Beyond Macrobenchmarks: Microbenchmark-based Graph Database Evaluation

Matteo Lissandrini, Martin Brugnara, Yannis Velegrakis

UNIVERSITY OF TRENTO

Universiteit Utrecht

Graph management systems are a relatively new technology, their features, performances, and capabilities are **not yet fully understood neither agreed upon. What solution works best?**

THERE IS NO SILVER BULLET:

Different Data Characteristics
Different Query Types
Different Use-cases

Different Query Processing Strategies
Different Indexing/Optimizations
Different Data Organization

GOAL: UNDERSTAND GRAPH DATABASES PERFORMANCE

1. Impacting Factors
- System Architecture
- Query Workload
- Data Characteristics

2. Desired Outcome
- Evaluate Pros/Cons of each design decision
- Identify cause of **underperforming operations**

BENCHMARK METHODOLOGIES

1. Macro-benchmark
- **Domain-dependent** queries
- Test oriented towards **complex and composite operations**
- **Hides performance** of specific operators focusing on global optimizations

2. Micro-benchmark *Our Proposal*
- Applicable **to different domains**
- Identifies **ubiquitous operators** common in different use-cases
- Allows to isolate the cause of **underperforming queries**

FEATURES & ADVANTAGES

- **Richest** set of queries
- **Open source** platform that is easily **extensible**
- **Multi-domain datasets** included
- **Gremlin based:** portable to new systems

Questions?
Look for the authors →


Matteo


Martin


Yannis

A **Micro-Benchmark**

for an in-depth understanding of

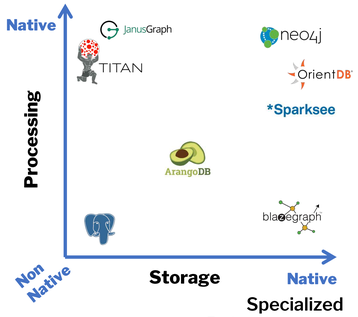
Graph Databases Performance

1. Highlights **advantages** of native systems in **large traversals**
2. Observes **fast insertions** but high **variability in delete/update** operations execution time
3. Shows that **relational** systems **struggle** with high **number of edge labels**, while they are fastest in index-based node search
4. Reveals overall **scalability problems** with large intermediate results

"Beyond Macrobenchmarks: Microbenchmark-based Graph Database Evaluation"
M. Lissandrini, M. Brugnara, Y. Velegrakis – *PVLDB'18*
Paper & code: https://graphbenchmark.com

SYSTEM LANDSCAPE

Specialized Query-processing &Algorithms



Specialized Data-structures & Indexes

QUERIES

#	Query	Description	Cat
1.	g.loadGraphSON("/path")	Load dataset into the graph 'g'	L
2.	g.addVertex(p[])	Create new node with properties p	C
3.	g.addEdge(v1 , v2 , l)	Add edge *l* from *v1* to *v2*	
4.	g.addEdge(v1 , v2 , l , p[])	*Same as Q.3*, but with properties *p*	
5.	v.setProperty(Name, Value)	Add property *Name*=*Value* to node *v*	
6.	e.setProperty(Name, Value)	Add property *Name*=*Value* to edge *e*	
7.	g.addVertex(...); g.addEdge(...)	Add a new node, and then edges to it	
8.	g.V.count()	Total number of nodes	R
9.	g.E.count()	Total number of edges	
10.	g.E.label.dedup()	Existing edge labels (no duplicates)	
11.	g.V.has(Name, Value)	Nodes with property *Name*=*Value*	
12.	g.E.has(Name, Value)	Edges with property *Name*=*Value*	
13.	g.E.has('label',l)	Edges with label *l*	
14.	g.V(id)	The node with identifier *id*	
15.	g.E(id)	The edge with identifier *id*	
16.	v.setProperty(Name, Value)	Update property *Name* for vertex *v*	U
17.	e.setProperty(Name, Value)	Update property *Name* for edge *e*	
18.	g.removeVertex(id)	Delete node identified by *id*	D
19.	g.removeEdge(id)	Delete edge identified by *id*	
20.	v.removeProperty(Name)	Remove node property *Name* from *v*	
21.	e.removeProperty(Name)	Remove edge property *Name* from *e*	
22.	v.in()	Nodes adjacent to *v* via incoming edges	T
23.	v.out()	Nodes adjacent to *v* via outgoing edges	
24.	v.both('l')	Nodes adjacent to *v* via edges labeled *l*	
25.	v.inE.label.dedup()	Labels of in coming edges of *v* (no dupl.)	
26.	v.outE.label.dedup()	Labels of outgoing edges of *V* (no dupl.)	
27.	v.bothE.label.dedup()	Labels of edges of *v* (no dupl.)	
28.	g.V.filter{it.inE.count()>=k}	Nodes of at least k-incoming-degree	
29.	g.V.filter{it.outE.count()>=k}	Nodes of at least k-outgoing-degree	
30.	g.V.filter{it.bothE.count()>=k}	Nodes of at least k-degree	
31.	g.V.out.dedup()	Nodes having an *incoming* edge	
32.	v.as('i').both().except(vs) .store(j).loop('i')	Nodes reached via breadth-First traversal from *v*	
33.	v.as('i').both(*ls).except(j) .store(vs).loop('i')	Nodes reached via breadth-First traversal from *v* on labels *ls*	
34.	v1.as('i').both().except(j).store(j) .loop('i'){it.object.equals(v2)} .retain([v2]).path()	Unweighted Shortest Path from *v1* to *v2*	
35.	*Shortest Path on 'l'*	*Same as Q.34*, but only following label *l*	

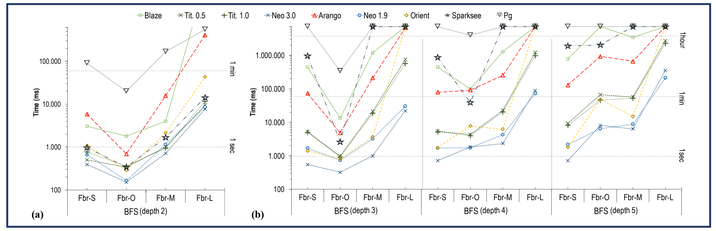
[] denotes a Hash Map; *g* is the graph; *v* and *e* are node/edges.

DATASETS

	|V|	|E|	|L|	Connected Component #	Connected Component Maxim	Density	Modularity	Degree Avg	Degree Max	Δ
Yeast	2.3K	7.1K	167	101	2.2K	$1.34*10^{-3}$	$3.66*10^{-2}$	6.1	66	11
MiCo	100K	1.1M	106	1.3K	93K	$1.10*10^{-5}$	$5.45*10^{-1}$	21.6	1.3K	23
Pok-O	1.9M	4.3M	424	133K	1.6M	$1.19*10^{-6}$	$9.82*10^{-1}$	4.3	92K	48
Pok-S	0.5M	0.3M	1814	0.16M	20K	$1.20*10^{-5}$	$9.91*10^{-1}$	1.3	13K	4
Pok-M	4M	3.1M	2912	1.1M	1.4M	$1.94*10^{-7}$	$7.97*10^{-1}$	1.5	139K	37
Pok-L	28.4M	31.2M	3821	2M	23M	$3.87*10^{-8}$	$2.12*10^{-1}$	2.2	1.4M	33
ldbc	184K	1.5M	15	1	184K	$4.43*10^{-5}$	0	16.6	48K	10

TRAVERSALS

Native systems with **JOIN-free adjacency** provide the best scalability **for generic traversals.**



GLOBAL READ & SEARCH

Depending on **the nature of the query** some systems perform best than others: e.g., **relational systems** perform best in high selectivity queries for attributes.