Using Graph Databases

Julius Hřivnáč[1,*] on behalf of the ATLAS Computing Group

[1]Université Paris-Saclay, CNRS/IN2P3, IJCLab, 91405 Orsay, France

Abstract. Data in High Energy Physics (HEP) usually consist of complext complex data structures stored in relational databases and files with internal schema. Such architecture exhibits many shortcomings, which could be fixed by migrating into Graph Database storage.
The paper describes basic principles of the Graph Database together with an overview of existing standards and implementations. The usefulness and usability are demonstrated using the concrete example of the Event Index of the ATLAS experiment at LHC in two approaches - as the full storage (all data are in the Graph Database) and meta-storage (a layer of schema-less graph-like data implemented on top of more traditional storage). The usability, the interfaces with the surrounding framework and the performance of those solutions are discussed. The possible more general usefulness for generic experiments' storage is also discussed.

1 HEP Storage

Traditionally, several data structures are used in High Energy Physics (HEP): tuples (tables), hierarchical structures like trees, relational (SQL-like) databases or their combinations (nested tuple, trees of tuples). They can be based on a schema or without a defined schema. Many HEP data sets are graph-like without general schema. They consists of entities with relations. Such structures are not well handled by standard tree-ntuple storage and relations need to be added and interpreted outside of that storage, in the application program. Such data are also not well covered by relational (SQL) databases because users should be able to add new relations, not covered by existing schema. We don't only need the possibility of adding new data with the existing defined relations, we also need to add new relations. It has been also proven difficult to manage HEP data by Object Oriented (OO) databases or serialization due to serious problems in distinguishing essential relations from volatile ones. The Object Oriented databases have been abandoned by most major particle physics experiment. [1]

2 Graph Databases

Graph Databases have existed for a long time, but they have matured only recently thanks to Big Data and Artificial Intelligence (Graph Neural Networks). [2],[3] Implementations and de-facto standards are available and evolving rapidly.

[*]e-mail: Julius.Hrivnac@cern.ch

By using Graph Databases, we are moving essential structure from code to data, together with a migration from an imperative to a declarative coding semantics. The new paradigm of Graph databases can be shortly described as *Things don't happen, they exist.* Structured data with relations then facilitates declarative analyses. The data elements appear in a context, which simplifies their understanding, analyses and processing. The difference between Relational and Graph Database is similar to the difference between Fortran and C++ or Java. On one side, we have a rigid system, which can be very optimized. On the other side, there is a flexible dynamical system, which allows expressing of complex structures. Graph Database can be considered a synthesis of the OO world and Relational databases. It supports the expression of the web of objects without fragility of the OO world by capturing only essential relations and not an object dump.

Graph Database stores graphs in a database store. Those graphs (G) are generally described as sets of vertices (V) and edges (E) with properties: $G = (V, E)$.

2.1 Graph Database Languages

There are, in general, three methods which can be used to access a Graph Database.

- **Direct manipulation** of vertices and edges is always available from all languages, but this doesn't use full graph expression power of more specialized approaches.
- **Cypher [5] (and GQL [6])** is a pure declarative language, inspired by SQL and OQL, but applied to schema-less databases. It comes from Neo4J [7] and has been accepted as an ISO/IEC standard. It is available to all languages via a JDBC-like API [8]. Its problematic feature is that it introduces a semantic mismatch as instructions are passed as a string and not as code understandable to the enveloping environment. There is a wall between code and database, with a thin tunnel which only strings can pass. The overall framework cannot easily understand and handle the logic of the database code as it is writtent in a different language and is using a different paradigm.

  The following Cypher code searches for names of all datasets with certain run number.

  ```
  MATCH (a:run)-[:has]->(b:dataset)
    WHERE a.rnumber = 98765
    RETURN b.name
  ```

- **Gremlin** [9] is a functional language extension motivated by Groovy [11] syntax, but available to all languages supporting functional programming. Gremlin is well integrated in the framework language.

  Gremlin can execute the same search as the Cypher example above with this code:

  ```
  g.V().has('run', 'rnumber', 98765)
       .out('has')
       .values('name')
  ```

3 ATLAS Event Index

The Event Index service [13] of the ATLAS [14] experiment at the Large Hadron Collider (LHC) keeps references to all real and simulated ATLAS events. Hadoop [15] Map files and HBase [16] tables are used to store the Event Index data, a subset of data is also stored in an Oracle database. The system contains information about events, datasets, streams, runs and their relations. Several user interfaces are currently used to access and search the data, from the simple command line interface, through a programmable API, to sophisticated graphical web services.

3.1 ATLAS Event Index History

The original Event Index stored all data in Oracle. It was too rigid, it was difficult to add columns or relations and the whole system also suffered from performance problems.

In 2013 it was decided to migrated to a Hadoop [15] ecosystem, while keeping a subset of data also in Oracle. All data were now stored in a tree of HDFS Map files. Such system was flexible and fast for mass processing, but too slow for search requests. Another problem was the lack of types of the Map files (they store only strings or bytes); the type system has been created in the application layer.

To solve the search performance problem, the data were partially migrated to HBase [16]. Those HBase tables contain ad-hoc relations (references to other entries). Those relations form a *poor-man graph database* on top of HBase.

Several prototypes have been then developed to study the next generation Event Index, which will be fully deployed for ATLAS Run 3 at the end of 2020. Those prototypes use Graph Database more directly in different ways:

- **Prototype 1** stores all data directly in a JanusGraph [17] database over HBase storage.
- **Prototype 2** stores data in a HBase table with a Phoenix [18] SQL interface, graph structure is added via another auxiliary HBase table.

3.2 Prototype 1 - JanusGraph

A subset of data has been fully imported into a JanusGraph [17] database storing data in an HBase table. Part of existing functionality has been implemented. A large part of code (handling relations and properties) has been migrated from the code to the structure of the graph. Most of the graphical interface has been implemented by a standalone JavaScript implementation.

Prototype 1 uses the de-facto standard language for Graph Database access - Gremlin [9]. Gremlin is a functional, data-flow language for traversing a property graph. Every Gremlin traversal is composed of a sequence of (potentially nested) steps. A step performs an atomic operation on the data stream. Every step is either a map-step (transforming the objects in the stream), a filter-step (removing objects from the stream), or a side-effect-step (computing statistics about the stream). Gremlin supports transactional and non-transactional processing in a declarative or imperative manner. Gremlin can be expressed in all languages supporting function composition and nesting. Among supported languages are Java, Groovy, Scala, Python, Ruby and Go. Gremlin is generally used within the TinkerPop[10] framework and currently the leading implementation is JanusGraph. It supports several storage backends: Cassandra, HBase, Google Cloud and Oracle BerkeleyDB, several graph data analytics: Spark, Giraph and Hadoop and several search tools: Elasticsearch, Solr and Lucene.

Gremlin, an API originated from Groovy language, uses functional syntax (with streams and Lambda calculus) and functional and navigational semantics. It is very intuitive, uses no special syntax and is easily integrated into existing frameworks. Database data are simply accessed as objects with structure, relations and nested collections with links.

Following examples show some capabilities of the Gremlin code:

```
# add a vertex 'experiment' with the name 'ATLAS'
g.addV('experiment').property('ename', 'ATLAS')
# add edges 'owns' from all vertices 'project' to vertex 'experiment' 'ATLAS'
g.V().hasLabel('project')
     .addE('owns')
     .from(g.V()
     .hasLabel('experiment')
```

```
      .has('ename', 'ATLAS'))
# show datasets with more events or number of events in an interval
g.V().has('run', 'number', 358031)
      .out()
      .has('nevents', gt(7180136))
      .values('name', 'nevents')
g.V().has('run', 'number', 358031)
      .out()
      .has('nevents', inside(7180136, 90026772))
      .values('name, 'nevents')
# Event-Lookup function (server side UDF)
def el(run, event, g) {
  e = g.V().hasLabel('run')          # all runs
           .has('rnumber', run)      # selected run
           .out('fills')             # all datasets filling that run
           .out('keeps')             # all events kept in that dataset
           .has('enumber', event)    # selected event
           .values('guid')           # its guid
  }
```

3.3 Prototype 2 - Phoenix

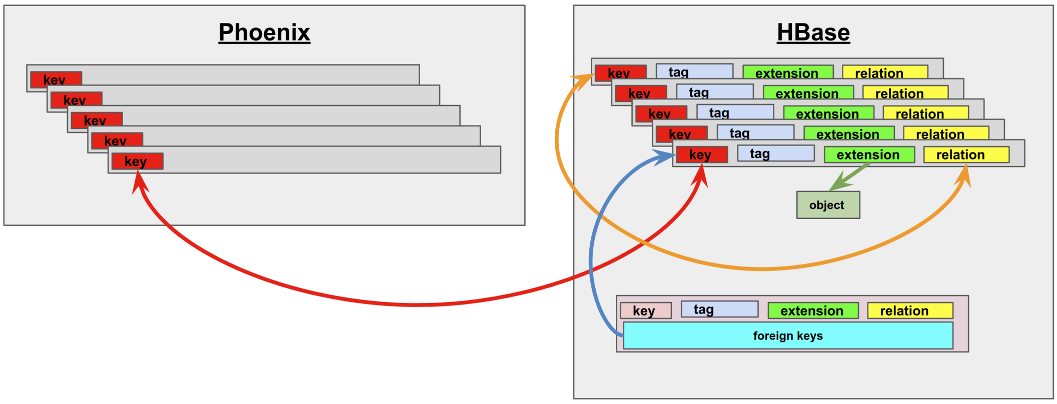


Figure 1. Schema of the Prototype 2 databases.

The aim of this prototype is to use simple and flexible NoSQL storage, be compatible with other SQL databases used in ATLAS and to take advantages of Graph Database environment.

This has been achieved by extending Phoenix [18] SQL interface (backed by HBase storage) with an additional pure HBase database. This way, we can keep Phoenix advantages (speed, SQL interface) for read-only data, while opening new possibilities and adaptability to changing environment. Phoenix is used for static data and HBase for dynamic data. Bulk data are stored in a pure HBase table with the Phoenix SQL API, graph structure is added via another HBase table. A subset of the Gremlin interface has been implemented for the graph part of the storage.

The graph structure is created on the top of an SQL database - a user sees just one database. Both databases share the same keys. All data of one key is represented by one Element (a generic class). The Graph HBase database is much smaller as it contains only subset of data. It can contain

- **Simple Tags** which can be also used in a search filter.
- **Extensions** by any object, like trigger statistics and overlap or duplicated events list.
- **Relations** to other elements (i.e. the Graph Database).

HBase can also contain elements without a Phoenix partner: **Hubs**. They represent virtual collections of Elements, like external tags, stream names, run numbers or project names. Hubs can be extended and searched in the same way as other elements. A schema of this prototype database is shown in Figure 1.

3.4 Web Service

Generic web service graphical visualization has been implemented completely in JavaScript so it doesn't require any server-side code. It can display any Gremlin-compatible database, it works with both Prototypes. The Gremlin server delivers a JSON view of data to the Web Service. Direct API and REST web service are also available.

The web service graphics uses a hierarchical zooming navigation. It can show all available data, their relations and properties and all available actions for them (see Figure 2).

3.5 Value-add and New Possibilities

Storing HEP data in a Graph Database can make the exeriment frameworks more flexible, readable, stable and performant.

A big part of the application code is absorbed in the Graph Database structure. Implementation and optimization details are delegated to a suitable database engine. The database carries information about the data structure which would otherwise have to be handled by the application code.

A user can use the system via a simple graphical access web service. A JavaScript client connects directly to Gremlin server.

Standards are used, so components can be replaced. For example, a prototyping work has been done using simpler Cassandra[12] database.

Virtual entities can be created, including virtual collections (whiteboard functionality), either personal or official. Requests results can be persisted. Results can be stored as new objects with relations (cache functionality).

3.6 Performance

Requests are in general executed in three phases:

- First the initial entry point (event, dataset, run, stream or version) is searched for. This part could be optimized by using natural order, indexes, Elastic Search, Spark or more hierarchical navigation.
- The graph is navigated. This part is very fast. The navigational database access time is small in comparison with the data management in the application code.
- Finally the results are accumulated.

Data can still be accessed directly, without Graph Database API, so the same performance as non-Graph Database can be easily achieved. The navigation step (instead of sub-search) can speed it up. In general, the system exhibits very fast retrieval and slower import, as the latter creates structure that are then used in faster and simpler search.

4 Graph Database for Functional Programming

Using Graph Database extends the parallel-ready functional model from code to data! Relations (edges) can be considered as functions, navigation as a function execution. From the user point of view, there is no difference between creating a new object and navigating to it. Both operations can be *lazy* (i.e. executed only when needed). Functional processing and graph navigation (*Graph Oriented Programming*[4]) can work well together. They use the same functional syntax. Both are an actual realization of Categories[19], where Vertex is an Object and Edge is a Morphism. Functional program can be also modeled as a graph and graph data can be navigated using functions. Graph data are ready for parallel access.

5 Graph Database for Deep Learning

Graph Neural Networks create a natural environment for Deep Learning.

A Neural Network is a graph so Graph Database is a natural environment to describe Neural Network itself. In many cases, Neural Network handles graph data (objects with relations), operating either on the individual nodes (Node-focused tasks) or on the whole graph (Graph-focused tasks). Graph Neural Networks can be seen as generalization of a (non-geometric) Convolutional Neural Network. That opens possibilities to impose contraints/knowledge to Neural Networks, either via Inductive Bias or Semantic Induction. More detailed information and references can be found in [20].

6 Conclusion

6.1 ATLAS Event Index

The Graph database part of the new ATLAS Event Index is ready for the new ATLAS offline framework being build for the ATLAS Run 3, which was originally scheduled for 2021, but is delayed because of the Coronavirus pandemic. The Graph database performance is at least as good as the performance of the current system based on pure HBase database. As the full chain has not been delivered yet and all existing data have not been replicated in the new Phoenix database, the overall performance gain cannot be faithfully evaluated. Some functionality enhancements and more intuitive interface has been already acknowledged and are helping in integrating the system in the new ATLAS offline framework.

6.2 Graph Databases for HEP

Significant HEP effort is spent making execution more structured and parallel by using parallel or functional programming. Less effort has been spent, so far, structuring the data which can lead to simpler and faster access.

Graph Databases offers many advantages:

- They deliver more transparent code by simplifying data access layer.
- Stable data structure is handled in the storage layer.
- They are suitable for Functional Style and Parallelism.
- They are suitable for Deep Learning.
- They are suitable for Declarative Analyses.
- They can help with Analysis Preservation as the stored data carry enough information for their interpretaion.

- They are language and Framework neutral.

There are two possible ways of how to proceed in using Graph databases in new HEP software. Either data can be stored directly in a real Graph Database. Or a graph layer can be build on top of the existing storage close to the database layer.

ATLAS Event Index uses graphs to store higher level (meta)data. Graphs can be also used to store events themselves and other auxiliary structures (geometry, conditions,...) to give graph functionality to all data.

References

[1] Becla, Jacek and Wang, Daniel. (2005). Lessons Learned from Managing a Petabyte. SLAC-PUB-10963, DOE: AC02-76SF00515

[2] Z. Zhang, P. Cui and W. Zhu, Deep Learning on Graphs: A Survey in IEEE Transactions on Knowledge and Data Engineering, DOI: 10.1109/TKDE.2020.2981333.

[3] Battaglia, Peter W. et al. Relational inductive biases, deep learning, and graph networks. ArXiv abs/1806.01261 (2018)

[4] Rey, Olivier. (2016). The graph-oriented programming paradigm

[5] Cypher: https://neo4j.com/developer/cypher-query-language

[6] GQL: https://www.gqlstandards.org

[7] Neo4J: https://neo4j.com

[8] Java Database Connectivity: https://www.oracle.com/technetwork/java/javase/jdbc/index.html

[9] Gremlin: https://tinkerpop.apache.org/gremlin.html

[10] Apache TinkerPop: http://tinkerpop.apache.org

[11] Apache Groovy: http://groovy-lang.org

[12] Apache Cassandra: https://cassandra.apache.org

[13] The ATLAS EventIndex: an event catalogue for experiments collecting large amounts of data, D. Barberis et al., J.Phys.Conf.Ser. 513 042002 (2014), DOI: 10.1088/1742-6596/513/4/042002

[14] The ATLAS Experiment at the CERN Large Hadron Collider, ATLAS Collaboration, JINST, 3, S08003 (2008), DOI: 10.1088/1748-0221/3/08/S08003

[15] Apache Hadoop: https://hadoop.apache.org

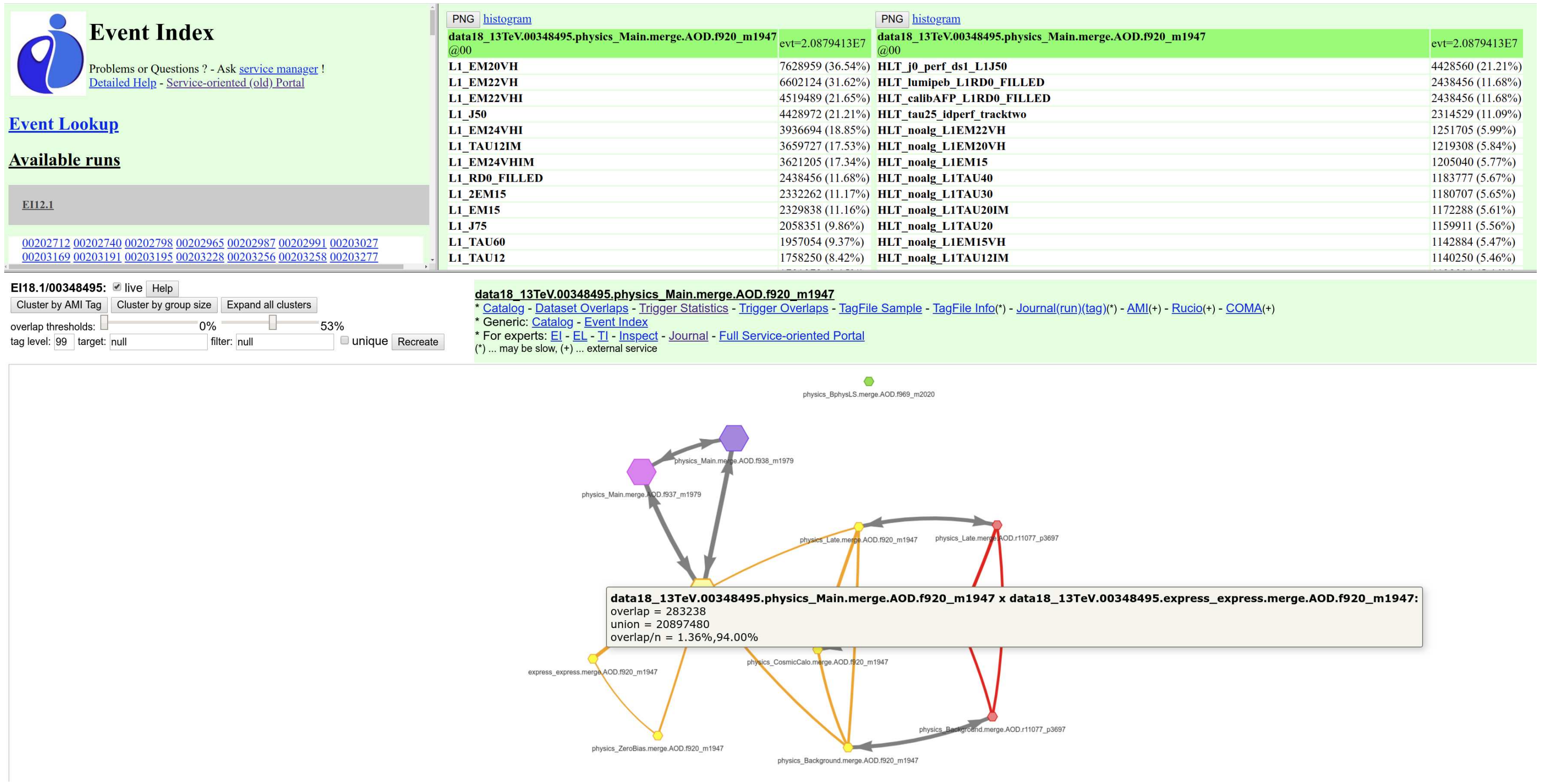
[16] Apache HBase: https://hbase.apache.org

[17] JanusGraph: https://janusgraph.org

[18] Apache Phoenix: https://phoenix.apache.org

[19] Lambek and Scott: Introduction to higher order categorical logic, ISBN: 9780521356534

[20] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborova: Machine learning and the physical sciences, Rev. Mod. Phys. 91, 045002



Event Index

Problems or Questions ? - Ask service manager !
Detailed Help - Service-oriented (old) Portal

Event Lookup

Available runs

EI12.1

00202712 00202740 00202798 00202965 00202987 00202991 00203027
00203169 00203191 00203195 00203228 00203256 00203258 00203277

PNG histogram

data18_13TeV.00348495.physics_Main.merge.AOD.f920_m1947 @00	evt=2.0879413E7
L1_EM20VH	7628959 (36.54%)
L1_EM22VH	6602124 (31.62%)
L1_EM22VHI	4519489 (21.65%)
L1_J50	4428972 (21.21%)
L1_EM24VHI	3936694 (18.85%)
L1_TAU12IM	3659727 (17.53%)
L1_EM24VHIM	3621205 (17.34%)
L1_RD0_FILLED	2438456 (11.68%)
L1_2EM15	2332262 (11.17%)
L1_EM15	2329838 (11.16%)
L1_J75	2058351 (9.86%)
L1_TAU60	1957054 (9.37%)
L1_TAU12	1758250 (8.42%)

PNG histogram

data18_13TeV.00348495.physics_Main.merge.AOD.f920_m1947 @00	evt=2.0879413E7
HLT_j0_perf_ds1_L1J50	4428560 (21.21%)
HLT_lumipeb_L1RD0_FILLED	2438456 (11.68%)
HLT_calibAFP_L1RD0_FILLED	2438456 (11.68%)
HLT_tau25_idperf_tracktwo	2314529 (11.09%)
HLT_noalg_L1EM22VH	1251705 (5.99%)
HLT_noalg_L1EM20VH	1219308 (5.84%)
HLT_noalg_L1EM15	1205040 (5.77%)
HLT_noalg_L1TAU40	1183777 (5.67%)
HLT_noalg_L1TAU30	1180707 (5.65%)
HLT_noalg_L1TAU20IM	1172288 (5.61%)
HLT_noalg_L1TAU20	1159911 (5.56%)
HLT_noalg_L1EM15VH	1142884 (5.47%)
HLT_noalg_L1TAU12IM	1140250 (5.46%)

EI18.1/00348495: ✓ live Help
Cluster by AMI Tag | Cluster by group size | Expand all clusters
overlap thresholds: 0% 53%
tag level: 99 target: null filter: null unique Recreate

data18_13TeV.00348495.physics_Main.merge.AOD.f920_m1947
* Catalog - Dataset Overlaps - Trigger Statistics - Trigger Overlaps - TagFile Sample - TagFile Info(*) - Journal(run)(tag)(*) - AMI(+) - Rucio(+) - COMA(+)
* Generic: Catalog - Event Index
* For experts: EI - EL - TI - Inspect - Journal - Full Service-oriented Portal
(*) ... may be slow, (+) ... external service

data18_13TeV.00348495.physics_Main.merge.AOD.f920_m1947 x data18_13TeV.00348495.express_express.merge.AOD.f920_m1947:
overlap = 283238
union = 20897480
overlap/n = 1.36%,94.00%

physics_BphysLS.merge.AOD.f969_m2020
physics_Main.merge.AOD.f938_m1979
physics_Main.merge.AOD.f937_m1979
physics_Late.merge.AOD.f920_m1947
physics_Late.merge.AOD.r11077_p3697
express_express.merge.AOD.f920_m1947
physics_CosmicCalo.merge.AOD.f920_m1947
physics_Background.merge.AOD.r11077_p3697
physics_ZeroBias.merge.AOD.f920_m1947
physics_Background.merge.AOD.f920_m1947

Figure 2. Data-oriented Graph Web Service.