ACCSE 2023

Performance and Scalability of Datastore Technologies for Software Analysis Models

Kanishqk Singh and Robert J. Walker
University of Calgary
Calgary, Canada

1

Software Systems Change over Time

- Real-world software systems are ...
  - large
  - developed over time
  - subject to changing business and technical environments
  - developed by changing groups of developers
- In principle, developers need only a text editor to make changes

Software Development & Analysis Tools (SDATs)

- In practice, specialized tools (SDATs) are needed to …
  - analyze potential changes
  - make actual changes
  - catch errors arising from incomplete or incorrect changes
- SDATs usually build atop analysis-oriented models of the software
  - abstract syntax trees (ASTs)
  - control-flow graphs
  - type hierarchies
  - call graphs
  - …

Size versus cost

- Computing any such model has a cost
- Small software: typically, low cost
- Large software: typically, high cost
  - e.g., system dependence graphs can take days to compute for enterprise-scale software
- When software undergoes changes, its models become obsolete
  - Model update can be complex, error-prone, and still expensive
  - Model re-computation has the same cost as the original
- Since real software undergoes change constantly, its models can be obsolete before they are fully re-computed

Caching versus Re-computation

- When the software system is re-started, we can …
  - recompute its models, paying the same cost as originally done
  - reload a cached version of its models from offline storage
  - perform a combination of these
- For caching & reloading, there are several sources of cost:
  - communicating with an offline storage system
  - writing to an external storage medium
  - reading from the external storage medium
  - communicating with the offline storage system

caching (first two items)

reloading (last two items)

size & complexity of models, details of storage technology

- Reloading a cached version may or may not be cheaper than re-computation!

Dimensions of Consideration

- Datastore technologies
  - Flat files: simple text; comma-separated values (CSV); JSON
  - Relational database management systems: e.g., MySQL, PostgreSQL, etc.
  - Non-relational database systems: NoSQL; graph databases (e.g., Neo4j); cloud storage (e.g., Google Cloud)
- Datasets
  - Academic studies tend to utilize toy datasets, constructed from random graphs
  - Non-academic studies tend to suffer from potential bias
- Use cases
  - SDATs use graphs, so graph-based operations should be studied

Our Study (1/4)

- Research question: *How do different database technologies perform on realistic operations over realistic software analysis models?*
- Many details in the paper

Our Study (2/4)

- The technologies we chose to examine:
  - CSV files via the Python-based *NetworkX* library
  - MySQL
  - PostgreSQL
  - Neo4j

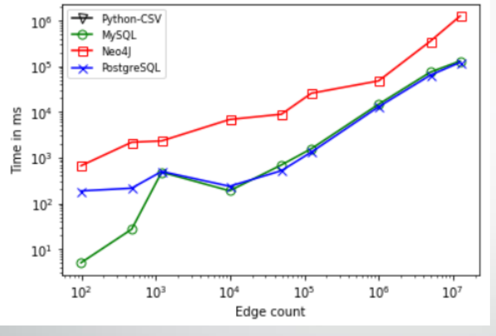
Our Study (3/4)

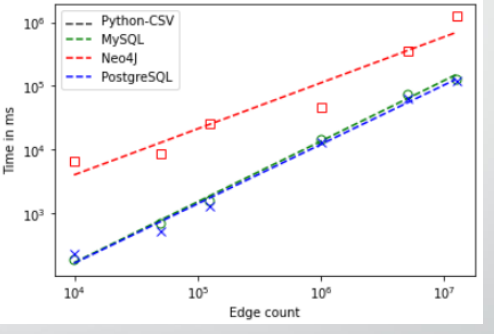
- We generate nine scale-free graphs via the Barabási–Albert model
  - linear preferential attachment model ("the rich get richer")
    - probability of adding an edge with a node is proportional to local degree of connectivity
  - two dimensions
    - #nodes: 100, 1 000, and 10 000
    - density: 2%, 10 %, 25%
- We used a custom Python implementation based on the *NetworkX* library to generate these

Our Study (4/4)

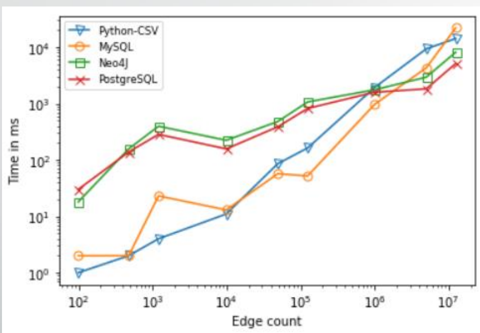
- We examine 8 use cases
  - (UC1) Create/store a graph
  - (UC2) Read/access a graph
  - (UC3) Add a node
  - (UC4) Add an edge
  - (UC5) Rename an edge
  - (UC6) Change source and target nodes of an edge
  - (UC7) Delete a node
  - (UC8) Delete an edge

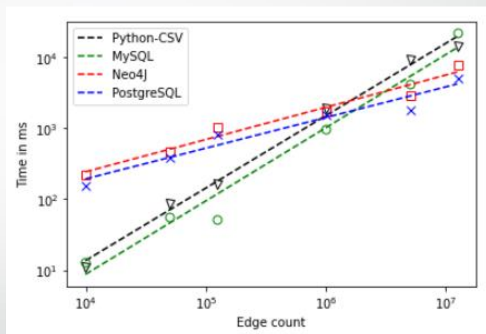
UC1: Create/Store a Graph



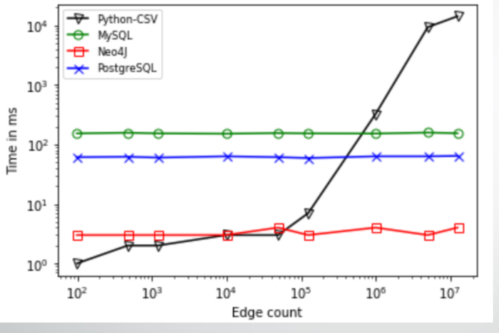


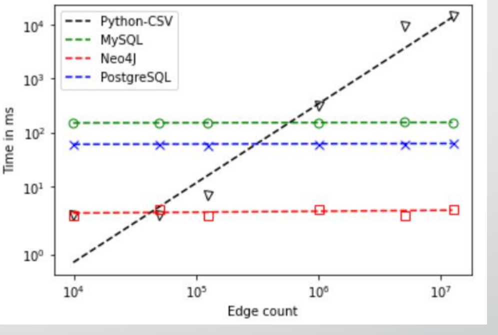
UC2: Read/Access a Graph



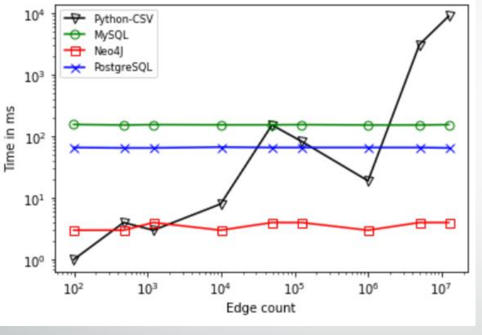


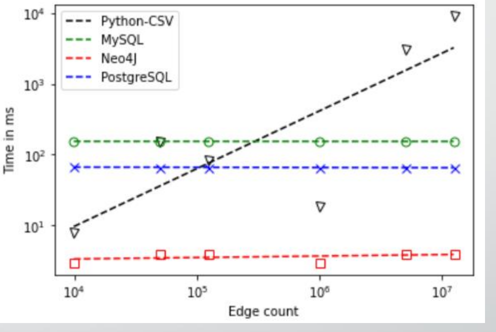
UC3: Create a Node without Edges



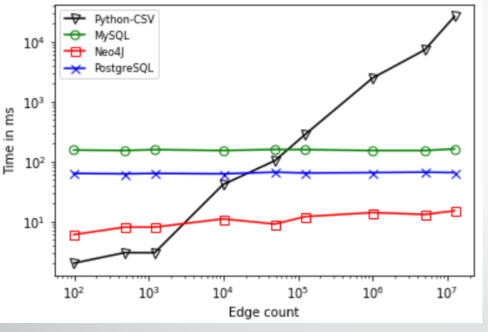


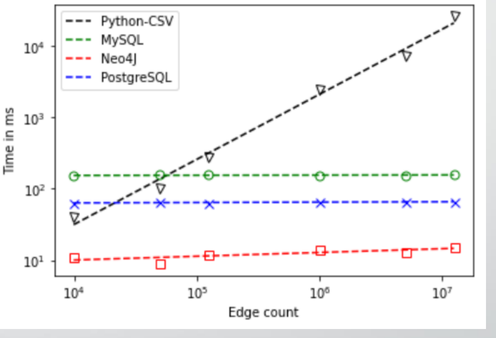
UC4: Create an Edge between Existing Nodes



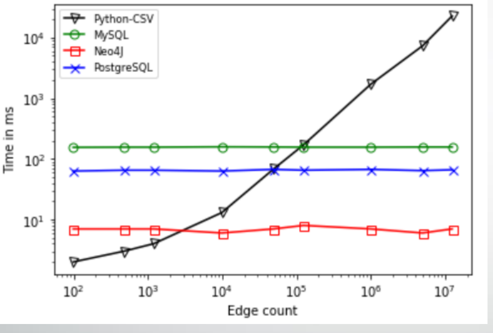


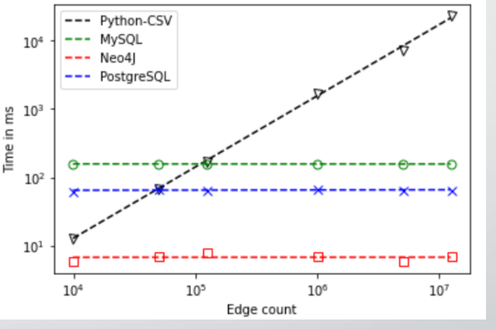
UC5: Rename a Node



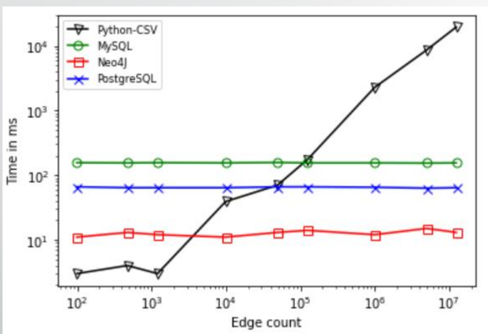


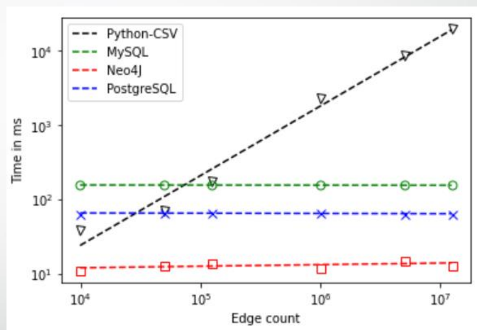
UC6: Change Source & Target Nodes of an Edge



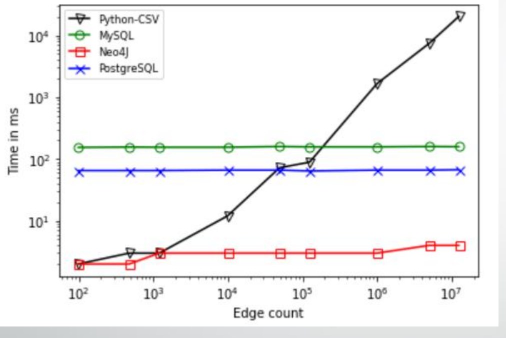


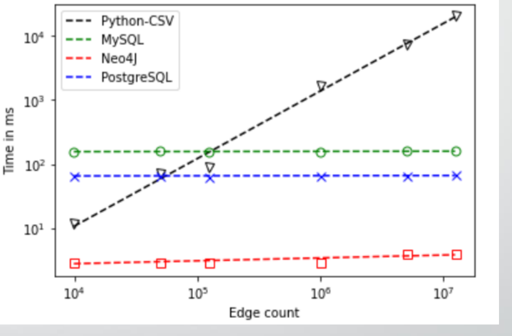
UC7: Delete a Node and Its Corresponding Edges





UC8: Delete a Specific Edge





Conclusions

- For creating/storing a graph, Python-CSV is the clear winner
- For reading a graph, PostgreSQL is the best option for large graphs
- For the other 6 use cases, Neo4j is the best option for large graphs
- The correct choice would depend on the profile of the application
- HOWEVER…
  - This study did not consider the cost of the connector technology needed for programmatic access to core-memory representations; this can be EXPENSIVE
  - A far simpler approach, like object serialization, could suffice for caching/reloading where external manipulation of the graphs is not needed
  - Additional study is needed to determine the comparative, full costs for both issues

THANK YOU

Prof. Robert J. Walker, walker@ucalgary.ca