Spark Graph: Property Graphs, Cypher Queries, and Algorithms

Authors:

Martin Junghanns (martin.junghanns@neo4j.com)
Max Kiessling (max.kiessling@neo4j.com)
Mats Rydberg (mats.rydberg@neo4j.com)
Philip Stutz (philip.stutz@neo4j.com)
Alastair Green (alastair.green@neo4j.com)
Weichen Xu (weichen.xu@databricks.com)

Xiangrui Meng (meng@databricks.com)

Background and motivation

GraphX was one of the foundational pillars of the Spark project, and is the current graph component. This reflects the importance of the graphs data model, which naturally pairs with an important class of analytic function, the network or graph algorithm.

However, GraphX is not actively maintained. It is based on RDDs, and cannot exploit Spark 2's Catalyst query engine. GraphX is only available to Scala users.

GraphFrames is a Spark package, which implements DataFrame-based graph algorithms, and also incorporates simple graph pattern matching with fixed length patterns (called "motifs"). GraphFrames is based on DataFrames, but has a semantically weak graph data model (based on untyped edges (i.e. relationships) and vertices (i.e. nodes)). The motif pattern matching facility is very limited by comparison with the well-established Cypher language[1].

The Property Graph data model has become quite widespread in recent years, and is the primary focus of commercial graph data management and of graph data research, both for on-premises and cloud data management. Many users of transactional graph databases also wish to work with immutable graphs in Spark.

The idea is to define a Cypher-compatible Property Graph type based on DataFrames; to replace GraphFrames querying with Cypher and to reimplement GraphX/GraphFrames algos on the PropertyGraph type.

To achieve this goal, a core subset of Cypher for Apache Spark (CAPS), reusing existing proven designs and code, will be employed in Spark 3.0. This graph query processor, like CAPS, will overlay and drive the SparkSQL Catalyst query engine, using the CAPS graph query planner.

Q1. What are you trying to do? Articulate your objectives using absolutely no jargon.

- The SPIP introduces Cypher 9 support for Apache Spark, i.e. given a single input PropertyGraph and a Cypher 9 query, the query engine creates a tabular result (just like the Neo4j database system) represented by a single DataFrame.
- The implementation is based on Spark SQL DataFrames, where PropertyGraphs are composed of at least one DataFrame (one or more for nodes, zero or more for relationships).

[1] See Appendix A for details on Cypher, and the current Cypher for Apache Spark (CAPS) project.

- The proposed Spark module is a migration target from GraphFrames and GraphX, i.e. it will supersede the functionality of both libraries and allow graph algorithms to work with PropertyGraphs.
- The SPIP introduces an API module and an implementation of that API, providing support for Cypher 9. However, it allows (extended) implementations of the proposed Graph APIs (e.g. Cypher-for-Apache-Spark or extended graph algorithms).
- Support saving PropertyGraphs to the file system in such a way as to easily enable Hive external tables and DF/data source optimizations relating to predicate pushdown and column filtering as data is lifted into memory.
- Provide Scala, Python, Java and R APIs to make the proposed features accessible from those languages.

Q2. What problem is this proposal NOT designed to solve?

- This proposal addresses the Cypher Property Graph data model (directed, labelled multi-graph with element properties) and does not aim to deal with variants of that model or with the RDF triples graph model.
- The first iteration of this proposal will not cover advanced Cypher 10 querying capabilities like query based graph construction or querying multiple named graphs in a single cypher query.
- The SPIP will not introduce a dedicated graph catalog nor include support for loading and storing graphs through arbitrary property graph data sources, or for specific non-file system data sources.

Q3. How is it done today, and what are the limits of current practice?

Within the Spark eco-system, there are currently three libraries that provide graph algorithms and graph querying capabilities: GraphX, GraphFrames and Cypher for Apache Spark (CAPS).

GraphX:

While already providing support for several graph algorithms based on graph-parallel computation, GraphX lacks support for graph querying and PropertyGraphs (i.e. no labels, no relationship types, no relationship identifiers). Additionally, GraphX uses RDDs rather than DataFrames, and is thus not able to use the Catalyst optimiser and suffers from handling large amount of small objects. GraphX is only available in Scala.

GraphFrames:

GraphFrames provides support for graph algorithms (some delegated to GraphX) and basic graph querying (i.e. motif detection). DataFrame-based algorithm implementations, e.g., connected components, demonstrated much better scalability than GraphX. However the proposed query language is less expressive than Cypher 9. Also, similar to GraphX, GraphFrames does not support the PropertyGraph data model.

Cypher for Apache Spark (CAPS):

CAPS already provides Cypher 9 query capabilities on top of Spark SQL. In addition, CAPS implements additional features, e.g. property graph data sources, and experimental features, e.g. support for multiple graphs as defined in the Cypher 10 language specification. CAPS is currently available as a third-party add-on. In fact, the proposed SPIP will reuse many of the implementation and design choices already made in CAPS.

The SPIP proposes to merge the functionality of both GraphX and GraphFrames, lift the algorithms to the Property Graph Model and replace motif detection with a true graph query language (Cypher 9) tailored for the Property Graph model.

Q4. What is new in your approach and why do you think it will be successful?

User experience

- Property Graphs are an important data model in many operational and analytical scenarios. The most prominent use cases for graphs are:
  - Fraud detection
  - Recommendation engines
  - Knowledge graphs
  - (Social) network analysis
  - AI and machine learning
- GraphX, the current built-in graph module, is not sufficient to support all those use cases. It lacks support for graph querying, only integrates with Scala, builts on top of RDDs and lacks scalability for some of the existing graph algorithms.

- Most Spark users already moved from RDD-based APIs to DataFrame-based ones. Building a graph module on top of the DataFrame-based APIs allows for easier integration with existing Spark modules and user applications. Making this graph component built-in allows us to use more advanced and recent features from Spark SQL.

- In comparison to motif detection in GraphFrames, Cypher is a widely used graph query language, formally defined within the openCypher project and key input of a standardization process (“GQL”). GraphFrames on the other hand has more scalable algorithm implementations than GraphX, which will be ported into the new module.

Technically

This design extends beyond GraphX and GraphFrames:

- Bases Spark graphs on DataFrames in line with Spark 2.

- Provides the power of the pre-existing, widely-used and multiply-implemented Cypher query language, enlarging the number of developers who will be able to easily operate on graphs in Spark.

- Cypher allows labels to be used to create mixin-style complex data types for nodes and relationships. A set of labels and its associated properties characterizes a node or relationship type. Labels can be used in graph pattern matches to simplify many queries.

- Having typed nodes and relationships allows for additional type-based constraints or descriptors. For example, permitted or actual combinations of start node/relationship/end node types. The resulting description of a Property Graph data model closely resembles an Entity-Relationship model, allowing graphs to be understood as linked semantic elements.

- Graphs can be understood as complex structures of tables, allowing interwoven graph and tabular processing using SQL and Cypher over the same data.

Q5. Who cares? If you are successful, what difference will it make?

Developers can leverage existing knowledge and documentation on use of Cypher to speed up adoption and make graph data easier to use in Spark.

SQL data modellers and app developers will be able to use a strongly-typed graph data model as a direct translation from conceptual data models expressed as ER models. This will aid graph adoption among database architects and developers.

Many real-world SQL queries are already graph-oriented, e.g. incident response, fraud detection etc. Cypher helps formulating those queries in a much simpler way, thereby helping understandability and maintainability.

The semantic model of a graph using node and relationship types also allows superior implementations of

- partitioning/data placement
- query planning (reduces search space for label-predicates, also allowing label-based inferencing).
- graph algorithms which can become "schema aware"

Q6. What are the risks?

- We think the risk for the implementation to fail is very low since we already went through the process of implementing such a query engine (i.e. Cypher-for-Apache-Spark) and will reuse major parts of it while implementing the SPIP.
- Addition of specialized DF types will increase footprint of Spark for alternate languages.

Q7. How long will it take?

- If accepted by the community by the end of January 2019, we predict to be feature complete by the mid of May followed by four weeks of QA, making the SPIP part of the next major Spark release (3.0, ETA July, 2019).

Q8. What are the mid-term and final “exams” to check for success?

Mid-term exam

- Spark PropertyGraph type and associated builders and utilities implemented in Scala.

- Subset of existing Cypher for Apache Spark (CAPS) tests covering proposed Cypher language features have been copied over into Spark and integrated with Spark testing frameworks.

- Spark PropertyGraph can be created, existing CAPS libraries can be dynamically loaded, and the ported Cypher functionality/coverage tests complete without regression using existing CAPS Cypher implementation.

- Spark PropertyGraph type can be successfully used by some Graph Algorithms ported from GraphFrames/GraphX.

- Canonical code examples showing use of Cypher, key Cypher features, interaction with SparkSQL completed.

- *Cypher for Apache Spark, or “CAPS” has additional features than those proposed for Cypher in Spark Graph, so the use of CAPS via dynamic loading will test the compatibility of Spark PropertyGraph and CAPS PropertyGraph implementations.*

Final exam

- Java, Python, R versions of API available.

- Subset of CAPS code implementing Cypher 9 will be incorporated in the main Spark project, and all relevant tests working on “full CAPS” operate without regression on the “Spark subset”.

- Canonical code examples showing integration of Cypher and Graph Algorithms. All Graph Algorithms ported from GraphFrames/GraphX.

- Documentation using the code examples for queries and algorithms completed.

- Implementation is ready for QA: required QA development has been completed to allow large-scale integration/acceptance tests for Spark 3.0 to integrate Spark Graph Cypher and Algorithms.

Appendix A: the Cypher query language

Cypher (originating with Neo4j, now maintained by the openCypher project) is the most widely used property graph query language. Unlike Apache Tinkerpop/Gremlin (which uses a very similar data model), Cypher is a declarative query language like SQL, which uses graph patterns to identify, retrieve and manipulate parts of the graph and the property values associated with them.

Cypher 9 is the version of Cypher implemented by Neo4j 3.3 and earlier, described in the openCypher [Cypher Reference]. Cypher 10 incorporates extended functionality, including date-time support [CIP2015-08-06], and multiple graph/composable query features ([CIP2018-05-03], [CIP2017-06-18]). This proposal focuses on the well-established, stable features of Cypher 9.

Cypher 9 has been implemented in other commercial products like SAP Hana Graph, Agens Graph, Tigergraph, Memgraph, and Redis Graph, and in half a dozen research projects. There are also two Apache-licensed OSS projects, Cypher for Apache Spark (CAPS) and Cypher for Gremlin (CfoG).

Importantly, openCypher is one of the key inputs to the proposed new ISO standard, GQL, which will complement SQL. GQL will not emerge until 2020 at the earliest; openCypher is a good stepping stone.