MillenniumDB: A multi-modal, multi-model graph database engine

Domagoj Vrgoč[1,2] Carlos Rojas[1] Renzo Angles[1,3] Marcelo Arenas[1,2] Vicente Calisto[1]
Benjamín Farías[1,2] Sebastián Ferrada[4] Tristan Heuer[1] Aidan Hogan[1,4] Gonzalo Navarro [1,4]
Alexander Pinto[1,2] Juan Reutter[1,2] Henry Rosales[1] Etienne Toussiant[1]
[1] IMFD [2] PUC Chile [3] Universidad de Talca [4] Universidad de Chile
Santiago, Chile
vrdomagoj@uc.cl,cirojas6@uc.cl,rangles@utalca.cl,marenas@ing.puc.cl,vicente.calisto@imfd.cl,bffarias@uc.cl,
scferradaa@gmail.com,theuer@uc.cl,ahogan@dcc.uchile.cl,gnavarro@dcc.uchile.cl,aupinto@uc.cl,jlreutte@uc.cl,
hrosmendez@gmail.com,tsst.etienne@gmail.com

ABSTRACT

Current knowledge graphs encompass diverse data formats, including images, text, tables, audio files, and videos. Additionally, the graph database ecosystem is required to support multiple coexisting data models. Addressing these challenges is essential for promoting interoperability between data sources. This demo introduces MillenniumDB, a high-performing, open-source graph database handling this diversity of data formats and models.

MillenniumDB is a multi-modal, multi-model graph database, supporting the popular property graph paradigm, the Semantic Web format RDF, and the multi-layered graph model, which combines and extends the two. In terms of querying, its provides support for a Cypher-like language over property graphs and multilayered graphs, as well as SPARQL 1.1 support over RDF. The engine is build on a solid theoretical foundation and it leverages worst-case optimal join algorithms in combination with traditional relational query optimization. It also support a wide array of graph-specific tasks such as path finding, pattern recognition, and similarity search on multimodal data. In this demo, we will showcase how MillenniumDB is currently being used to host three public multi-modal knowledge graphs. The first one, a multi-layered graph called TelarKG, was developed at IMFD Chile to track the information about the Chilean constitutional reform. In the second one, called BibKG, we integrate information about Computer Science publications from different sources and make them available as a property graph. Finally, for RDF, we provide a SPARQL endpoint for Wikidata, the largest knowledge graph openly available online. We remark that our endpoints have stable links, allowing the audience to post queries using their Web browser with no restrictions, and will be available during the review process and during the demo.

KEYWORDS

graph databases, knowledge graphs, property graphs

1 INTRODUCTION

Graph databases have become a key database model in recent years. Graphs offer a much more intuitive and flexible representation of several application domains than traditional relational databases, including bioinformatics, social networks, transport, and more besides [2, 13]. Graphs can also be used to integrate information that spans multiple domains, as seen in open knowledge graphs [13] such as Wikidata [10]. The popularity of graph databases has spurred the development of several data models, such as property graphs [1] and RDF [7]; query languages including Cypher [11], SPARQL [6] and GQL [9]; numerous graph database engines, and commercial products such as Neo4j [24], Amazon Neptune [17] and TigerGraph [19]; among other developments.

But even though the graph database ecosystem is thriving, there are still many requirements that are not yet addressed or entirely solved by current systems. As we explain below, some of these requirements involve the wide range of models and query languages that are currently available. We also find newer technical requirements related to graph analytics and machine learning, driven by modern applications using graph data. This shows that there is still a need to develop graph database systems.

In this demonstration we present MillenniumDB [23]: an open-source graph database system built specifically to address the following three fundamental requirements. The first and most important requirement is **multi-modality**. Knowledge graphs today are composed of data that is stored in different formats. Wikidata [10], for example contains images, text data, tables, and even audio files. However, existing systems can only point to these resources, filtering them based on their neighboring graph structure, or possible information about their names. Hence, we envision that graph database systems should **be able to filter multi-modal data in a native way**, allowing queries that, for example, *return scientists from a particular university working on topics relevant for SIGMOD (text)*, or *return paintings similar to the artwork by a given artist (images)*. We address this by incorporating vector-based similarity search into our graph query engine.

The second fundamental requirement is **incorporating support for different data models and query languages**. A key challenge relating to interoperability within the graph database ecosystem is the variety of models and query languages. From property graphs through RDF to RDF*, and from Cypher through GQL to SPARQL, numerous standardization efforts have not yet resulted in a single model and query language followed and implemented by every graph database system. Furthermore, given the differences between, say, property graphs and RDF, there are enough reasons to assume we are several years short of a unifying standard, if indeed such a standard will ever emerge. Hence, we envision that **systems should natively support several query languages and data models**. We have addressed this requirement by making the interoperability of our engine a priority from its inception, and a key

criterion of its design. The foundations of these distinctive models and query languages share key characteristics that can be exploited for interoperability purposes. Different data models are taken care of by various indexes, all of which provide interoperable interfaces that are then consumed by our engine. This allows us to support, on the same system, a range of graph-based models, including RDF, property graphs, and even graphs using a custom model based on quads that we call multilayer graphs [3].

As is usual in graph databases, **performance** is one of the most important aspects. In this context, MillenniumDB aims to provide and push the boundaries of state-of-the-art performance. Notably, our query engine integrates traditional join-based algorithms with novel worst-case optimal algorithms. The coordination between these two query answering approaches is important to ensure best performance when dealing with complex queries [12].

In comparison to existing alternatives, most graph databases work with one data model (with a notable exception being Amazon Neptune). In contrast, MillenniumDB supports both RDF and multilayer graphs, the latter being a model flexible enough to store a generalized form of property graphs [3]. In terms of multi-modality, to the best of our knowledge, there are no graph database systems that can support both semantic similarity search and graph query answering at the same time. Neo4j, for example, supports storing tensor data for nodes, but such tensors can only be consumed for analytics, and do not integrate with querying functionalities. Unlike many alternatives, MillenniumDB is an open source graph database engine that can be accessed, extended and tested freely.

2 SYSTEM OVERVIEW

We now give a brief overview of the MillenniumDB graph database system. Our architecture is grouped into two main components: the Storage Manager, that orchestrates the index data structures and manages the disk buffer, and the Query Processor, in charge of producing and executing query plans. Figure 1 provides a quick overview of our architecture, and the full codebase can be found at https://github.com/MillenniumDB.

2.1 Storage Manager

MillenniumDB stores the graph internally as tuples of 8-byte identifiers, distinguishing the different components in data models (e.g. nodes, edges and values in a property graph database). Then we use B+ trees to store graphs. The number of B+ trees depends on the data model used for a particular domain graph, but, for example, in the case of property graphs one would have to store relations (sourceid, edgeid, targetid) specifying the id of the source and target nodes, and of the edge nodes, a relation storing all ids, a relation storing the label of each object, and a relation storing every attribute of a node or edge. In order to support traditional and worst-case optimal joins, we store different permutations for each of these relations (as in e.g. [14]), which allow us to access easily all information for nodes or edge, for example, all target nodes of a given edge, or all edges going from a source node, etc.

To support similarity queries, we also support a binary relation that contains tensor data for (a subset of) node or edge ids. These tensors are further indexed using an LSHForest scheme [5], which can be created for a variety of distances, including Euclidean distance and angular distance.

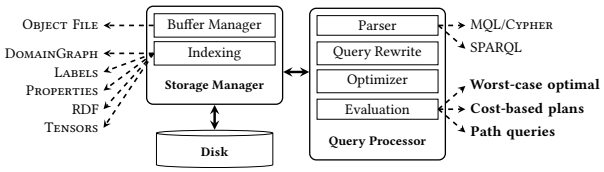

Figure 1: MillenniumDB system architecture

2.2 Query Processor

The evaluation of queries in MillenniumDB follows the standard pipeline of a database query evaluation process. The string of the query is first parsed, and then translated into a logical plan. In turn, this logical plan is analyzed and transformed into a physical plan, which can then be evaluated.

Two notable inclusions in our engine are **support for multiway joins using worst-case optimal algorithms** and **support for similarity search**. Regarding multiway joins, our logical query plans may include pieces of graph patterns that are evaluated using Leapfrog Trie Join (LTJ) [21], a worst case optimal join that works by computing the join of several relations at once. These types of joins have been shown to produce much better results for more complex graph patterns (see e.g. [4, 14, 16]), and indeed we also perceive this in our own evaluation [23]. Regarding similarity search, our system integrates a query command to ask for node ids that are most similar to a given node or resource. To evaluate this command, the query engine treats LSHForests as a virtual relation, which is then integrated into physical plans.

2.3 Performance

In a previous article [23], we presented an experimental evaluation of MillenniumDB. In particular, we evaluated five versions of MillenniumDB: MillenniumDB-LF, the default version implementing the Leapfrog Trie Join algorithm; MillenniumDB-GR, implementing a greedy algorithm for selecting the join order; MillenniumDB-SL, which implements the Selinger join planner; MillenniumDB-BFS, which uses a breadth-first search algorithm to evaluate path queries; and MillenniumDB-DFS, which uses a depth-first algorithm. We also evaluated five graph-oriented database systems: Blazegraph [20], a Java-based RDF store; JenaTDB [18], a component of Jena for storing and query RDF data; JenaLF [14], a version of JenaTDB implementing a Leapfrog-style algorithm; Virtuoso [8], a relational-based RDF store; and Neo4J [24], a popular graph database system.

The experimental evaluation was based on Wikidata [22], which is one of the largest and most diverse real-world knowledge graphs that is publicly available. Specifically, we used a "truthy" dump version, keeping only triples in which the subject position is a Wikidata entity, and the predicate is a direct property. The size of the dataset was 1,257,169,959 RDF triples, resulting in the following storage costs for each system: MillenniumDB = 203GB, BlazeGraph = 70GB, JenaTDB = 110GB, JenaLF = 195GB, Virtuoso = 70GB, and Neo4j = 112GB. MillenniumDB uses extra disk space because of the additional indices needed to support worst-case optimal join over domain graphs (similar to the case of JenaLF).

We conducted a performance evaluation focused on two fundamental query features: graph pattern matching and path matching. Our evaluation of graph pattern matching included 835 real-world graph pattern queries and 850 synthetic queries. The real-world queries were obtained from the Wikidata SPARQL query log [15], and were grouped into single (399 queries) and multiple (436 queries), according to the number of triple patterns. The former group tests the triple matching capabilities of the systems, whereas the latter group tests join performance. In order to create the synthetic graph pattern queries, we selected 17 different complex join patterns (based on [14]), then we generated 50 different queries for each pattern, resulting in a total of 850 queries. These synthetic queries were designed to test the performance of worst-case optimal joins. For evaluating path matching, we used 1,683 queries involving regular path expressions (i.e. 2RPQ queries). These queries were extracted from a log of SPARQL queries that produced timeouts on the Wikidata endpoint [15].

To simulate a realistic behaviour, we did not split queries into cold/hot run segments. Rather, we ran queries in succession, one after another, after a cold start of each system, and after cleaning the OS cache. We recorded the execution time of each individual query, which includes iterating over all results, and set a limit of 100,000 distinct results for each one. Additionally, we defined a timeout of 10 minutes per query for each system.

In the evaluation of graph pattern queries, MillenniumDB was the fastest, followed by Virtuoso, JenaLF, Blazegraph, JenaTDB and Neo4j (this considering the best average time each). Specifically, for single queries, MillenniumDB obtained an average time of 0.07s (the best time for graph pattern queries), and Blazegraph was the second with 2.21s. In the case of real-world multiple queries, MillenniumDB-LF obtained an average time of 4.84s, followed by Virtuoso with 7.87s, and Neo4j was the last one with 75.55s (the worst time for graph pattern queries). For synthetic queries, MillenniumDB-LF obtained an average time of 0.38s, followed by JenaLF with 0.88.

In the case of path queries, MillenniumDB shown the lowest times, followed by Virtuoso, Jena, Neo4j, and BlazeGraph. Specifically, both MillenniumDB-BFS and MillenniumDB-DFS obtained an average time of 1.1s, Virtuoso was the second with 5.8s, and Blazegraph was the last one with 27.6s. It is important to mention that MillenniumDB and Neo4J were the only systems able to handle timeouts without being restarted. Moreover, Blazegraph, Jena and Virtuoso reported execution errors during path query evaluation.

3 DEMONSTRATION

The goals of this demonstration are twofold. First, we aim to show how MillenniumDB addresses the three fundamental requirements presented in the introduction: how it handles multi-modal data, how it incorporates support for different data models and query languages, and how its design pushes the boundaries of state-of-the-art performance in query evaluation. Second, the demonstration is designed to provide attendees with a hands-on experience with MillenniumDB. Specifically, during the demonstration, attendees will learn about our datasets and services, retrieve interesting information in a variety of domains, get an idea of how MillenniumDB works, and understand how it could be used for their research or applications.

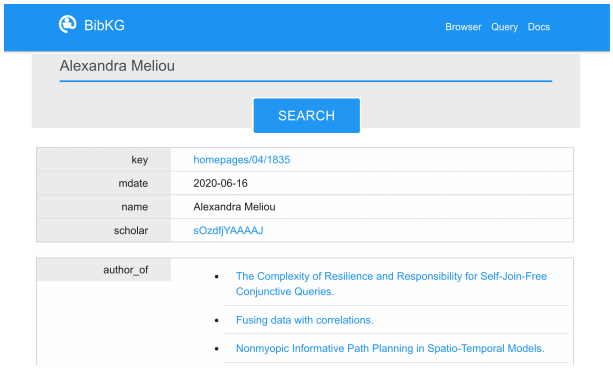

Figure 2: Exploring scientific contributions of Alexandra Meliou with BibKG, powered by MillenniumDB. Data is queried using a property graph query language.

3.1 The Experience

For each dataset, participants will be able to either access the endpoints through their web browser, or by navigating in a dedicated computer. Endpoints carry the following information:

- A brief description of the dataset, how data was gathered, and what it contains.
- A description of the schema of the graph.
- A wide variety of query examples they can try out.
- A white-box interface where users can alter examples or try out new queries on their own.

Learning about the internals of the database. Additionally, for the demonstration, a selection of example queries will be linked with a description of the logical plan, the physical plan, and the search process that generated these plans. This will give users more knowledge on how their queries were evaluated, whether traditional or worst-case optimal joins were used, and how was the semantic similarity query processed, if applicable.

Following the same idea, our dedicated computer will have a version of the endpoint for which the console will show, in real time, the plans selected for this query. Our team will then be able to explain, in real time, the process used to select the best alternative for evaluating the query, and the technology used by the engine.

Finally, attendees will be encouraged to download and try out MillenniumDB on their personal computers. The documentation and installation instructions are currently available at our documentation page (https://mdb.imfd.cl/doc/).

3.2 The Datasets

Users will be able to query and explore the following three knowledge graphs. Please note that these endpoints are currently maintained under the assumption that they will receive a light number of requests. Prior to the demo these will be set up in an environment ready to accept and coordinate a larger load of requests.

BibKG. This is a bibliographic knowledge graph constructed by integrating DBLP (https://dblp.org) and ArnetMiner (https://www.aminer.org/). It contains full information about publications, scientists and their fields, and is an ideal database to grasp the concept of browsing. Figure 2 shows a possible pathway users may engage with when browsing the graph: they would start searching

Figure 3: Querying for the material that a certain member of the Chilean Constitutional Convention voted in favor with TelarKG, powered by MillenniumDB. Data is queried using a property graph query language, and this text is also subject to be queries for semantic similarity.

for a particular author, then browse their papers, and continue from there. Users in this endpoint will be encouraged to run so-called triangle queries, and other complex graph queries, so that they can experience for themselves how our engine selects between traditional and worst-case optimal joins for their query plans. The endpoint is currently available at https://bibkg.imfd.cl/.

TelarKG. This is a political knowledge graph containing information about the (first) recent Chilean constitutional process. This knowledge graph contains information in different modalities: everything that the members of the constitutional convention stated and voted for in plenary sessions, including both video and transcripts of the sessions, plus their social network trace, and further public social and media information that we retrieved during the process. Figure 3 shows how users can query and interact with text data, in this case looking for the text that a particular convention member voted in favor of. Furthermore, users of this endpoint will be encouraged to try out our semantic similarity functionalities, allowing them to query, for example, for all videos in which a particular member of the convention spoke about a subject. The endpoint is currently available at https://telarkg.imfd.cl/.

Wikidata. This is a knowledge graph natively available in RDF format. We use this to showcase how MillenniumDB can handle different data models and/or query languages (in this case RDF/SPARQL). This endpoint corresponds to a snapshot of the entire Wikidata database [10], and is currently available at https://wikidata.imfd.cl/. Figure 4 shows the querying page of this endpoint, where nuances of SPARQL, such as prefixes, are provided to help users write their queries. The SPARQL query in this figure is a triangle pattern, which MilleniumDB evaluates efficiently using worst-case optimal joins.

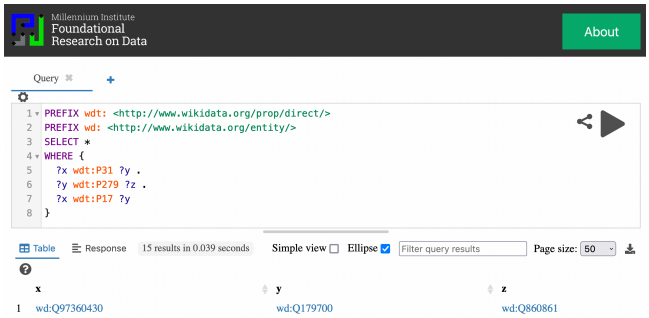

Figure 4: Wikidata endpoint powered by MillenniumDB. The endpoint uses SPARQL for querying and a YASGUI frontend.

REFERENCES

[1] Renzo Angles. 2018. The Property Graph Database Model. In *Proc. Alberto Mendelzon International Workshop on Foundations of Data Management (AMW)*, Vol. 2100. CEUR Workshop Proceedings.

[2] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoč. 2017. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.* 50, 5 (2017). https://doi.org/10.1145/3104031

[3] Renzo Angles, Aidan Hogan, Ora Lassila, Carlos Rojas, Daniel Schwabe, Pedro A. Szekely, and Domagoj Vrgoc. 2022. Multilayer graphs: a unified data model for graph databases. In *GRADES-NDA'22*. ACM, 11:1–11:6. https://doi.org/10.1145/3534540.3534696

[4] Diego Arroyuelo, Aidan Hogan, Gonzalo Navarro, Juan L Reutter, Javiel Rojas-Ledesma, and Adrián Soto. 2021. Worst-case optimal graph joins in almost no space. In *Proceedings of the 2021 International Conference on Management of Data*. 102–114.

[5] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. 2005. LSH forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web*. 651–660.

[6] World Wide Web Consortium et al. 2013. SPARQL 1.1 overview. (2013).

[7] World Wide Web Consortium et al. 2014. RDF 1.1 concepts and abstract syntax. (2014).

[8] Orri Erling. 2012. Virtuoso, a Hybrid RDBMS/Graph Column Store. *IEEE Data Eng. Bull.* 35, 1 (2012), 3–8. http://sites.computer.org/debull/A12mar/vicol.pdf

[9] Alin Deutsch et. al. 2022. Graph Pattern Matching in GQL and SQL/PGQ. In *SIGMOD '22*. https://doi.org/10.1145/3514221.3526057

[10] The Wikimedia Foundation. 2021. Wikidata:Database download. https://www.wikidata.org/wiki/Wikidata:Database_download

[11] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An Evolving Query Language for Property Graphs. In *SIGMOD 2018*. https://doi.org/10.1145/3183713.3190657

[12] Michael J. Freitag, Maximilian Bandle, Tobias Schmidt, Alfons Kemper, and Thomas Neumann. 2020. Adopting Worst-Case Optimal Joins in Relational Database Systems. *Proc. VLDB Endow.* 13, 11 (2020), 1891–1904. http://www.vldb.org/pvldb/vol13/p1891-freitag.pdf

[13] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. 2022. Knowledge Graphs. *ACM Comput. Surv.* 54, 4 (2022), 71:1–71:37. https://doi.org/10.1145/3447772

[14] Aidan Hogan, Cristian Riveros, Carlos Rojas, and Adrián Soto. 2019. A Worst-Case Optimal Join Algorithm for SPARQL. In *18th International Semantic Web Conference*. Springer, 258–275. https://doi.org/10.1007/978-3-030-30793-6_15

[15] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. 2018. Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph. In *ISWC 2018*.

[16] Dung Nguyen, Molham Aref, Martin Bravenboer, George Kollias, Hung Q Ngo, Christopher Ré, and Atri Rudra. 2015. Join processing for graph patterns: An old dog with new tricks. In *Proceedings of the GRADES'15*. 1–8.

[17] Amazon Neptune Team. 2021. What Is Amazon Neptune? https://docs.aws.amazon.com/neptune/latest/userguide/intro.html

[18] Jena Team. 2021. Jena TDB. https://jena.apache.org/documentation/tdb/

[19] TigerGraph Team. 2021. TigerGraph Documentation – version 3.1. https://docs.tigergraph.com/

[20] Bryan Thompson, Mike Personick, and Martyn Cutcher. 2014. The Bigdata® RDF Graph Database. In *Linked Data Management*. Chapman and Hall/CRC, 193–237.

[21] Todd L. Veldhuizen. 2014. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*. 96–106. https://doi.org/10.5441/002/icdt.2014.13

[22] Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[23] Domagoj Vrgoč, Carlos Rojas, Renzo Angles, Marcelo Arenas, Diego Arroyuelo, Carlos Buil-Aranda, Aidan Hogan, Gonzalo Navarro, Cristian Riveros, and Juan Romero. 2023. MillenniumDB: A Persistent, Open-Source, Graph Database. *Data Intelligence* 5 (3) (2023), 560–610. https://doi.org/10.1162/dint_a_00229

[24] Jim Webber. 2012. A programmatic introduction to Neo4j. In *SPLASH '12*, Gary T. Leavens (Ed.). https://doi.org/10.1145/2384716.2384777