GRAPHIUM: Visualizing Performance of Graph and RDF Engines on Linked Data

Alejandro Flores, Guillermo Palma, Maria-Esther Vidal, Domingo De Abreu, Valeria Pestana, José Piñero, Jonathan Queipo, José Sánchez

Universidad Simón Bolívar, Caracas, Venezuela
{aflores,gpalma,mvidal,dabreu,vpestana,jpinero,jqueipo,jsanchez}@ldc.usb.ve

Abstract. We present GRAPHIUM a tool to visualize trends and patterns in the performance of existing graph and RDF engines. We will demonstrate GRAPHIUM and attendees will be able to observe and analyze the performance exhibited by Neo4j, DEX, HypergraphDB and RDF-3x when core graph-based and mining tasks are run against a variety of benchmarks of graphs of diverse characteristics.

1 Introduction

Graphs are commonly used to represent linked data, and several efficient algorithms have been proposed to consume and mine graphs. For example, Saha et al. [8] and Thor et al. [9] have defined densest subgraphs and graph summarization techniques to mine linked datasets and identify patterns between concepts and links. Further, algorithms for pattern matching, graph traversal, and graph reachability have been extensively studied in the literature [1]. The majority of these algorithms are computationally complex, and rely on main-memory structures to efficiently solve core graph tasks. Additionally, different engines have been developed to manage, store and query graph databases (e.g., Neo4j [7], DEX [4], HypergraphDB [2], RDF-3x [6]). Each graph database engine implements particular structures and usually relies on indices to speed up execution time; additionally, some engines make available APIs comprised of methods to solve core graph-based tasks. Although existing graph and RDF engines could be used to store linked data, mined and consumed by existing graph algorithms, there is no clear understanding of how these algorithms may behave on these engines. We present GRAPHIUM a visualization tool that exploits different graphical representations to report on the results of evaluating Neo4j, DEX, HypergraphDB and RDF-3x on a variety of benchmarks of graphs and graph-based tasks. Visualization techniques used in GRAPHIUM facilitate the understanding of trends and patterns between the performance exhibited by these engines during the execution of tasks of reachability, traversal, adjacency, pattern matching, densest graph, and graph summarization on a variety of graphs of different density and size. During the demonstration attendees will go through the visualization of different patterns that will allow them to uncover the properties and limitations of existing graph engines, as well as to reach conclusions about which engine is more appropriate for a given task. Demo is available at `http://graphium.ldc.usb.ve/demo/`.

2 The GRAPHIUM architecture

GRAPHIUM is built on top of a catalog that keeps experimental results collected during the evaluation of existing graph database and RDF engines against a variety of

benchmarks. GRAPHIUM exploits visualization services implemented by the D3.js JavaScript library[1]. Figure 1 shows GRAPHIUM GUI. In the area enclosed in red rectangle number 1, a user can select to analyze: *i*) a particular graph, e.g., the dense graph DSJC1000.9; *ii*) an engine, e.g., Neo4j, DEX; and *iii*) a particular task, e.g., reachability. Results are visualized in the area enclosed by the blue rectangle number 2; GRAPHIUM exploits visualization capabilities of the Parallel Coordinates[2] to illustrate patterns and trends in the performance of each engine.

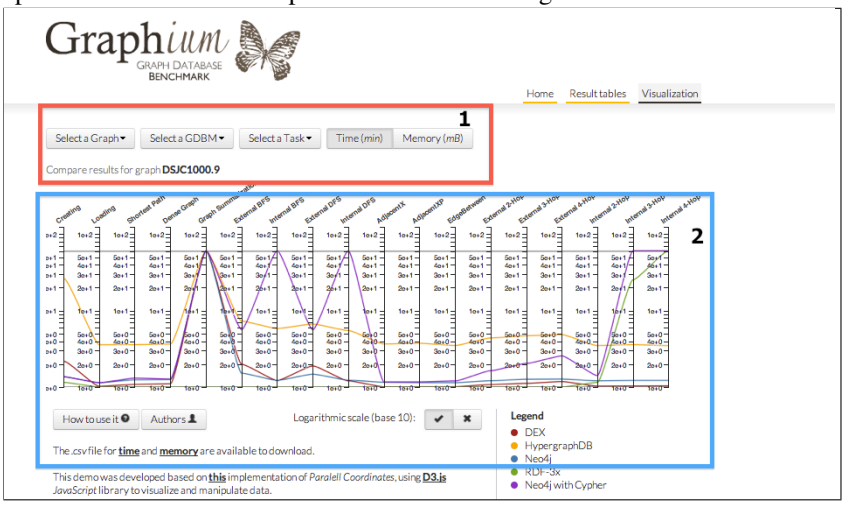

Fig. 1. The GRAPHIUM GUI for Neo4j, DEX, HypergraphDB and RDF-3x. 1-Selection Area: Graphs, GDBMs, Tasks, and Metrics can be selected. 2-Visualization Area: visualization ranges and scales can be chosen; explanation of how create and remove visualization ranges is presented.

3 Demonstration of Use Cases

We consider a benchmark of six graphs: DSJC1000.1, DSJC1000.5, DSJC1000.9, USA-road-d.NY, USA-road-d.FLA, and Berlin10M. The family of DSJC1000.X graphs were randomly generated using the techniques proposed by Johnson et al. [3] as instances to solve the graph coloring problem[3]; all these graphs have 1,000 nodes, and the graph density varies from 0.1 to 0.9. Instances of USA-road-d.NY and USA-road-d.FLA correspond to the New York City and Florida State road networks that were part of the 9th DIMACS Implementation Challenge - Shortest Paths[4]. Finally, Berlin10M was generated with The Berlin SPARQL Benchmark[5]. The goal of the demonstration is to visualize trends and patterns that can be found in the performance of Neo4j, DEX, HypergraphDB, and RDF-3x, where performance is measured in terms of execution time (elapsed time in msecs.), main-memory required to execute the graph-task (measured in KB), and secondary-memory needed to store the internal representation of

[1] http://d3js.org/
[2] http://mbostock.github.io/d3/talk/20111116/iris-parallel.html
[3] https://sites.google.com/site/graphcoloring/vertex-coloring
[4] http://www.dis.uniroma1.it/challenge9/download.shtml
[5] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/

the graph (measured in MB). Experiments were run on a Sun Fire X4100 M2 machine with two AMD Opteron 2218 processors with 16GB RAM, running a 64-bit Linux CentOS 5.5. All tests were executed in cold cache, i.e., we cleared the cache before running each task by performing the command `sh -c "sync ; echo 3 > /proc/sys/vm/drop_caches"`. Additionally, the machine was dedicated exclusively to run these experiments. The evaluated graph-based tasks are the following:

Graph Creation: creates and stores internal representation of a graph.

Adjacency: checks node/edge adjacencies.

Reachability: traverses a graph following different strategies: *Breadth-first search* (BFS) and *Depth-first search* (DFS). Additional, k*-hops* retrieves sets of nodes such that there is a path of length k from a given start node. External implementations rely on basic adjacency methods, while internal implementations use API methods provided by the engines to solve the task.

Pattern matching: solves subgraph isomorphisms. It was evaluated as the result of traversing the graphs and finding the subgraphs that meet the given patterns; we call this implementation `internal`. Additionally, pattern matching tasks were specified as SPARQL and Cypher queries and evaluated in RDF-3x and Neo4j, respectively.

Densest subgraph: given a graph $G = (V, E)$ this tasks is to find a bipartite subgraph BSG between subsets S and T of V, such that, that BSG maximizes the density, i.e., $d(S,T) = \frac{|E(S,T)|}{\sqrt{(|S||T|)}}$ where *E(S,T)* is the set of edges going from S to T. The evaluated algorithm corresponds to the one proposed by Saha et al.[8]; our implementation exploits node/edge adjacency API methods of the engines.

Graph summarization: given a graph $G = (V, E)$ this tasks is to find a compact representation of G or aggregate graph SG comprised of hyper-nodes, hyper-edges, and corrections. Hyper-nodes correspond to sets of nodes in G, while a hyper-edge connects two hyper-nodes and represents set of edges between all pairs of nodes in the two hyper-nodes. The set of corrections corresponds to additions or deletions of edges represented in the hyper-edges of SG and that are either not present in G (deletions) or that are not presented in the hyper-edge but that were in G. We evaluate the performance of the greedy algorithm proposed by Navlakha et al.[5] on Neo4j, DEX and HypergraphDB; our implementation exploits node/edge adjacency API methods of the engines.

We will demonstrate the following use cases:

Effects of graph characteristics on the performance of the graph and RDF engines. Graphs are characterized by density, number of edges and nodes, and label distribution. Attendees will be able to choose between diverse graphs, and analyze the performance (time and memory) of the different engines in all the studied graph-based tasks. First, time required to create the internal representation of a graph is affected by both the density of the graph and the number of edges in any engine. Additionally, we will be able to observe that even RDF-3x outperforms the rest of the engines in pattern matching, its performance is impacted whenever the graph is dense. Further, graph density, size and number of labels affect the performance of both graph summarization and densest subgraph in all the engines. Nevertheless, graph summarization seems to be more impacted by the graph density and the number of labels than for the size of the graph. Contrary, densest subgraph is more influenced by the size of the graphs.

Effects of the techniques implemented by a given engine in the performance of the

graph-based tasks in different graphs. Attendees will observe that RDF-3x exhibits the best performance during graph creation and adjacency tasks (expressed as SPARQL queries); in case of k-*hops*, RDF-3x also outperforms the rest of the engines, except in the case of dense graphs. DEX seems to overcome the rest of the engines when the graphs are dense, while Neo4j exhibits better performance in sparse graphs whenever they have a large number of labels, e.g., USA-road-d.NY and USA-road-d.FLA.

Impact of a given tasks in the performance of the graph and RDF engines. We show the impact that a given task can have in the performance of an engine. For example, during graph creation RDF-3x can exploit main-memory data structures, B+-tree indices and internal representation of a graph, and exhibits the best performance. Similarly, because RDF-3x implements optimization and execution techniques that exploit the properties of a graph internal representation; thus, the best implementation of this task seems to be on top of RDF-3x. During the evaluation of k-*hops*, DEX and Neo4j are competitive. For traversals, internal implementations are able to exploit the properties of the data structures and indices implemented by each engine as well as the methods exported in their APIs; in both BFS and DFS, Neo4j and DEX exhibit a similar performance. Finally, when the mining tasks of densest subgraph and graph summarization are considered, DEX performs quite well in mining tasks if graphs are dense, while Neo4j has better performance in sparse graphs with a large number of labels.

4 Conclusions

GRAPHIUM allows to visualize patterns in the performance of graph and RDF engines when they are executed against different benchmarks of graphs and tasks. Different configurations will be analyzed allowing the attendees to understand the graph characteristics and tasks that benefit the performance of existing graph and RDF engines.

References

1. C. C. Aggarwal and H. Wang. Graph data management and mining: A survey of algorithms and applications. In *Managing and Mining Graph Data*, pages 13–68. 2010.
2. B. Iordanov. Hypergraphdb: A generalized graph database. In *WAIM Workshops*, pages 25–36, 2010.
3. D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. *Operations research*, 39(3):378–406, 1991.
4. N. Martínez-Bazan, V. Muntés-Mulero, S. Gómez-Villamor, J. Nin, M.-A. Sánchez-Martínez, and J.-L. Larriba-Pey. Dex: high-performance exploration on large graphs for information retrieval. In *CIKM*, pages 573–582, 2007.
5. S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *ACM SIGMOD*, pages 419–432. ACM, 2008.
6. T. Neumann and G. Weikum. x-rdf-3x: Fast querying, high update rates, and consistency for rdf databases. *PVLDB*, 3(1):256–263, 2010.
7. I. Robinson, J. Webber, and E. Eifrem. *Graph Databases*. O'Reilly Media, 2013.
8. B. Saha, A. Hoch, S. Khuller, L. Raschid, and X.-N. Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *RECOMB*, pages 456–472, 2010.
9. A. Thor, P. Anderson, L. Raschid, S. Navlakha, B. Saha, S. Khuller, and X.-N. Zhang. Link prediction for annotation graphs using graph summarization. In *ISWC*, pages 714–729, 2011.