Image from http://fc06.deviantart.net/fs71/f/2012/346/f/d/blueprint_3d_iphone_5_by_dracu_teufel666-d5nsnqe.jpg

Comparison of Open Source Databases in Completing Common Queries

JUSTIN ALBANO • STEPHEN JONES • DOMINICK TOURNOUR

Introduction

Project Overview

- Purpose and goal of analysis
- Background information

Individual Comparisons

- Graph database
- NoSQL database
- SQL database

Group Comparison

- Group analysis and results
- Alternative methods & conclusions

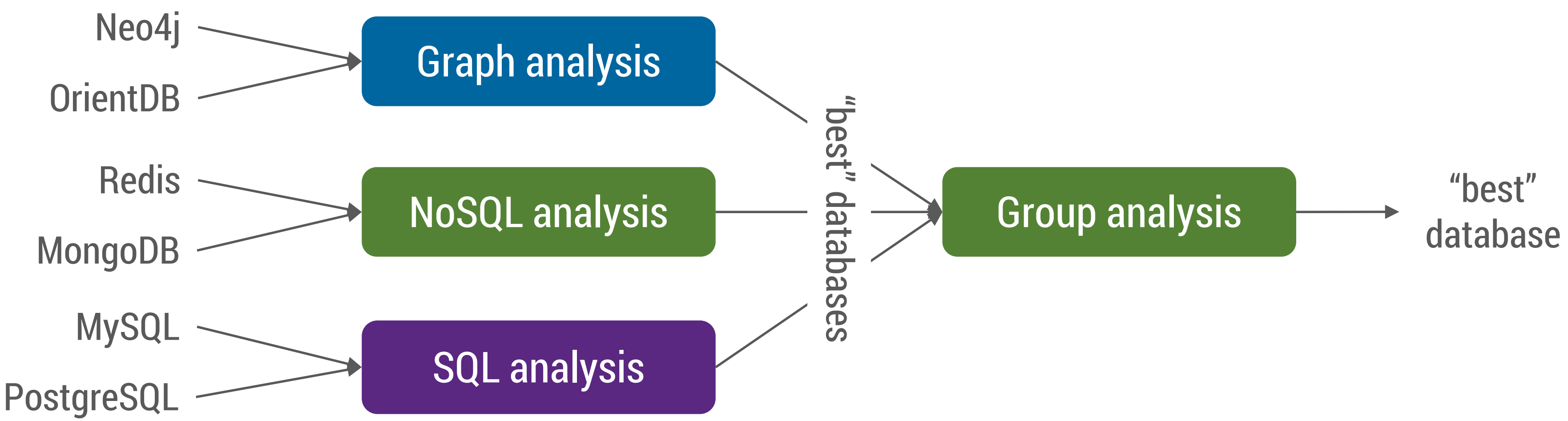
Team Members

Justin Albano
Stephen Jones
Dominick Tournour

Introduction

Purpose & Goals

- Analyze a group of databases from each major category of databases in use today
- Select the best from each group and then compare them head-to-head



Introduction

Design of Experiments

- The specifics of the analysis for each database category varied, but some commonalities exist:
  - Use of algorithms common to the use cases of each domain
  - Generally read-intensive algorithms
  - Both algorithm, and the data set on which the algorithms were executed, varied

Scope & Constraints

- All selected databases are open source
  - Allows the analyst to obtain a free version
  - Allows the analyst to see details that would otherwise be hidden

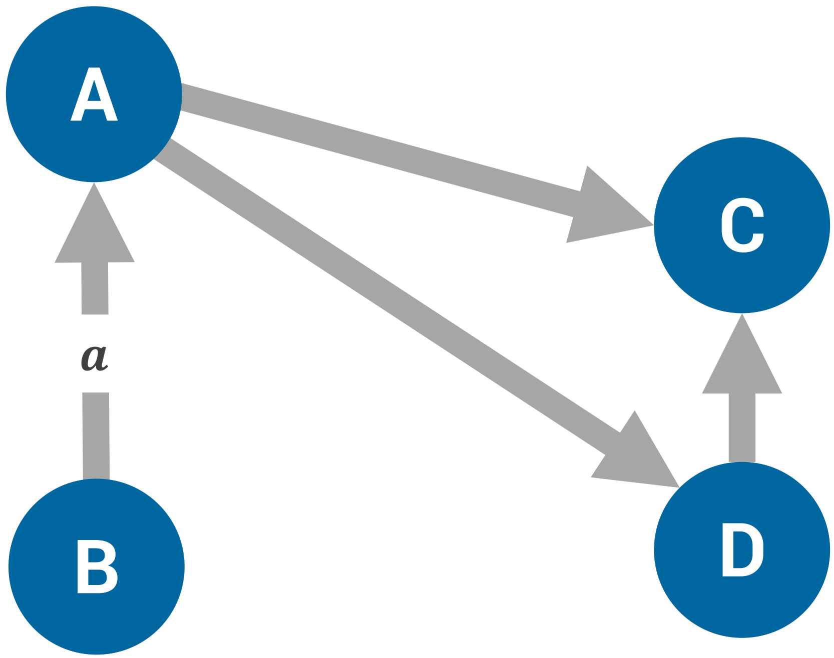
Individual Comparisons

Graph Database

- Selected databases:
    1. Neo4j
    2. OrientDB

Property Graphs

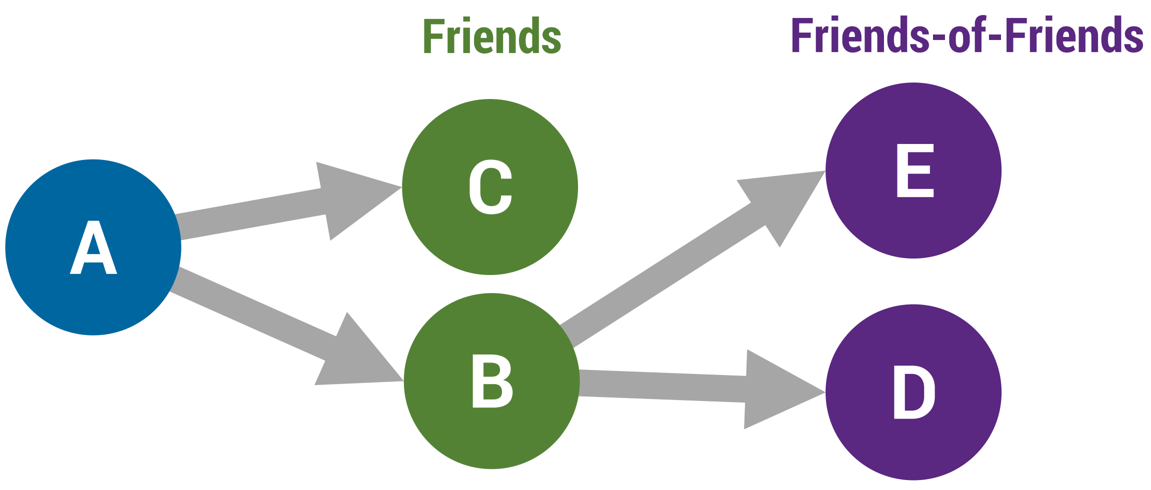
- A graph containing nodes and edges
- Nodes contain properties (key-value pairs)
- Relationships contain properties
- Relationships are named and directed, originating and terminating at a node



Individual Comparisons

Selected Queries

1. **Friend-of-friends**: obtain all the friends of a single person in the database, repeated for every person
2. **Get property**: obtain a single property from each node in the graph, similar to obtaining the name of each person in the database



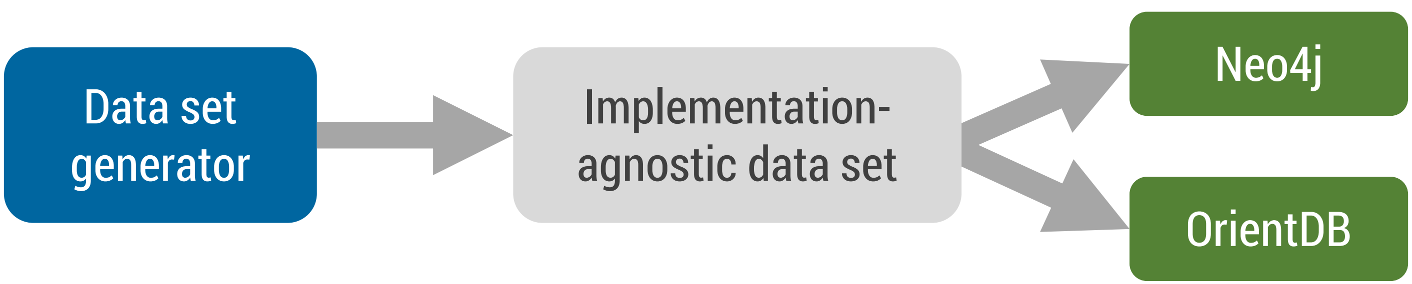
Individual Comparisons

Conceptual Data Sizes

Node count	Relationships
1,000	maximum 50 per node
10,000	maximum 50 per node
100,000	maximum 50 per node
1,000,000	maximum 50 per node

Actual Data Sizes

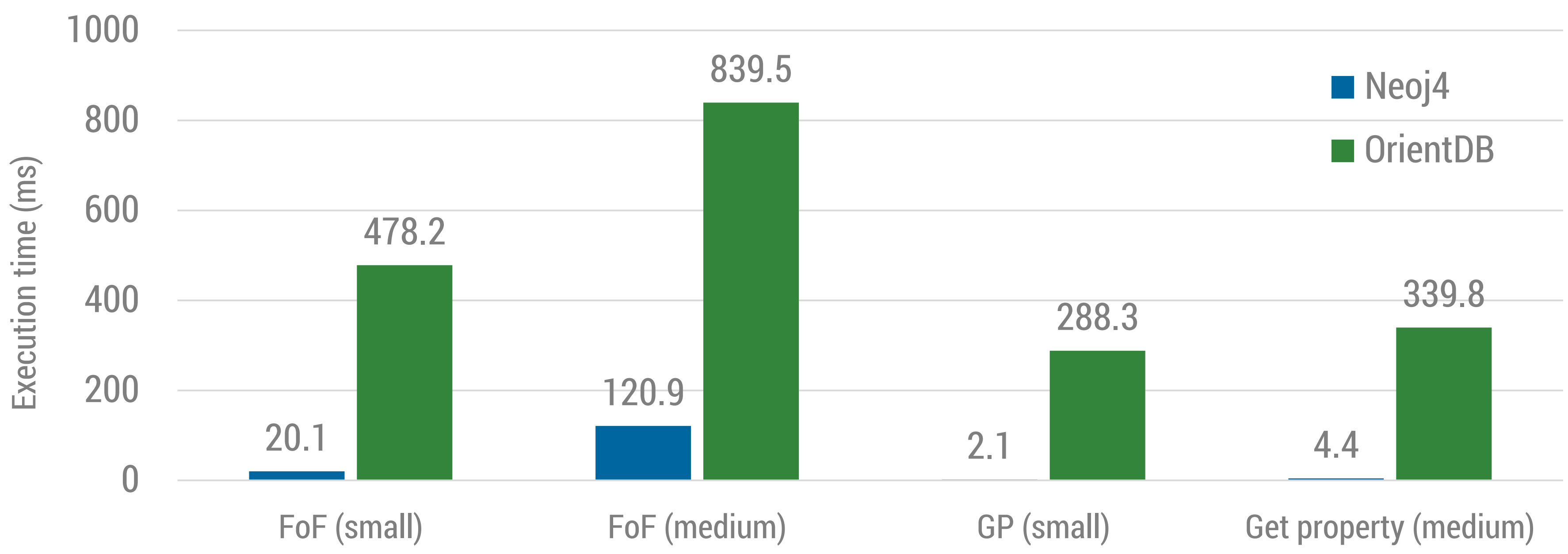
Node count	Relationships
1,000	10,009
10,000	100,486
100,000	1,001,176
1,000,000	10,015,575



Individual Comparisons

Results

- Neo4j was only able to a create a graph using the first two data sets
- OrientDB was only about to create a graph using the first three data sets



Individual Comparisons

Results

- All differences were found to be statistically significant to 95% confidence:
  - Pairwise comparisons were conducted

$$(c_1, c_2) = \bar{x} \mp z_{1-\frac{\alpha}{2}} \frac{s}{\sqrt{n}}$$

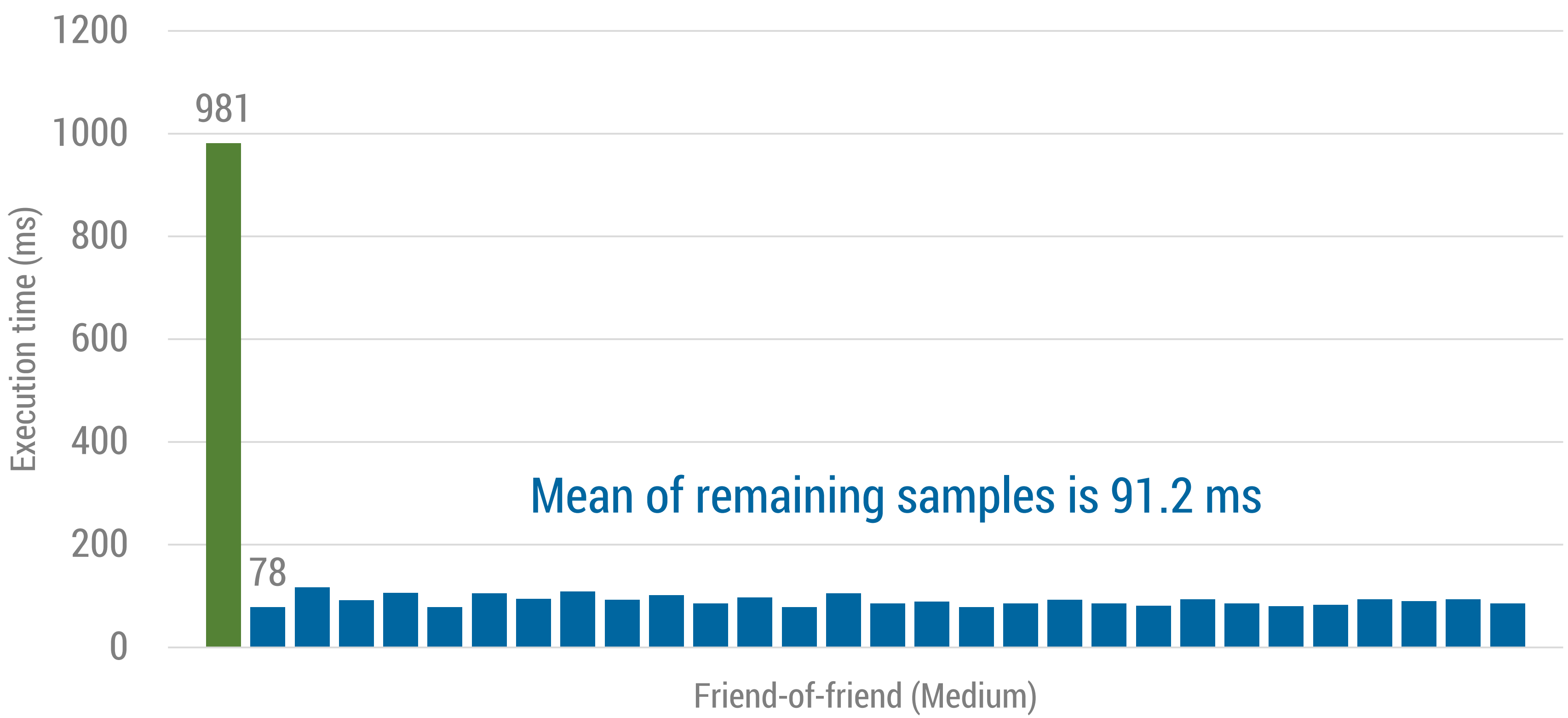
  - All resulting confidence intervals did not contain the value 0

Effects of Caching

- Neo4j clearly cached a large amount of data, which resulted in outliers
- These outliers were not discarded, since they were valid data

Individual Comparisons

Effects of Caching



Individual Comparisons

Conclusion

- Although Neo4j was not able to complete as many experiments as OrientDB, it is nonetheless the selected winner:
  - Although OrientDB scales better, the group comparison is in the range of small or medium data sets, thus this advantage is mute in the group comparison
  - In the pairwise experiments, Neo4j completely outperformed OrientDB

Selected: Neo4j

Individual Comparisons

NoSQL Database

- Non-relational structure
- Typically distributed system
- Scale extremely well
- Do not typically make ACID guarantees; instead focus on availability through BASE

Data Models

- Key-value Store
- Document Store
- Column Family
- Graph Database

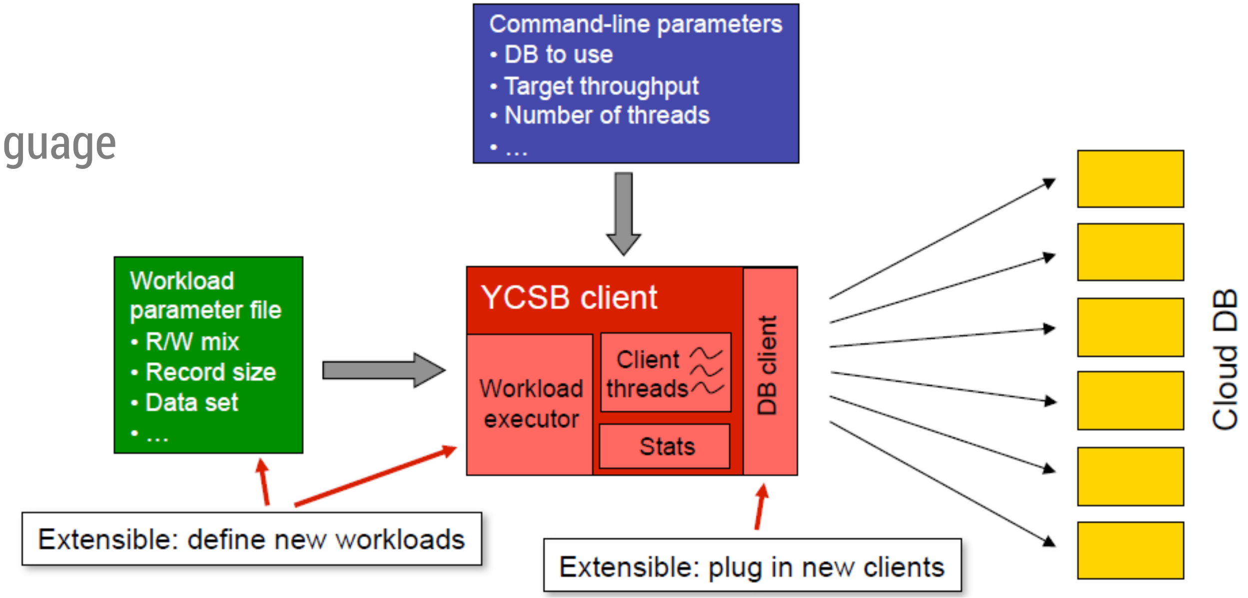
Selected Databases

1. Redis v2.8.19
2. MongoDB v2.4.9

Individual Comparisons

Yahoo! Cloud Serving Benchmark (YCSB) Tool

1. Automates development of consistent data sets and workloads
2. Widely accepted
3. Developed in Java language
4. Open-source
5. Highly extensible



Source: B.F. Cooper. “Yahoo! Cloud Serving Benchmark”. Yahoo! Inc. 2010. Web. <http://labs.yahoo.com/files/ycsb-v4.pdf>

Individual Comparisons

Selected Workloads

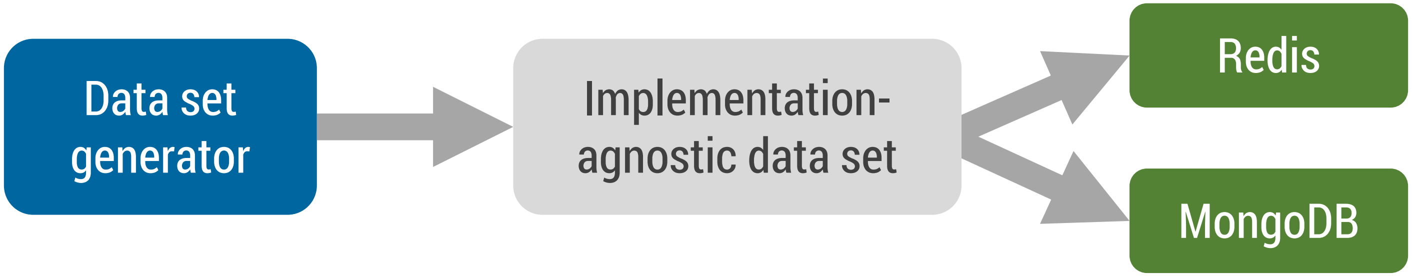
- Core YCSB Workloads A, B and F

Workload	Operations	Application Example
A – Update heavy	Read: 50% Update: 50%	Session store recording recent actions in a user session
B – Read heavy	Read: 95% Update: 5%	Photo tagging; add a tag is an update, but most operations are to read tags
F – Read-Modify-Write	Read: 50% Read-Modify-Write: 50%	User database, where user records are read and modified by the user or to record user activity.

Individual Comparisons

Data Size Tiers

Tier	Identifier	No. records	No. operations
1	1k	1,000	1,000
2	10k	10,000	1,000
3	100k	100,000	10,000
4	1M	1,000,000	100,000



Individual Comparisons

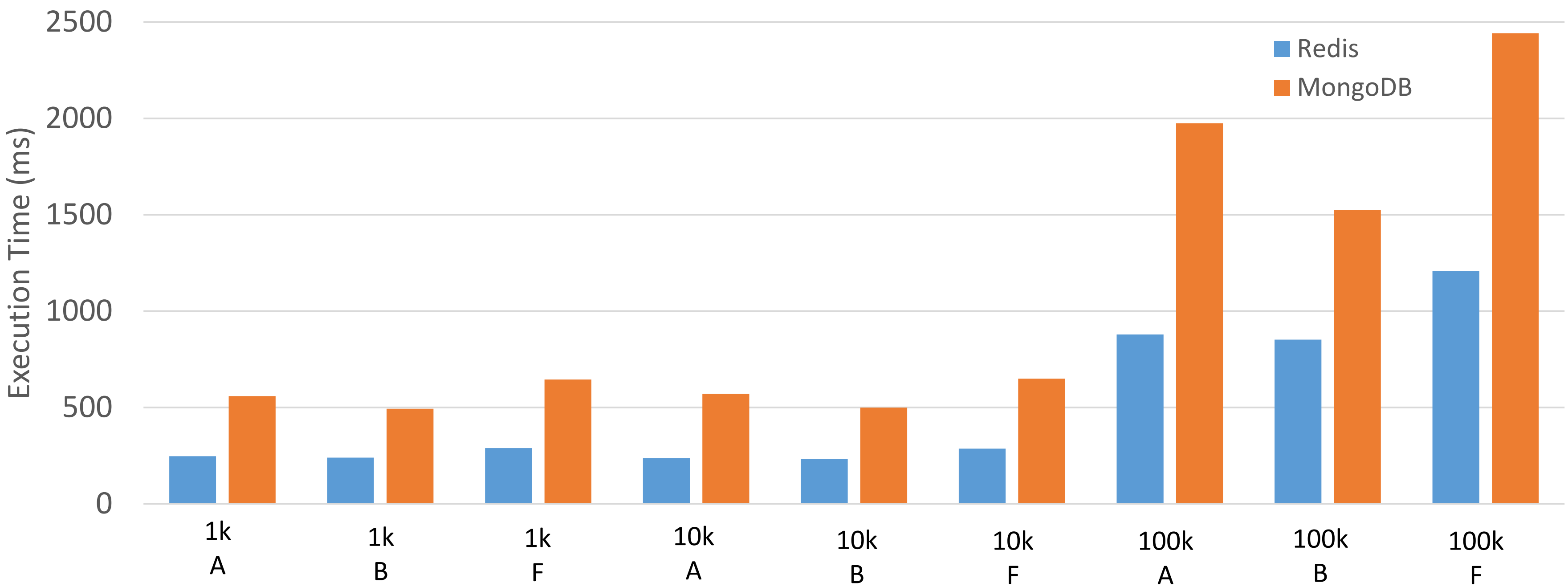
Experiment Design Summary

- Execution-time Performance Comparison
- Two Databases
- Three Workloads
- Four Data Size Tiers
- 30 Repetitions of each Workload

Individual Comparisons

Results

- Redis completed all workloads in approx. 50% of the time MongoDB required
- MongoDB could not load the largest data set size



Individual Comparisons

Results

- All differences were found to be statistically significant to 95% confidence:
  - Difference of means method for comparing alternatives was conducted:

$$s_x = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \qquad (c_1, c_2) = \bar{x} \mp z_{1-\frac{\alpha}{2}} s_x$$

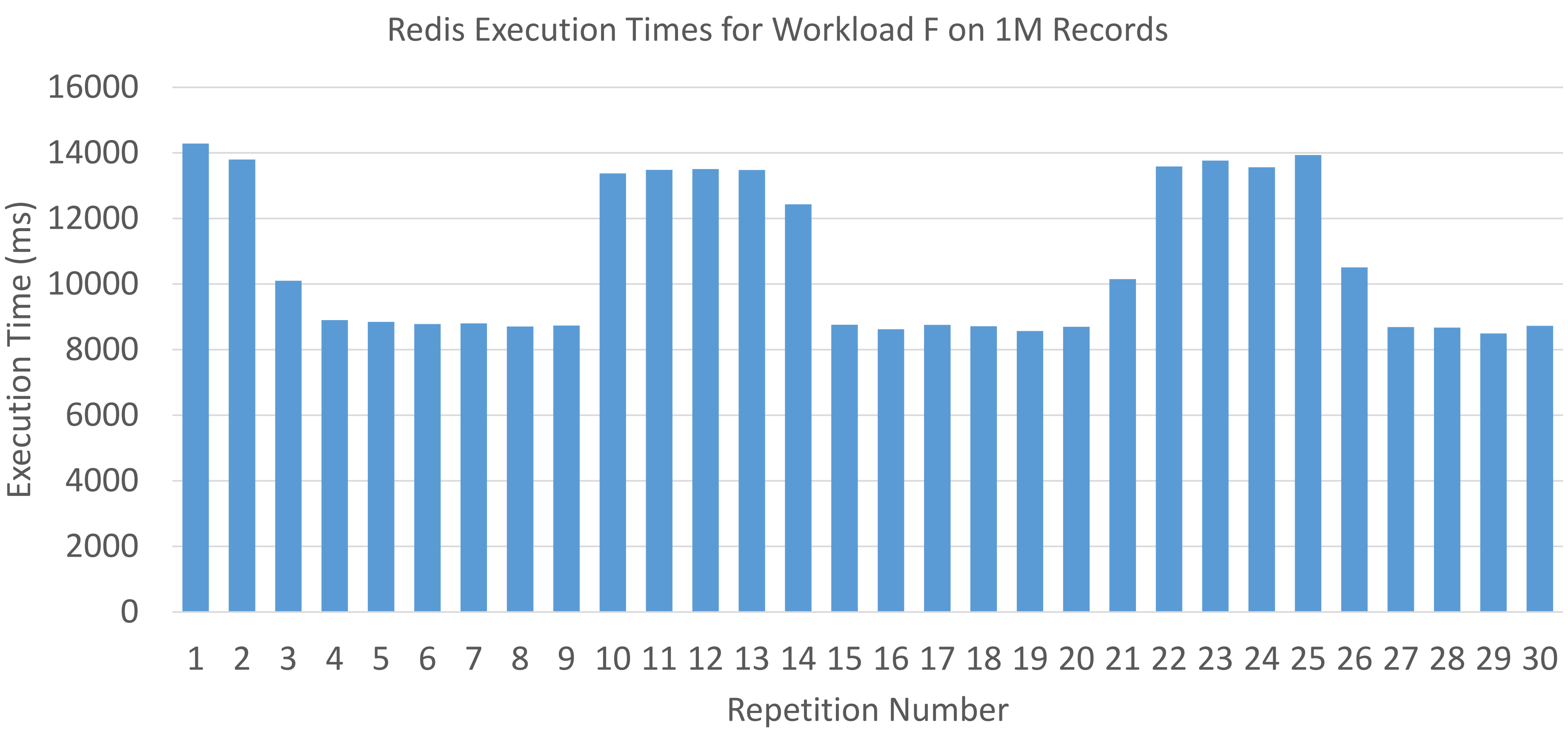
  - All resulting confidence intervals did not contain the value 0, thus no evidence to suggest that the difference is not statistically significant

Effect of Redis Backups to HDD

- For large quantities of data modification operations, Redis regularly backed up data to HDD, which affected execution time
- These outliers were not discarded, since they were valid data

Individual Comparisons

Effect of Redis Backups to HDD



Individual Comparisons

Conclusion

- Redis had a significantly shorter execution time for all database sizes and all workloads
- Redis could also handle larger quantities of data than MongoDB
  - Difference is statistically significant

Selected: Redis

Individual Comparisons

Relational Database

- Databases being compared
  - MySQL
  - PostgreSql

Tables

- Database contains multiple tables.
- Relating tables to one another with using primary and foreign keys.
- Each table contains multiple values (columns).
- Multiple rows in each table represent multiple entries.

Individual Comparisons

OrderTable	
Client_ID	Order_ID
varchar(200)	varchar(200) [P]

(a)

ClientTable		
Client_ID	Email	Address
varchar(200) [P]	varchar(200)	varchar(200)

(b)

ProductTable	
Description	Product_ID
varchar(200)	varchar(200) [P]

(c)

OrderProductTable			
Order_Number	Order_ID	Count	Product_D
varchar(200) [P]	varchar(200)	INT	varchar(200)

(d)

Individual Comparisons

Database Configurations

Small: 1,000 cumulative rows between all tables
Medium: 10,000 cumulative rows between all tables
Large: 100,000 cumulative rows between all tables
Extra-Large: 1,000,000 cumulative rows between all tables

	OrderTable	ClientTable	OrderProductTable	ProductTable	Total
Small	100	10	880	10	1,000
Medium	1,000	100	8,800	100	10,000
Large	10,000	1,000	88,000	1,000	100,000
Extra-Large	100,000	10,000	880,000	10,000	1,000,000

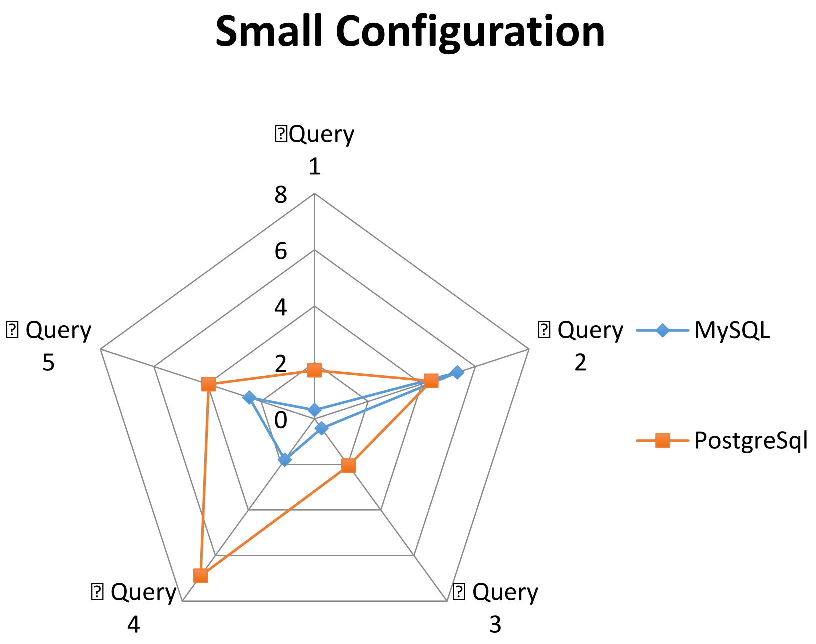
Individual Comparisons

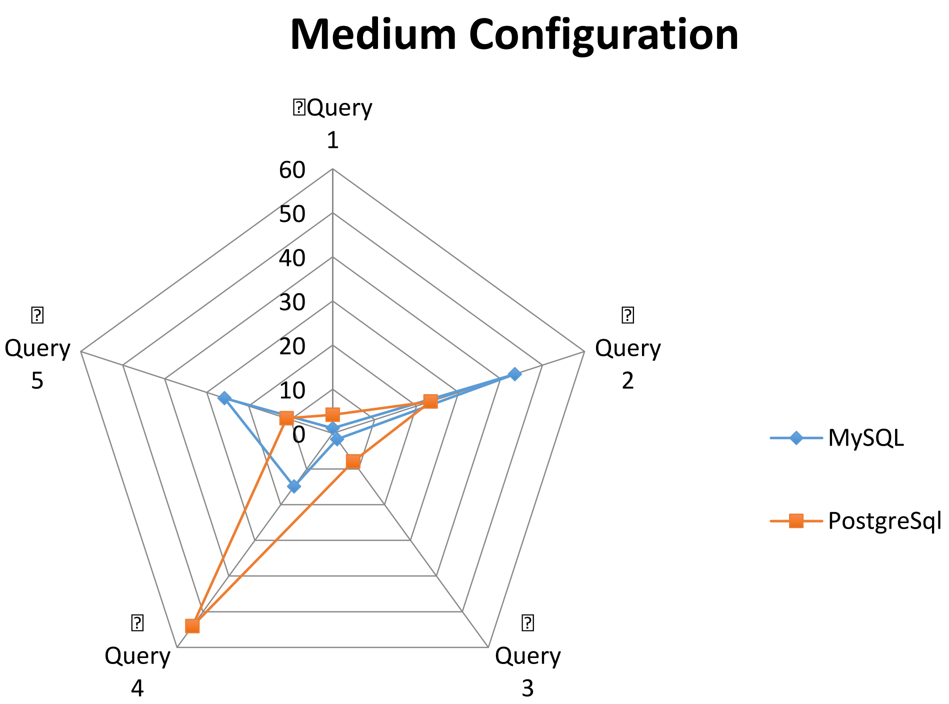
Queries

1. Select all information from OrderTable
2. Select all Orders and order them from higher to lowest count.
3. Find the number of orders per Client and order them most to least.
4. Update all orders where count is greater than 5 to have a count value of 5.
5. Nested selects to use all tables together in one large query. In the end will get the description of all items ordered from a certain email

Individual Comparisons

Results Per Configuration

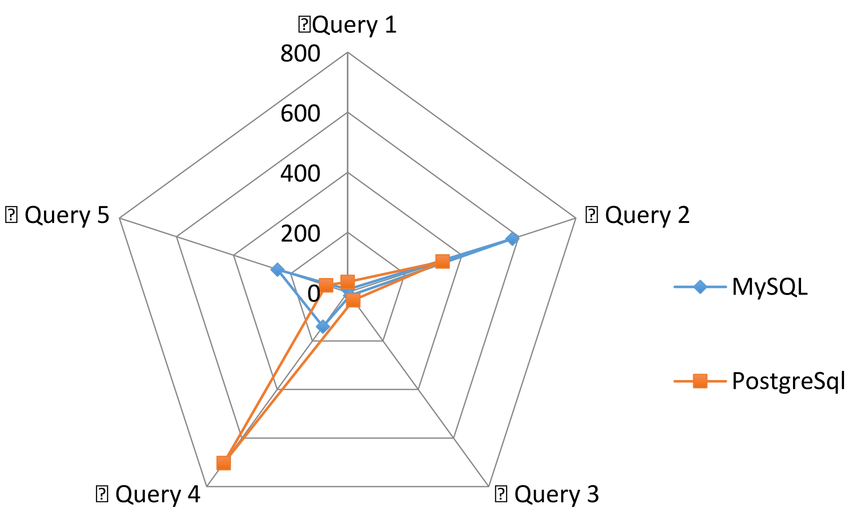




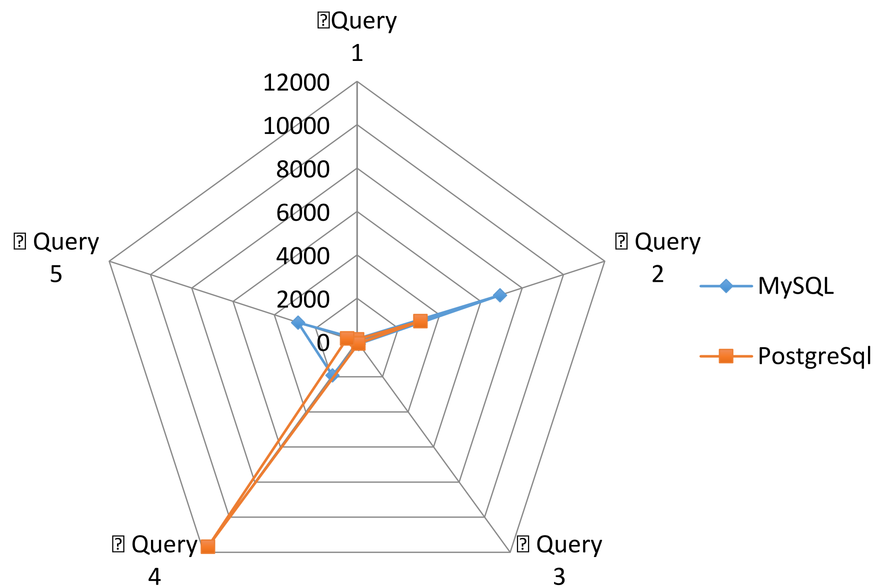
Individual Comparisons

Results Per Configuration

Large Configuration



Extra-Large Configuration



Individual Comparisons

Execution Times

Results (ms)		Query 1		Query 2		Query 3		Query 4		Query 5	
		Average	Std_dev	Average	Std_dev	Average	Std_dev	Average	Std_dev	Average	Std_dev
Small	MySQL	0.3111	0.0578	5.324	0.484	0.417	0.0869	1.8029	0.496	2.437	0.4869
	PostgreSql	1.7188	0.246	4.3544	2.962	2.057	0.648	6.8869	0.6693	3.956	0.418
Medium	MySQL	1.1638	0.143	43.385	3.741	1.687	4.827	14.884	4.8275	25.785	6.103
	PostgreSql	4.2004	1.050	23.3558	2.260	7.889	1.588	54.049	4.636	10.956	4.802
Large	MySQL	9.5801	0.96	577.222	37.485	13.331	4.95	140.835	15.914	245.69.2	28.411
	PostgreSql	34.676	18.66	332.849	21.491	32.316	15.481	702.753	108.56	75.779	16.489
Extra Large	MySQL	128.172	11.333	6919.29	188.94	129.734	16.63	1921.8	197.812	2855.03	586.81
	PostgreSql	109.229	16.19	3070.22	220.214	103.039	16.6.07	11682.4	463.33	478.169	50.6203

Individual Comparisons

Confidence Intervals

SMALL

	1	2	3	4	5
diff mean	-1.4077	0.9696	-1.6405333	-5.0840333	-1.5194
s1	0.05786138	0.48416829	0.08699061	0.49607253	0.48695474
s2	0.24640586	2.96288942	0.64832588	0.66935132	0.41818448
n	30	30	30	30	30
alpha	0.9	0.9	0.9	0.9	0.9
alpha/2	0.95	0.95	0.95	0.95	0.95
zscore	1.64	1.64	1.64	1.64	1.64
sx	0.04621103	0.54812203	0.11942833	0.15210952	0.11718976
c1	-1.4837104	0.0680195	-1.8369755	-5.3342312	-1.71216
c2	-1.3316896	1.8711805	-1.4440912	-4.8338354	-1.32664
90% confident?	MySQL	PostgreSql	MySQL	MySQL	MySQL

(a)

MEDIUM

	1	2	3	4	5
diff mean	-3.0365667	20.0293	-6.2017667	-39.164533	14.8286667
s1	0.14367457	3.74162801	0.33955193	4.82753966	6.10397871
s2	1.05066035	2.26021275	1.58804054	4.63669502	4.89024493
n	30	30	30	30	30
alpha	0.9	0.9	0.9	0.9	0.9
alpha/2	0.95	0.95	0.95	0.95	0.95
zscore	1.64	1.64	1.64	1.64	1.64
sx	0.19360867	0.79808817	0.2964888	1.22207583	1.42797119
c1	-3.3550246	18.7165618	-6.6894473	-41.174669	12.4798631
c2	-2.7181087	21.3420382	-5.714086	-37.154397	17.1774703
90% confident?	MySQL	PostgreSql	MySQL	MySQL	PostgreSql

(b)

LARGE

	1	2	3	4	5
diff mean	-25.0962	244.3724	-18.9849	-561.9179	169.912933
s1	0.96001755	37.4858281	4.95006133	15.9141059	28.4114854
s2	18.6643443	21.4918064	15.4819725	108.565214	16.4890806
n	30	30	30	30	30
alpha	0.9	0.9	0.9	0.9	0.9
alpha/2	0.95	0.95	0.95	0.95	0.95
zscore	1.64	1.64	1.64	1.64	1.64
sx	3.41213219	7.88899032	2.9675724	20.0330265	5.99750582
c1	-30.708658	231.396166	-23.866122	-594.8693	160.047914
c2	-19.483742	257.348634	-14.103678	-528.9665	179.777953
90% confident?	MySQL	PostgreSql	MySQL	MySQL	PostgreSql

(c)

EXTRA LARGE

	1	2	3	4	5
diff mean	18.9420333	3849.0727	26.6946667	-9761.1279	2376.86337
s1	11.3338099	188.948944	16.6309695	197.81299	586.810748
s2	16.1901422	220.214926	16.6073732	463.333366	50.6203151
n	30	30	30	30	30
alpha	0.9	0.9	0.9	0.9	0.9
alpha/2	0.95	0.95	0.95	0.95	0.95
zscore	1.64	1.64	1.64	1.64	1.64
sx	3.60821263	52.9768242	4.29105267	91.9796693	107.534378
c1	13.0070517	3761.93358	19.6365131	-9912.421	2199.98506
c2	24.877015	3936.21182	33.7528202	-9609.8348	2553.74168
90% confident?	PostgreSql	PostgreSql	PostgreSql	MySQL	PostgreSql

(d)

Individual Comparisons

Regression Analysis (Query 5)

Linear Regression

Regression Statistics

R	0.99991
R Square	0.99982
Adjusted R Square	0.99973
S	22.81337
Total number of observations	4

A =- 13.8889 + 0.0029 * B

ANOVA

	d.f.	*SS*	*MS*	*F*	*p-level*
Regression	1.	5,763,630.11817	5,763,630.11817	11,074.32317	0.00009
Residual	2.	1,040.89975	520.44988		
Total	3.	5,764,671.01792			

	Coefficients	*Standard Error*	*LCL*	*UCL*	*t Stat*	*p-level*	*H0 (5%) rejected?*
Intercept	-13.8889	13.68742	-72.78112	45.00331	-1.01472	0.41703	No
B	0.00287	0.00003	0.00275	0.00298	105.23461	0.00009	Yes
T (5%)	4.30265						

LCL - Lower value of a reliable interval (LCL)
UCL - Upper value of a reliable interval (UCL)

Residuals

Observation	*Predicted Y*	*Residual*	*Standard Residuals*
1	-11.02256	13.46009	0.72261
2	14.7745	11.01056	0.59111
3	272.74516	-27.05232	-1.45231
4	2,852.45167	2.58167	0.1386

Linear Regression

Regression Statistics

R	0.99871
R Square	0.99742
Adjusted R Square	0.99613
S	14.0777
Total number of observations	4

A = 12.4078 + 0.0005 * B

ANOVA

	d.f.	*SS*	*MS*	*F*	*p-level*
Regression	1.	153,227.11194	153,227.11194	773.16505	0.00129
Residual	2.	396.36327	198.18163		
Total	3.	153,623.47521			

	Coefficients	*Standard Error*	*LCL*	*UCL*	*t Stat*	*p-level*	*H0 (5%) rejected?*
Intercept	12.40775	8.44625	-23.93352	48.74903	1.46903	0.27958	No
B	0.00047	0.00002	0.0004	0.00054	27.80585	0.00129	Yes
T (5%)	4.30265						

LCL - Lower value of a reliable interval (LCL)
UCL - Upper value of a reliable interval (UCL)

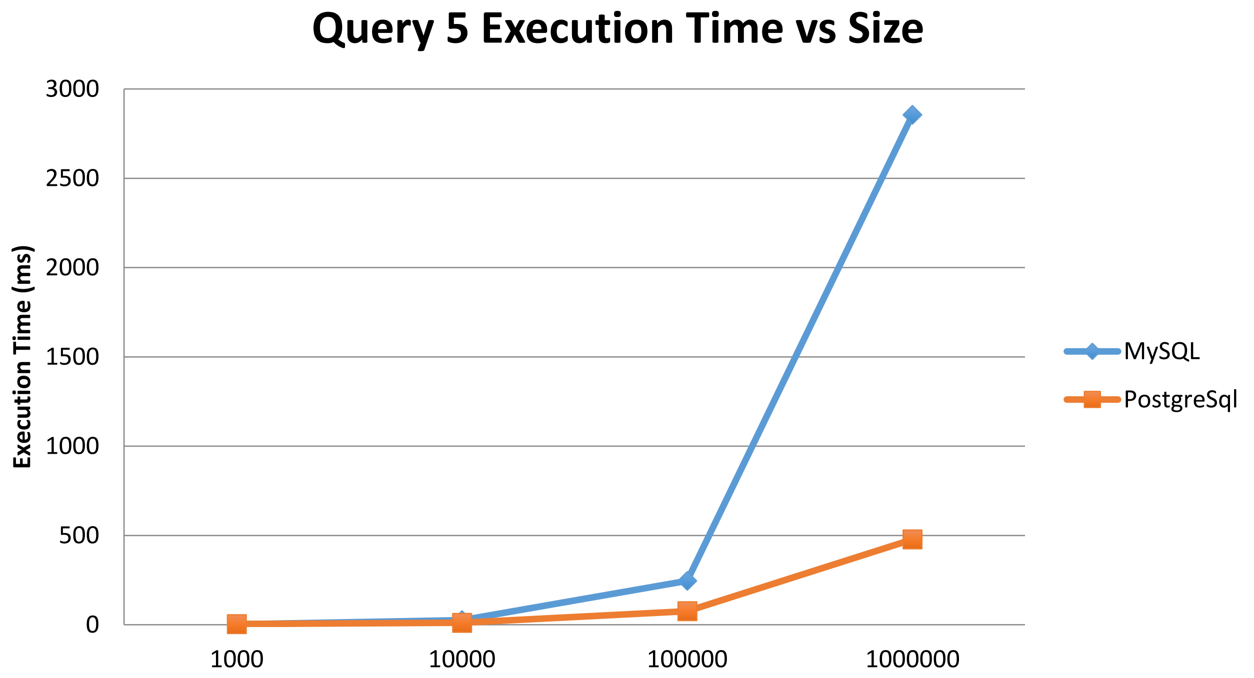
Residuals

Observation	*Predicted Y*	*Residual*	*Standard Residuals*
1	12.87511	-8.91818	-0.77587
2	17.08131	-6.12491	-0.53286
3	59.14332	16.63658	1.44736
4	479.76346	-1.59349	-0.13863

$$-13.8889 + 0.0029x = 12.4078 + 0.0005x$$
$$.0024x = 26.2967$$
$$x = 10{,}956.9 \Rightarrow 10{,}957$$

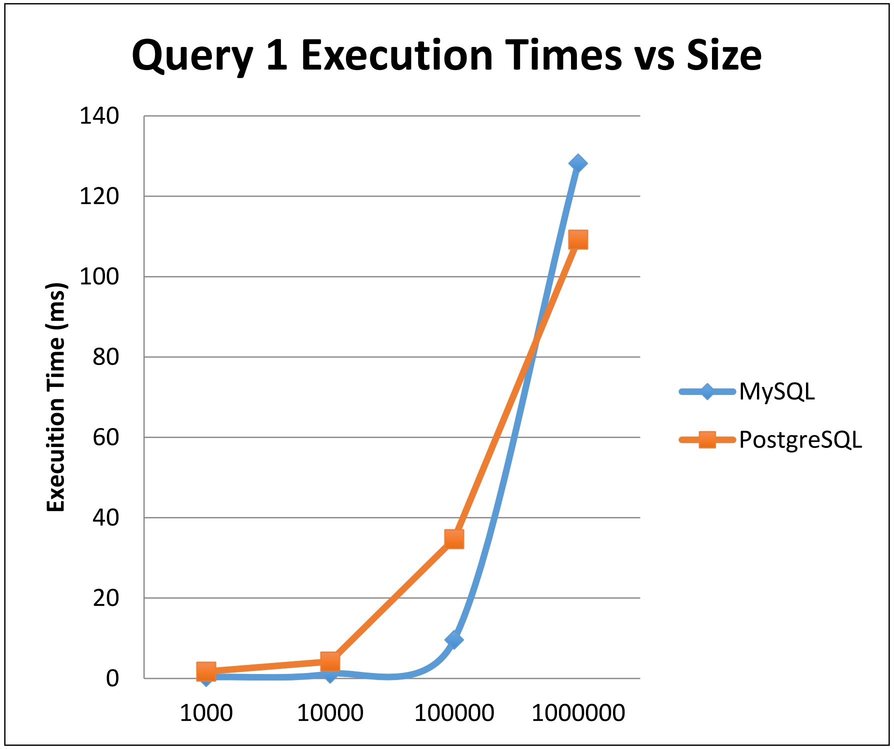
Individual Comparisons

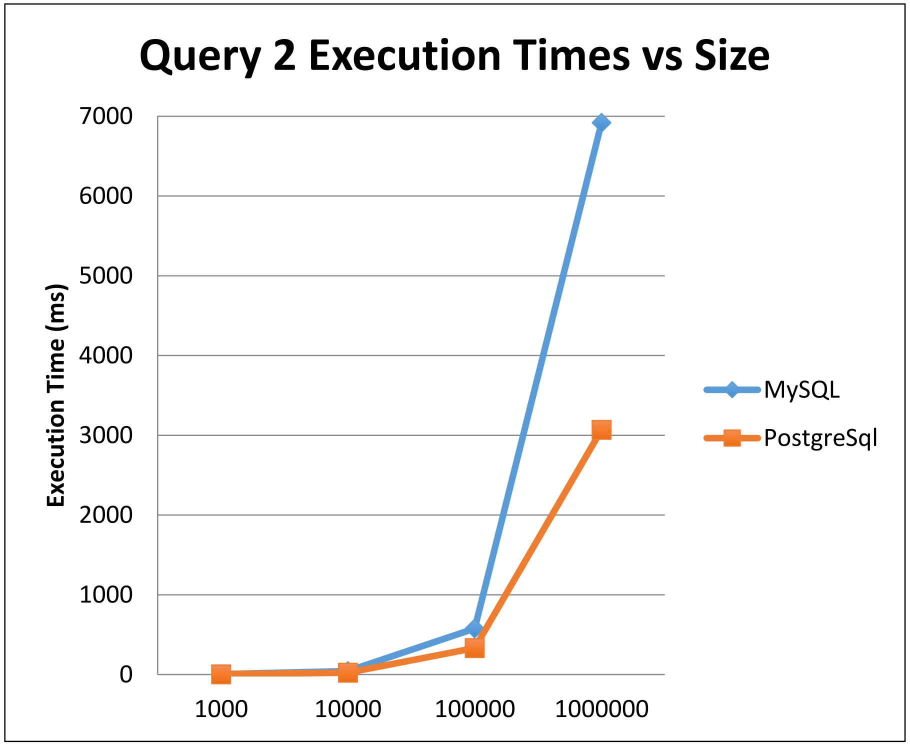
Regression Analysis



Individual Comparisons

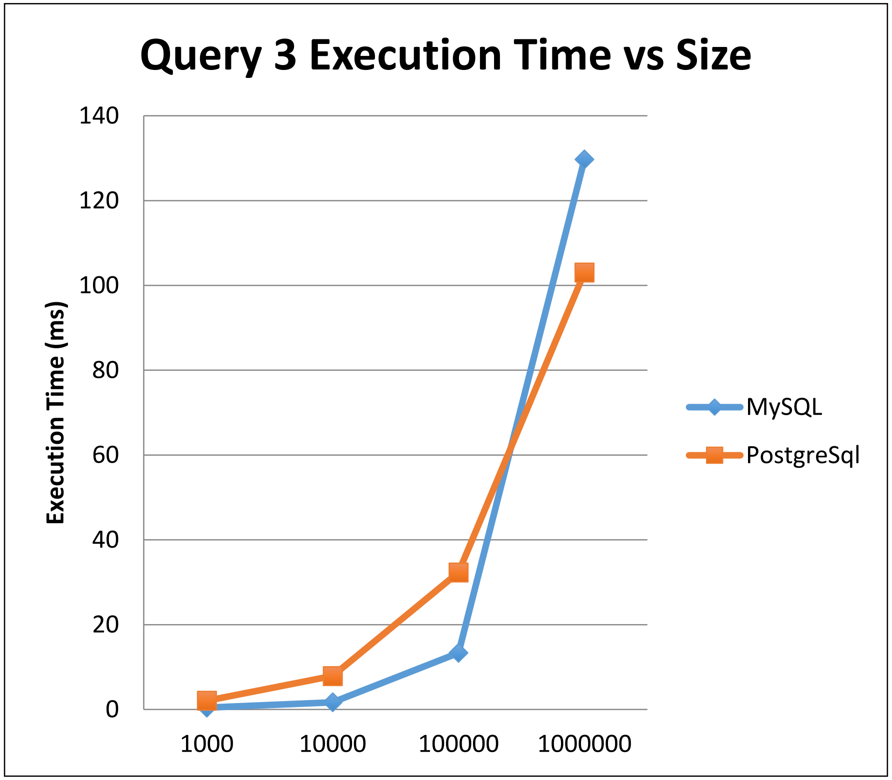
Execution Time Comparisons

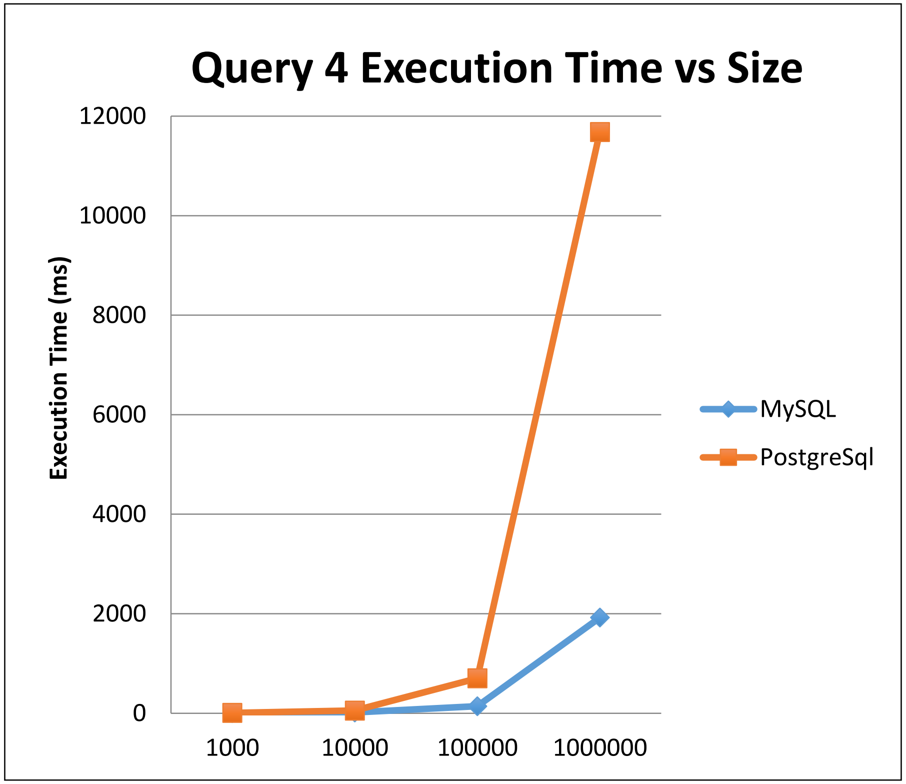




Individual Comparisons

Execution Time Comparisons





Individual Comparisons

Conclusion

- MySQL and PostgreSql were both able to perform the queries on all four sizes of the database
  - MySQL was significantly better (90%) at the lower sizes
  - PostgreSql started to become more efficient after about 100,000 rows depending on the query
  - Small dataset for final comparison favored MySQL.

Selected: MySQL

Group Comparisons

Group of Databases

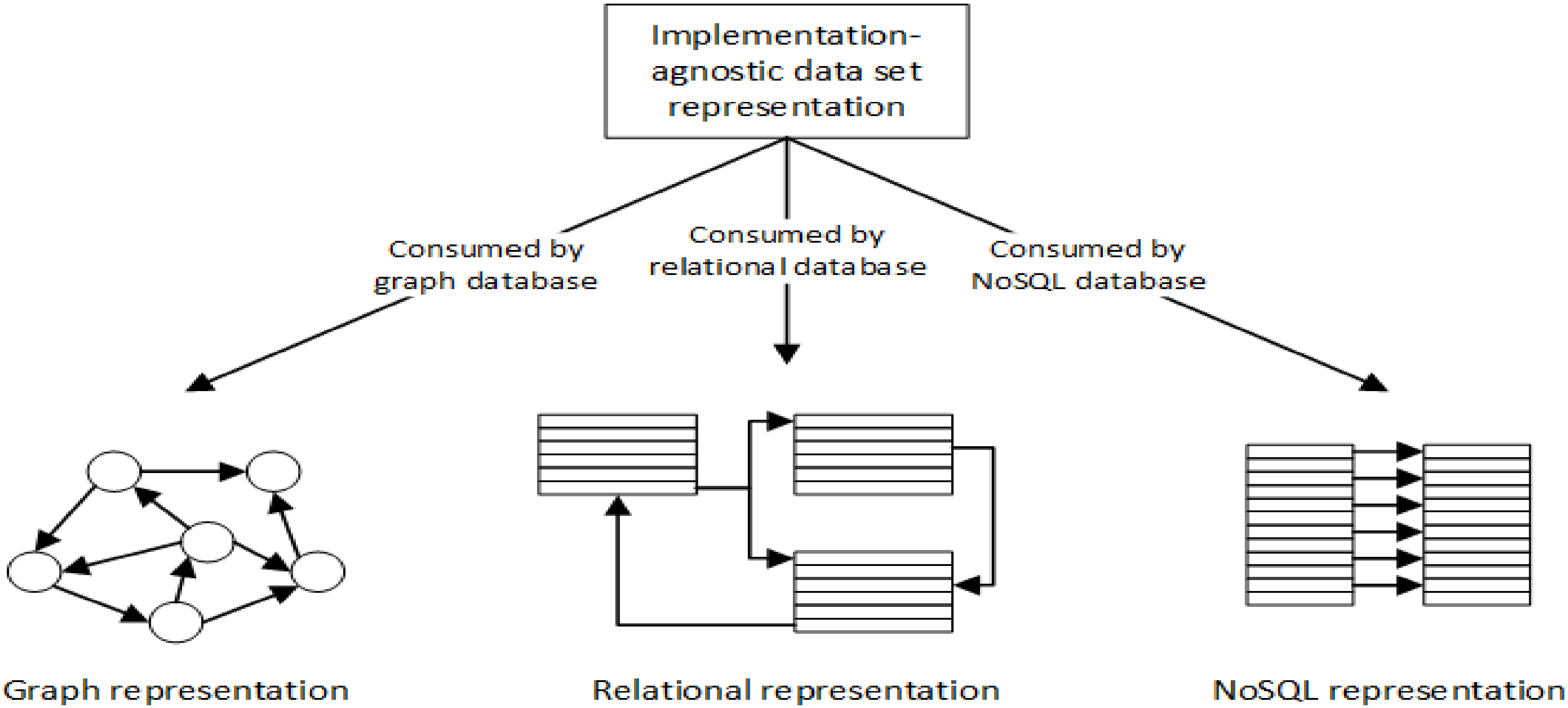
- Databases being compared
  - MySQL
  - Neo4j
  - Redis

Group Comparisons

Group Dataset

- Common dataset
  - 1,000 people
  - 1,000 interests to choose from
  - 100 places of work
  - Suggestions of coworkers
  - Each person has 10 interests
  - Each person has worked at 5 jobs

Group Comparisons



Group Comparisons

Database Layouts

- MySQL
  - Use of table structure. People, Interest, Work, Employee, and Relationship Tables.
- Redis
  - Use of key-value pairs. Lists of data saved for each persons workplace and interest and list of workers saved for each workplace.
  - Slight repetition of data.
- Neo4j
  - Use of nodes and relationships. People and Interests are nodes. Relationships link people to other people and people to interests

Group Comparisons

Group Queries

1. Get all interests of a person
    - Simple NoSQL query to simply get all of a persons interests.
2. Recommend interests based on interests of coworkers
    - Graphing Query that must relate coworkers and interest of those coworkers together.
3. Obtain all coworkers and common place of work
    - Relational Query that returns the workplace of an employee along with all people who worked at that workplace.

Group Comparisons

Average Execution Times

Group Comparisons

Conclusion

- NoSQLQuery
  - Graph database is significantly faster than NoSQl and SQL. NoSQL is significantly faster than SQL. All done at 95% confidence
- Graph Query
  - Graph database is significantly faster than NoSQl and SQL. Both done at 95% confidence
- Relational Query
  - Graph database is significantly faster than NoSQl and SQL. NoSQL is significantly faster than SQL. All done at 95% confidence.

Selected: Neo4j

Alternative Methods

Analytical Modeling

- For example, graph operations can be divided into micro-, micro-, and algorithmic operations, and the execution of the higher-level operations are a function of the execution of lower-level operations
  - Reading a vertex, edge, or property: R
  - Get all neighbors via incoming, outgoing, both edges: $2nR, n \equiv count_{neighbors}$

Simulation

- For transactional databases, the transactions of write, update, etc. can be simulated
  - Test how well a database will operate in the deployment environment
  - Provide a base-line for trade-off analysis between transactional databases

Final Conclusions

- Databases are essential components of many software applications
- Multitudes of database types and end products exist
- Performance analysis is important and required to determine an appropriate application solution
- Graph, NoSQL and relational database types have been analyzed and compared both in isolation and in a final combined comparison through statistical analysis and empirical experimentation
- Neo4j (graph), MySQL (SQL) and Redis (NoSQL) products were individual “best-in-class” performers
- Neo4j (graph) consistently outperformed MySQL and Redis for read-centric queries executed in the combined performance comparison; Redis consistently came in second place
- Alternative methods of performance analysis are possible but were not explored