements

45x more rows scanned than sorted

Quick Summary

MySQL-MySIAM

- Loading takes time
- Pay attention to query algorithms
- Limited performance for large joins
- Nice documentation with good out-of-box performance for analysis

ANALYSIS SYSTEM WISE

Data cleaning/importing

Importing tool

- use of graphipedia to import compressed dataset
- LinkExtractor to transform xml to a links xml
- Import graph which uses the links to create nodes and then relationships. Also indexes the data

Graph Structure

Node: pages with property "title"

Relationship: "Link"

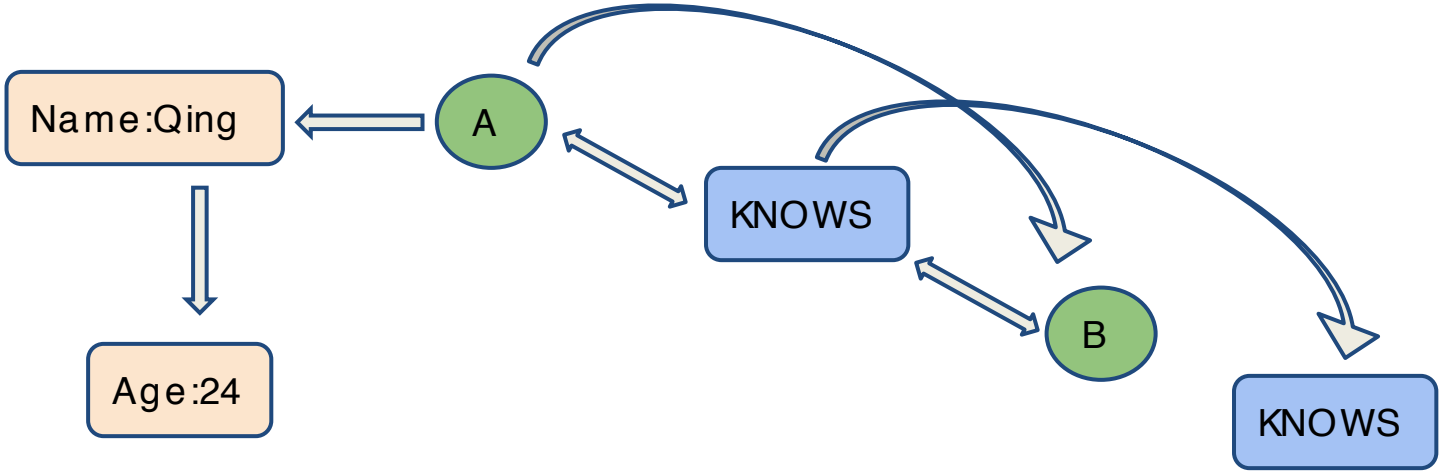
Lucene index

Algorithm implementation

Neo4j GraphAlgoFactory

Benchmark	**Implementation**
six degree	findSinglePath with max depth
shortest path	shortestPath
most cited node	get all relationships, maintain count

Internals



- records basically linked list of nodes, relations - suffers when need to traverse a lot of linked lists - most cited page
- major win is joins - and then it becomes dependent on configuration and resource availability

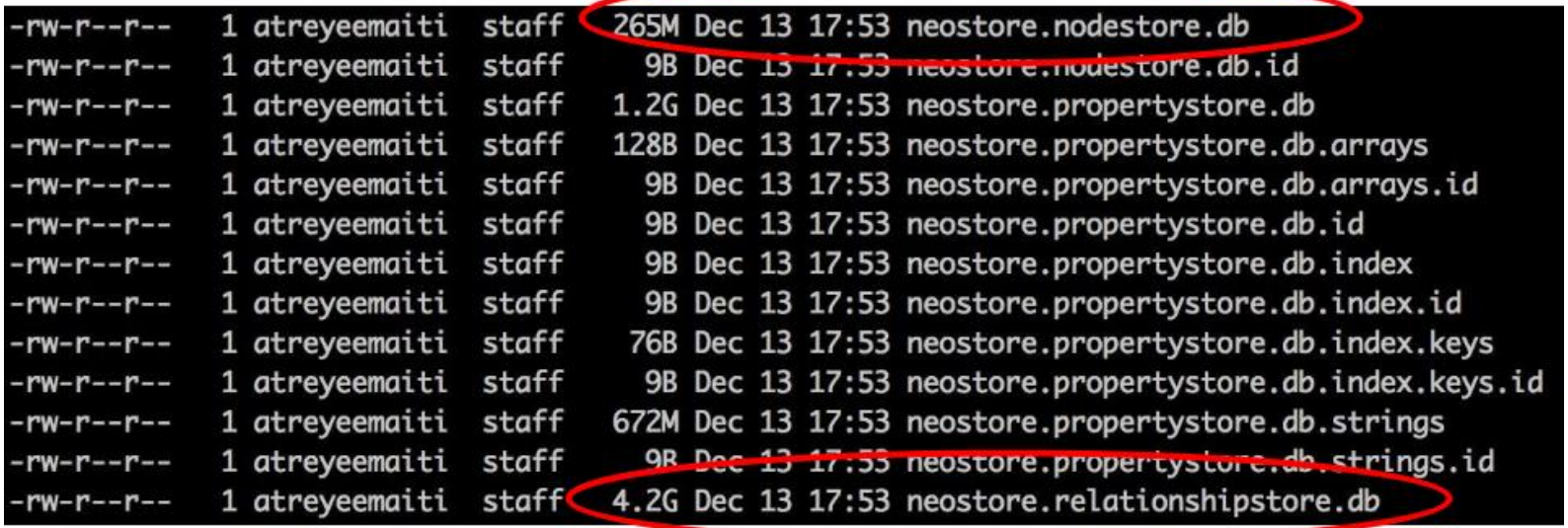
Caching

Two types

- file buffer caching - use of native i/o to cache data in memory - storage file data similar representation as disk for fast traversal
- object caching - using the allocated area for the heap - caches individual nodes and relationships and their properties in a form that is optimized for fast traversal of the graph - relies on garbage collection for eviction from the cache in an LRU manner.

cache levels

- in heap
- in file buffer cache
- disk


-rw-r--r-- 1 atreyeemaiti staff 265M Dec 13 17:53 neostore.nodestore.db
-rw-r--r-- 1 atreyeemaiti staff 9B Dec 13 17:53 neostore.nodestore.db.id
-rw-r--r-- 1 atreyeemaiti staff 1.2G Dec 13 17:53 neostore.propertystore.db
-rw-r--r-- 1 atreyeemaiti staff 128B Dec 13 17:53 neostore.propertystore.db.arrays
-rw-r--r-- 1 atreyeemaiti staff 9B Dec 13 17:53 neostore.propertystore.db.arrays.id
-rw-r--r-- 1 atreyeemaiti staff 9B Dec 13 17:53 neostore.propertystore.db.id
-rw-r--r-- 1 atreyeemaiti staff 9B Dec 13 17:53 neostore.propertystore.db.index
-rw-r--r-- 1 atreyeemaiti staff 9B Dec 13 17:53 neostore.propertystore.db.index.id
-rw-r--r-- 1 atreyeemaiti staff 76B Dec 13 17:53 neostore.propertystore.db.index.keys
-rw-r--r-- 1 atreyeemaiti staff 9B Dec 13 17:53 neostore.propertystore.db.index.keys.id
-rw-r--r-- 1 atreyeemaiti staff 672M Dec 13 17:53 neostore.propertystore.db.strings
-rw-r--r-- 1 atreyeemaiti staff 9B Dec 13 17:53 neostore.propertystore.db.strings.id
-rw-r--r-- 1 atreyeemaiti staff 4.2G Dec 13 17:53 neostore.relationshipstore.db

Tuning/choices made

JVM options:

initial heap size = 512m

max heap size = 1024m

CMSInitiatingOccupancyFraction=50

UseConcMarkSweepGC

Cache type:

weak cache type (object cache) - Provides short life span for cached objects. Suitable for high throughput applications where a larger portion of the graph than what can fit into memory is frequently accessed.

Memory mapping options: (based on sizes of the corresponding store files)

nodes = 200M

relationships = 5G

propertystore = 500M

Optimizations impact

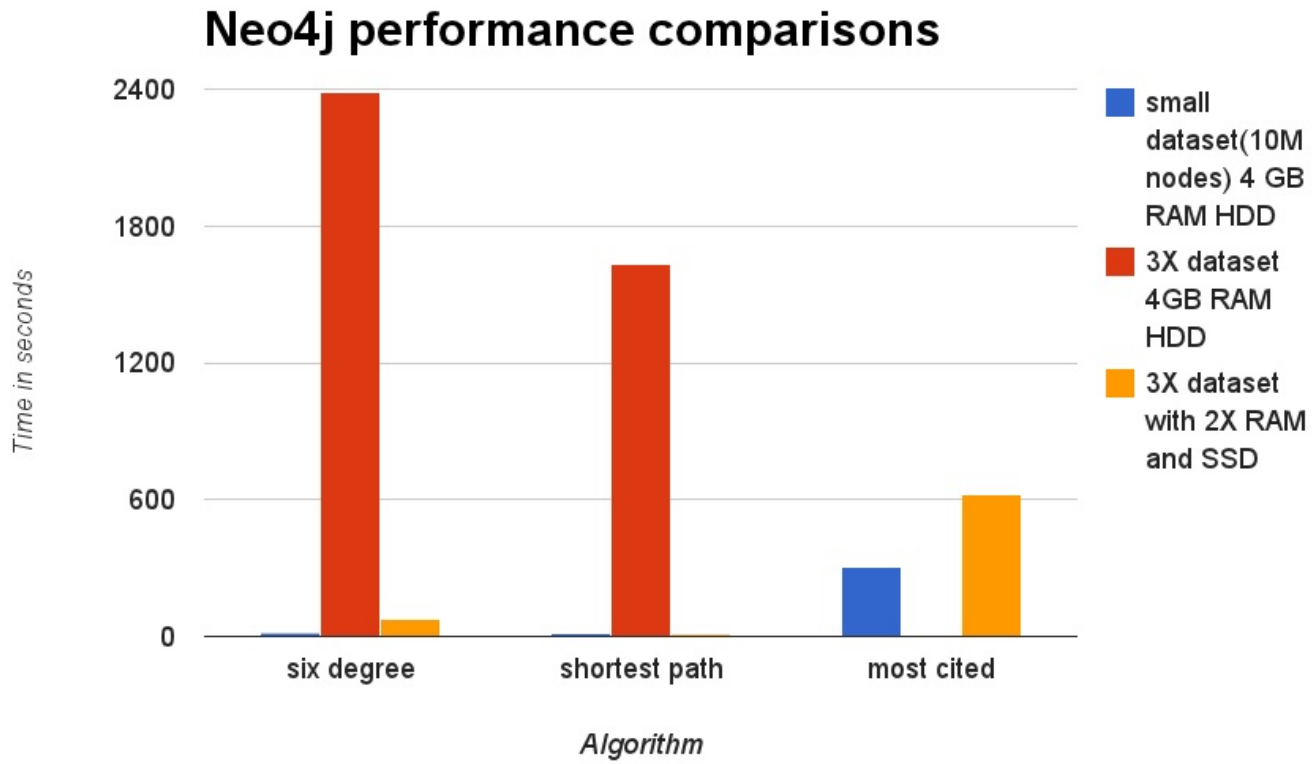
- Make a lot of difference for long running queries
- default tunings most cited node => 23 minutes with optimizations => ~5 mins!!

Run time split

- reading into memory - for warm reads are from memory
- populate file buffer cache
- waiting on gc
- query execution

More Resources

Algorithm	**On 8GB RAM, SSD**	**On 4GB ram, HDD**
Six degree	5 seconds	11
Shortest path	5 seconds	11
Most cited node	161 seconds	256


Neo4j performance comparisons
2400
1800
1200
600
0
Time in seconds
six degree
shortest path
most cited
Algorithm
small dataset(10M nodes) 4 GB RAM HDD
3X dataset 4GB RAM HDD
3X dataset with 2X RAM and SSD

CONCLUSION

Takeaways

- for running graph algorithms that require joins using relational, the algorithm needs to be optimized to a large extent - for unknown destination graph algorithms, mysql is very poor
- for scanning entire tables, mysql is good but neo4j performs poorly
- neo4j has a lot of knobs that can help it in performing fast but they need to be known and explored. Increasing heap is not always the solution!!
- mysql isam query performance is good out of box, needs tuning mostly for importing large data
- performance also depends on the structure of your graph - how large it is, the fan out of the graph
- all in all it is a use case based decision that should be taken

System specific learnings

Description	system	comments
highly connected nodes problem	neo4j specific	neo4j 2.1 will be solving this but not yet released
neo4j has a huge set of algorithms that can be used out of the box	neo4j specific	
neo4j community is very active	neo4j specific	

Future work

- **Incorporate algorithms which require multiple hops**
- **With same setup:**
  - run on postgresql
  - running on SSD
- **On new systems:**
  - Graph processing systems
  - SciDB
- **neo4j feedback**

THANKS!

Acknowledgements

We thank Professor Andy Pavlo for giving us direction at various points in the project. We are also grateful to AWS for the funding towards running the experiments.

References

http://www.slideshare.net/thobe/an-overview-of-neo4j-internals
http://event.cwi.nl/grades2013/07-welc.pdf
http://docs.neo4j.org/chunked/milestone/configuration-caches.html
http://www.slideshare.net/markhneedham/football-graph-neo4j-and-the-premier-league
https://github.com/mirkonasato/graphipedia
http://istc-bigdata.org/index.php/benchmarking-graph-databases/
http://dev.mysql.com/doc/refman/5.5/en/index.html
http://dumps.wikimedia.org/
http://vldb.org/pvldb/vol5/p358_jungao_vldb2012.pdf