M2Bench: A Database Benchmark for Multi-Model Analytic Workloads

Bogyeong Kim
Seoul National University
Seoul, Republic of Korea
bgkim@dbs.snu.ac.kr

Kyoseung Koo
Seoul National University
Seoul, Republic of Korea
koo@dbs.snu.ac.kr

Undraa Enkhbat
Seoul National University
Seoul, Republic of Korea
undraae@dbs.snu.ac.kr

Sohyun Kim
Seoul National University
Seoul, Republic of Korea
chloek409@dbs.snu.ac.kr

Juhun Kim
Seoul National University
Seoul, Republic of Korea
johnjhkim@dbs.snu.ac.kr

Bongki Moon
Seoul National University
Seoul, Republic of Korea
bkmoon@snu.ac.kr

ABSTRACT

As the world becomes increasingly data-centric, the tasks dealt with by a database management system (DBMS) become more complex and diverse. Compared with traditional workloads that typically require only a single data model, modern-day computational tasks often involve multiple data sources and rely on more than one data model. Unfortunately, however, there is currently no standard benchmark program that can evaluate a DBMS in the various aspects of multi-model databases, especially when the array data model is concerned. In this paper, we propose *M2Bench*, a new benchmark program capable of evaluating a multi-model DBMS that supports several important data models such as relational, document-oriented, property graph, and array models. *M2Bench* consists of multi-model workloads that are inspired by real-world problems. Each task of the workload mimics a real-life scenario where at least two different models of data are involved. To demonstrate the efficacy of *M2Bench*, we evaluated polyglot or multi-model database systems with the *M2Bench* workloads and unfolded the diverse characteristics of the database systems for each data model.

PVLDB Reference Format:
Bogyeong Kim, Kyoseung Koo, Undraa Enkhbat, Sohyun Kim, Juhun Kim, and Bongki Moon. M2Bench: A Database Benchmark for Multi-Model Analytic Workloads. PVLDB, 16(4): 747 - 759, 2022.
doi:10.14778/3574245.3574259

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/snu-dbs/m2bench.

1 INTRODUCTION

The relational model has been the dominant data model for decades due to its wide applicability in traditional business enterprises [13]. Since the emergence of the Internet era, however, data sources and types have become more diverse, and the relational model is no longer a one-size-fits-all solution to increasingly complex data management needs. As a result, a variety of alternative data models have arisen. The document-oriented model has emerged and demonstrated its success in the NoSQL industry to manage semi-structured data [9, 14, 45]. The graph data model has been commonly used for representing knowledge graphs and social networks [1]. The array data model, which has been used to represent scientific data, is gaining popularity in data analytics due to its applicability in machine learning [4, 6, 19, 31, 38]. Evidently, the tasks and workloads a database management system (DBMS) is expected to deal with have become more complex and diverse.

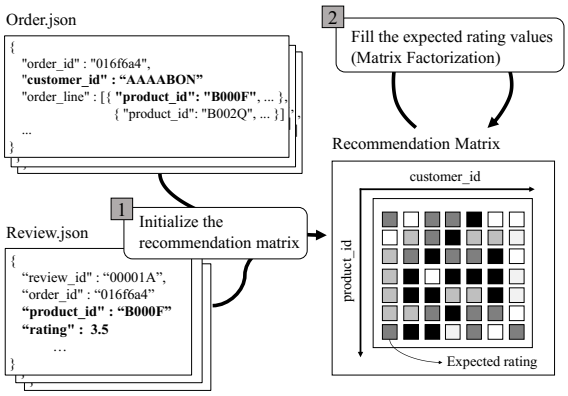


Figure 1: An example of a multi-model task

Unlike traditional data management tasks based on a single data model, recent workloads often require the management and processing of multiple models of data simultaneously. For example, a recommendation system relies on data stored in the JSON format and computes rating values in a matrix to recommend products to customers. As is shown in Figure 1, the attributes in `Order` and `Review` (in the document model) are selected and joined to initialize the `Recommendation matrix` (in the array model). Then, the expected rating values can be filled in via matrix factorization. Thus, the underlying DBMS needs to support database workloads based on both the document model and the array model. The demand for multi-model data management is already around us, and we anticipate that it will be even more prevalent in the future.

Proceedings of the VLDB Endowment, Vol. 16, No. 4 ISSN 2150-8097.
doi:10.14778/3574245.3574259

However, there is no standard benchmark program that can test the performance of a DBMS for handling multiple models of data. Most of the existing benchmarks do not encompass multiple data models. Popular benchmarks such as TPC-H and TPC-DS [41] assume only the relational data model to provide E-Commerce analytic tasks. XMark [34] and NOBENCH [12] assume only the document data model and consist of basic NoSQL queries (*e.g.*, selection, projection, aggregation) or path-finding queries over the document hierarchy. LinkBench [3], GooDBye [26], and LDBC-SNB [17] assume the graph data model and consist of analytic queries over the graphs as well as simple CRUD operations (*i.e.*, Create, Read, Update, and Delete). SS-DB [15] and GenBase [39] assume the array data model, and they consist of queries for manipulating arrays. Each of these benchmark programs confines its query set to a certain data model.

There are indeed a few benchmark programs that assume multiple data models, but the models they support are rather limited. For example, BigBench [22] covers the semi-structured and unstructured data as well as the relational data model, but it does not support the graph data model or the array data model. UniBench [51] is a recent benchmark proposed for a multi-model DBMS. However, it does not support the array data model, which is essential and common for machine learning and scientific data analytics. Given that there are many practical use cases where the array data model is used along with other data models [18, 25, 44, 48], the lack of support for the array data model is a non-trivial limitation of these benchmark programs.

In this paper, we propose *M2Bench*, a new benchmark program for multi-model databases. *M2Bench* provides a set of complex analytic tasks encompassing four data models (relational, document, graph, and array) and a data generator. For the model complexity pertaining to workloads, each task is designed to involve at least two different data models. In addition, to cover diverse workloads from distinct fields, each task is designed to fall into one of the three domain categories: E-Commerce, Healthcare, and Disaster & Safety. The domain category, data models, and key operations associated with each task are summarized in Table 1. The design objective of *M2Bench* is to help users understand the impact of a data model on the performance of a multi-model database system. The full description of our workloads, data generator, and implementations are available in our public repository [36].

In order to demonstrate the efficacy of *M2Bench*, we tested three multi-model database systems, namely, Polyglot persistence [46], ArangoDB [2], and AgensGraph [5]. The *M2Bench* was able to evaluate the overall performance of the database systems quantitatively and comparatively on the multi-model workloads and offer explanations why a certain system might be more appropriate for certain tasks. Besides, the impact of native storage engines was analyzed for individual tasks to better understand the performance characteristics of the multi-model database systems.

The paper is organized as follows. In Section 2, the overview of *M2Bench* is presented. In Section 3, the characteristics of *M2Bench* are covered. In Section 4, the workload of *M2Bench* is introduced along with schemas of data. In Section 5, the data generator of *M2Bench* and the source of the dataset are covered. In Section 6, we present the experimental results. Lastly, the related works and the conclusion are covered in Sections 7 and 8, respectively.

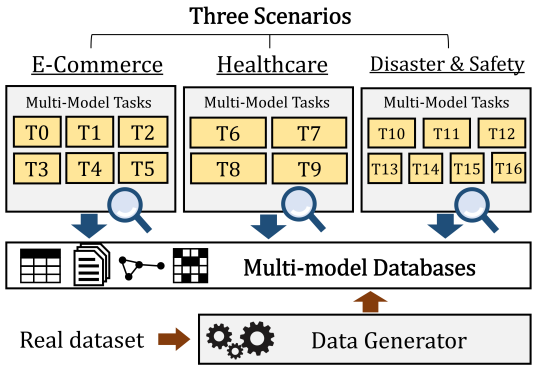


Figure 2: *M2Bench* overview

2 *M2BENCH* OVERVIEW

This section briefly overviews the main components of *M2Bench*. To cover the diverse and realistic workloads, *M2Bench* provides a multitude of *tasks* divided into three domain categories (or *scenarios*), namely, *E-Commerce, Healthcare*, and *Disaster & Safety*.

(1) **E-Commerce:** This is one of the most utilized domains in benchmark studies. It analyzes data of online commercial systems and provides insights into profit generation.
(2) **Healthcare:** This is an important research field directly related to human health. Many studies have been conducted to extract information embedded in biomedical and healthcare datasets to find better ways to prevent and treat diseases.
(3) **Disaster & Safety:** This field is related to environmental problems, such as earthquakes and fine dust. Monitoring these data is becoming increasingly important due to the dangerous and unpredictable nature of disasters.

As is shown in Figure 2, each *scenario* consists of a set of *tasks*, each of which is a series of multi-model queries producing an analytic result. A task is designed to involve at least two different data models for the model complexity. *M2Bench* provides a total of seventeen multi-model tasks inspired by real-world problems. The detailed descriptions of individual tasks are provided in Section 4.

The data generator of *M2Bench* produces multi-model base data from real-world datasets and then scales up the base data by a chosen scale factor. Different methods are adopted to scale up the base data of different data models. Refer to Section 5 for the details.

The basic benchmarking steps are as follows. First, datasets produced by the data generator are loaded into a DBMS by executing a provided script. Second, a set of provided queries are executed by the DBMS. Lastly, the query executions are analyzed under a given set of profiling rules. Three sets of query scripts are currently provided – one for each of the polyglot (composed of MySQL, MongoDB, Neo4j, and SciDB), ArangoDB, and AgensGraph systems.

3 CHARACTERISTICS OF *M2BENCH*

3.1 Data Models

The most commonly used data models for data analytics lately include the relational model, the document-oriented model, the property graph model, and the array model. Many applications can represent their data in a combination of those four data models.

Table 1: Characteristics of *M2Bench* tasks

Categories	*Tasks*	*Relational*		*Document*				*Graph*				*Array*		
		Sel	Agg	Sel	Agg	Unw	Dot	Sel	Agg	Pat	S.P	Sel	Agg	L.A
E-Commerce	T0	✓	✓	✓				✓		✓				✓
	T1	✓	✓	✓	✓	✓	✓							
	T2			✓	✓		✓							✓
	T3	✓	✓	✓	✓	✓	✓	✓		✓				
	T4	✓	✓	✓		✓	✓	✓	✓	✓				
	T5	✓		✓			✓	✓		✓				
Healthcare	T6	✓		✓	✓	✓	✓							
	T7	✓	✓					✓	✓	✓				
	T8	✓				✓	✓	✓	✓	✓				
	T9	✓	✓			✓	✓					✓		✓
Disaster & Safety	T10	✓		✓			✓	✓		✓				
	T11	✓		✓	✓		✓	✓			✓			
	T12	✓	✓	✓	✓		✓	✓			✓			
	T13	✓		✓	✓		✓							
	T14			✓	✓		✓					✓	✓	
	T15							✓			✓	✓	✓	
	T16			✓	✓	✓	✓					✓	✓	

Sel: selection, **Agg**: aggregation, **Unw**: unwind, **Dot**: nested attribute, **Pat**: pattern matching, **S.P**: shortest path, **L.A**: linear algebra

M2Bench also represents a database in a combination of those four data models. This section briefly describes the data models focusing on two aspects: *data representation* and *data manipulation*.

3.1.1 ***Data Representation.*** Each data model represents data in different forms. The relational data model represents data as tables. Each table consists of a set of records, each of which consists of a fixed number of attributes. The tables can be connected to other tables via the primary-foreign key relationships.

The document (oriented) model stores the name and value pair of an attribute similarly to a key-value pair. A single document stores one or more of these pairs in a more flexible way than the relational model. While all the records in a relational table have the same number of attributes, the documents in a collection may have a varying number of attributes. Moreover, an attribute can be another (nested) document or an array of attributes.

In the (property) graph model, data objects and their relationships are represented by the vertices and edges of a graph. The vertices and edges can have a set of attributes. The edges additionally contain information about their source and destination vertices. Social networks and road networks are often represented by the graph model.

In the array model, each data point is represented as a cell in an array, which is composed of dimensions and attributes. For example, a satellite image can be represented as an array by mapping each pixel to a cell in the array, where the coordinates and the RGB colors of the pixel are considered its dimensions and its attributes, respectively.

In *M2Bench*, we assume that data objects in one model can be linked to data objects in another model via what we refer to as the *inter-model primary-foreign key relationships*.

3.1.2 ***Data Manipulation.*** The key operations supported by the data models are summarized in Figure 3. The relational data model supports operations to manipulate table data. The relational data

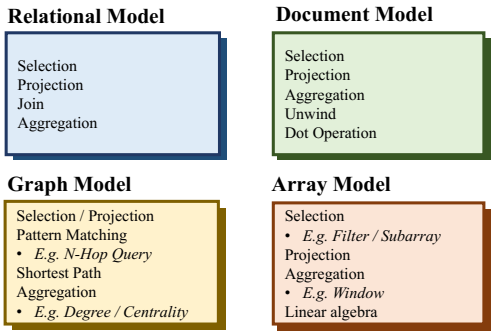


Figure 3: Data models and key operations of *M2Bench*

model and the document data model have a few common operations due to the similarities in their data representation. The document model, however, provides its own unique operations such as *unwind* (or *unnest*) and *dot*. The *unwind* operation supports processing of an array of attributes,[1] and the *dot* operation supports accessing nested attributes. The graph model supports *pattern matching* operations to facilitate graph exploration. Besides, various advanced operations (*e.g.*, a shortest path query) can also be supported by the graph model. The array data model supports operations for manipulating arrays such as *subarray* and *filter* and linear algebra operations for processing matrix data.

The tasks of *M2Bench* are designed to cover all those key operations in Figure 3 so that a multi-model database system can be evaluated effectively and thoroughly. Table 1 presents the characteristics of all the tasks with respect to their data models, key operations, and domain categories. The check marks in the table

[1]While traditional RDBMSes do not define or support the ARRAY data type, some of the recent systems actually support the *unwind (unnest)* operation.

indicate the specific key operations involved in a certain task. The *projection* and *join* operations are omitted in the table, as they are commonly used in all the data models.

Table 1 clearly shows that the tasks of *M2Bench* reflect the diverse workloads arising from the real-world multi-model applications. For example, task `T0` models an e-commerce application that involves all four data models to carry out selection, aggregation, pattern matching, and linear algebra operations. The performance impact of individual data models can be identified by referencing Table 1 and profiling the execution results. Profiling can break down the total elapsed time of a task into the times consumed separately by different data models. The profiling rules adopted for *M2Bench* will be described in Section 6.3.

3.2 Domain Categories for Diverse Workloads

M2Bench selects three domain categories to design workloads with diverse characteristics. In the E-Commerce scenario, tasks are based heavily on a large document collection called *Orders*. Consequently, efficient query processing is more important for data in the document data model than those in the other models. *Matrix factorization* as an advanced array operation is also included, which requires a large volume of matrix multiplication.

The Healthcare scenario is mainly based on network datasets that represent the relationships between drugs and diseases. Thus, the tasks in this scenario utilize *pattern matching* queries to extract advanced information about drugs and diseases (*e.g.*, interacting drugs, similar diseases). Besides, an array operation, *cosine-similarity*, is also included to calculate drug similarities.

The Disaster & Safety scenario is based on geospatial and raster datasets. The tasks in this scenario include *shortest path* queries as well as conventional geospatial queries utilizing spatial indexes. In regard to the array model, the tasks in this scenario do not require linear algebra operations but often rely on other array operations such as *window* and *subarray* to analyze raster data.

4 WORKLOAD

Each benchmarking scenario of *M2Bench* consists of a set of tasks, each of which functions as a standard workload for a single unit of analysis (*e.g.*, finding the groups of patients with similar diseases). Though each task can be expressed as a single query, we break it down to a series of smaller queries for higher readability. Consequently, the queries of a task are processed sequentially, and each query result is used as an input to the next query of the task. All tasks of *M2Bench* are briefly summarized in Table 2.

It would be best for clarity to use a standard query language to express the tasks. However, there does not exist such a standard query language that covers all the four data models as yet. To get around this limitation, we opt to augment the Structured Query Language (SQL) by adopting a few essential features from existing non-relation query languages or database systems. In particular, the *dot* notation (.) and the *unnest* syntax are used to access nested attributes in document data [27]. Some of the CYPHER language features such as *match* and *pattern* are used to express pattern matching queries over graph data [5, 20, 28]. Besides, the Array Functional Language (AFL) of SciDB [38] is used to express the manipulation of array data. The grammar of each language is not strictly followed unless it gives rise to ambiguity. Some of the key syntactic structures and functions adopted in *M2Bench* are listed in Table 3. Readers can refer to the public archive of *M2Bench* for more detailed explanations [37].

Table 2: Tasks in *M2Bench*

	Tasks	*Description*
E-Commerce	T0	Build a logistic regression model to predict if a user prefers the given brand.
	T1	Compute sales performance of each product of a top-selling Brand.
	T2	Perform product recommendations based on past customer ratings.
	T3	Analyze purchase propensities of people within a 2-hop social network distance from the customer who made the highest number of purchases.
	T4	Find the interests of the top-N famous customers who made more than a certain number of orders from a given product category.
	T5	Extract the social network graph of female customers who purchased the given product and wrote a review within the one year from the given date.
Healthcare	T6	Find drugs that interact with the prescribed drugs of a given patient.
	T7	Find the number of female and male patients suffering from similar diseases.
	T8	Find all potential interaction drugs which share the same targets as the given patient's prescription drugs.
	T9	For the prescription of a given patient, find similar drugs based on the adverse effects.
Disaster & Safety	T10	Find the road network sub-graph within 5km from the earthquakes' location.
	T11	Find the cost of the shortest paths for each GPS coordinate and shelter pair.
	T12	For a shelter near a populous region during the given time interval, find the five closest buildings from the shelter.
	T13	For earthquakes of magnitude 4.5 or above, find the statistics of buildings near the earthquake.
	T14	Find the nearest building from a finedust hot spot for each date between two given timestamps.
	T15	Find the shortest path from the current coordinates to a hotspot of finedust between two given timestamps.
	T16	For a given timestamp, hindcast the pm10 values of the schools.

The database schemas of the benchmarking scenarios are illustrated in Figures 4-6. The primary keys (PK) and foreign keys (FK) of a database are annotated next to the corresponding attributes in the figures. The *inter-model primary-foreign key relationship* is specified by a line linking the corresponding attributes. For example, in Figure 4, the *customer_id* attributes are common in the *Customers* table and the *Orders* collection, and they are linked by a line to indicate the one-to-many relationship between the two datasets. Each database consists of scalable datasets and unscalable ones, which are clearly distinguished in the corresponding schema diagram. A detailed description of the database schema and a sample task is presented for each benchmarking scenario in this section.

Table 3: Key syntactic structures and functions adopted in *M2Bench*

Syntax/Function	Description
pattern: `(a:A)-[r:R]-(b:B)`	Node `a` is linked with node `b` through the edge `r`. `A`, `R` and `B` denote the label or field name. Directed edge can be represented using " $-[r:R]\rightarrow$ " instead of " $-[r:R]-$ " .
MATCH *pattern* WHERE *condition* RETURN *variables*	Match all *patterns* satisfying the *condition* in the graph and return a set of *variables* in the matched *patterns*.
`Unnest`	If a document or object contains a nested array, perform a join of the nested array with its parent object and return the joined object.
`Dot(.)`	Return the value of the corresponding nested field.
`toArray(`d_1`,`d_2`,...,`d_n`,` A_1`,`A_2`,...,`A_n`)`	Convert an object to an array of which size is (d_1,d_2,...,d_n) having A_1,A_2,..., A_n as attributes.
`MatMul(`m_1`,` m_2`)`	Matrix multiplication on two matrices m_1 and m_2.
`Factorization`	Perform the matrix factorization and produce two matrices, m_1 and m_2, containing the latent factors.
`ST_ClosestObject(`d`,` o`,` c`)`	In the dataset d, find the object o of which coordinate is closest from the coordinate c.
`ShortestPath(`G`,` s`,` e`)`	Find the shortest path and its cost in the graph G with start node s and end node e.
`OVER WINDOW (`d_1`,` d_2`, ...,` d_n`)`	Calculate window aggregation with a size of $d_1 \times d_2 \times \cdots \times d_n$.

4.1 E-Commerce *Scenario*

The database of the E-Commerce scenario includes customer and order information and is analyzed to gain insight into profit generation. The social network data are also explored to utilize the relationship among the customers. The workload in this scenario consists of six multi-model tasks, from `T0` to `T5`, which are inspired by the previous studies [8, 25, 41, 51]. Below are the database schema adopted in this scenario and the description of task `T2` given as an example.

Database Schema. The multi-model schema of the E-Commerce database is shown in Figure 4. The relational data model includes the *Customers*, *Products*, and *Brands* tables. The document model includes the *Orders* and *Reviews* document collections. Lastly, the graph model includes the *Social Network* property graph, which is composed of nodes (*persons* and *hashtags*) and edges (*follows* and *interests_in*). The *Customer* table, the *Orders* and *Reviews* collections, and the *Social Network* graph are scalable and annotated with their relative cardinalities in Figure 4. For example, the *Customers* table contains 9,949 rows when the scale factor is one ($SF = 1$).

Example 1 (**T2. Product Recommendation**). *This task involves the generation of a product recommendation matrix based on the past customer ratings.*

```
Q1. A = SELECT customer_id, product_id, rating
          FROM Reviews, Orders
          WHERE Reviews.order_id = Orders.order_id

Q2. B, C = A.toArray(dim1: customer_id, dim2: product_id,
                     val: avg(rating))
              .Factorization

Q3. D = MatMul(B,C)
```

Task `T2` *involves two data models, the document and array models, and it consists of three queries (Q1, Q2, and Q3) that are executed in sequence. Q1 fetches a set of tuples (customer_id, product_id, rating) from the document collections Orders and Reviews. Q2 first transforms the result of Q1 to an array by setting the customer_id and product_id as dimensions and the average of ratings as the attribute value of an array cell. Q2 then factorizes the array to produce two matrices, B and C. Finally, Q3 returns a recommendation matrix D by multiplying the two matrices B and C.*

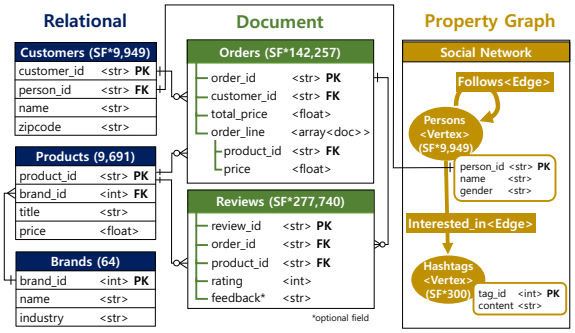


Figure 4: E-Commerce schema

4.2 Healthcare *Scenario*

The database of the Healthcare scenario includes Electronic Health Record (EHR) data as well as drug and disease data. The biomedical information is extracted and analyzed to provide insight into the medical records. The workload in this scenario consists of four multi-model tasks from `T6` to `T9`, which are derived from the previous studies of common use cases in healthcare [18, 33, 44].

Database Schema. The multi-model schema of the Healthcare database is shown in Figure 5. The relational data model includes the *Patients*, *Prescriptions*, and *Diagnoses* tables. The graph model includes the *Disease Network* property graph. Lastly, the document model includes the *Drugs* document collection. Only the relational tables are scalable in the database, and they are annotated with their relative cardinalities in Figure 5.

Example 2 (**T7. Patients with similar diseases**). *Given the diseases of patient_id X, this task finds the number of female and male patients suffering from similar diseases.*

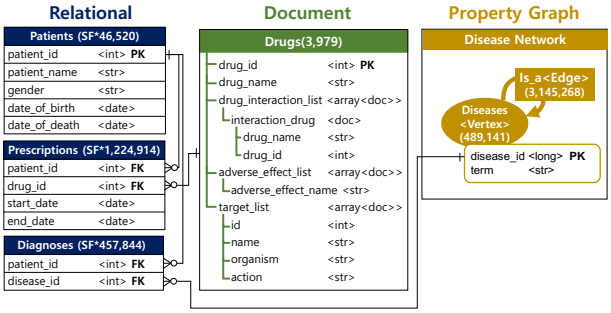


Figure 5: Healthcare schema

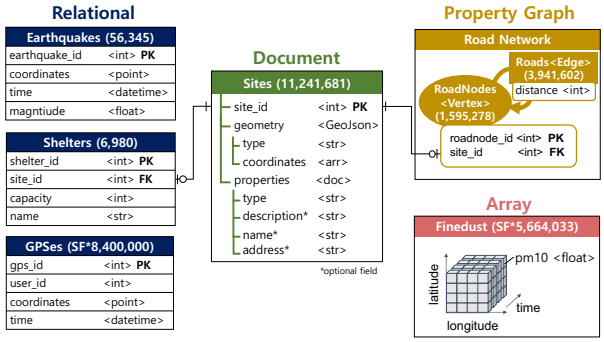


Figure 6: Disaster & Safety schema

```
Q1. A = SELECT disease_id
        FROM Diagnoses
        WHERE patient_id = "X"

Q2. B = SELECT distinct(d3.disease_id)
        FROM A,
        (MATCH (d1: Diseases)-[:is_a]->(d2: Diseases)
         <-[:is_a]-(d3: Diseases) RETURN d1, d2, d3)
        WHERE A.disease_id = d1

Q3. C = SELECT distinct(patient_id)
        FROM B, Diagnoses
        WHERE Diagnoses.disease_id = B.disease_id
          AND Diagnoses.patient_id != "X"
          AND B.disease_id NOT IN A

Q4. D = SELECT gender, count(gender)
        FROM Patients, C
        WHERE Patients.patient_id = C.patient_id
        GROUP BY gender
```

Task T7 *consists of four queries from Q1 to Q4. For a given patient_id X, Q1 fetches disease_ids from the Diagnoses table. Q2 then finds diseases that are in the sibling relationship with the disease_ids in the Disease Network graph. Note that the subquery of Q2 in the CYPHER language returns a set of matched node tuples (d1, d2, d3) of similar diseases. Then, Q3 finds the patients who suffer from the diseases returned by Q2. Lastly, Q4 aggregates the patients from Q3 by gender. The relational and graph models are involved in this task.*

4.3 Disaster & Safety *Scenario*

The database of the Disaster & Safety scenario includes earthquake event data as well as shelter and road network information so that it can support an earthquake response system and a fine dust alert system. The workload for an earthquake response system includes filtering the road network, finding the shortest paths, and searching for specific buildings. It consists of four multi-model tasks from T10 to T13. The workload for a fine dust alert system includes finding the sources of fine dust and providing route recommendations for the cleaning vehicles [24, 48, 49]. It consists of three multi-model tasks from T14 to T16.

Database Schema. The multi-model schema of the Disaster & Safety database is shown in Figure 6. The relational data model includes the *Shelters*, *Earthquakes*, and *GPSes* tables. The document model and the graph model include the collection *Sites* and the *Road Network* property graph, respectively. Lastly, the array model includes the *Finedust* array. The *GPSes* table and the *Finedust* array are scalable in the database, and they are annotated with their relative cardinalities in Figure 6.

Example 3 (**T15. Fine Dust Cleaning Vehicles**). *Given a time interval [ts1, ts2] and the coordinates of a certain location* X*, this task finds the shortest path from the location to a hotspot in the Finedust array within the time interval.*

```
Q1. A = SELECT (latitude, longitude) AS coo,
                    avg(pm10) AS pm10_avg
          OVER WINDOW (*, 5, 5)
          FROM FineDust
          WHERE timestamp >= ts1 AND timestamp <= ts2

Q2. B = SELECT ShortestPath(
           RoadNetwork,
           ST_ClosestObject(Sites, RoadNodes, "X"),
           ST_ClosestObject(Sites, RoadNodes, A.coo))
          FROM A, RoadNetwork, Sites
          WHERE Sites.site_id = RoadNodes.site_id
          ORDER BY A.pm10_avg DESC
          LIMIT 1
```

Task T15 *consists of two queries, Q1 and Q2. Q1 first fetches a sub-array within the time interval [ts1, ts2] from the Finedust array. It then applies a window aggregation (e.g., PostgreSQL [40]) in order to compute the average* pm10 *values for the array cells within the* 5×5 *overlapping windows. Q2 locates a hotspot, a region that has the maximum pm10_avg value, in the result from Q1. It then finds the shortest path from the current location X to the hotspot over the Road Network graph. The array, graph, and document models are involved in this task.*

5 DATA GENERATOR

The data generator of *M2Bench* creates databases for multi-model benchmarking purposes. This section describes the process of constructing the database of scale factor one for each scenario and expanding it for larger scale factors. The databases generated for *M2Bench* are summarized in Table 4 along with the scalability of individual components.

5.1 Constructing Databases of Scale Factor One

M2Bench exploits public real-world datasets to build a realistic multi-model database. *M2Bench* also uses the data generation tools of existing benchmarks.

5.1.1 **E-Commerce Database.** We use the data generators of UniBench [51] and TPC-DS [41] to construct the scale factor one (SF1) E-commerce database. Specifically, the *Products* and *Brands* tables, the *Orders* and *Reviews* document collections, and the *Social Network* graph are constructed by adding and modifying the attributes of the UniBench datasets [51]. The *Persons* vertices of the *Social Network* graph and the *Customers* table are created by taking the corresponding data from the TPC-DS database.

The primary-foreign key relationships between different sources of data are established based on *Customers* and *Orders*. For example, a *person* node of the *Social Network* graph corresponds to a *person ID* of the *Customers* table, and a *customer ID* corresponds to the ID of the person who has made his or her orders in *Orders* collection.

5.1.2 ***Healthcare Database.*** The Healthcare database is created entirely from real-world datasets. The *Patients, Diagnoses* and *Prescriptions* tables are derived from MIMIC Critical Care [23], which is a collection of electronic health records associated with more than 40,000 patients in the Beth Israel Deaconess Medical Center. The *Drugs* documents are derived from Drugbank [47], and the *Disease Network* graph is derived from SNOMED [35]. Drugbank is a collection of detailed information about drugs such as side effects, drug-drug interactions, enzymes, and targets. SNOMED is a dataset for clinical terminology.

Data cleaning is necessary for the real-world datasets so that unused attributes are pruned out and some of the attribute names are conveniently aligned among the datasets. A script for data cleaning is provided in the public repository of *M2Bench* [36]. The primary-foreign key relationships between the different sources of data are established based on the linked IDs provided by the data sources or the attributes bearing the same name.

5.1.3 ***Disaster & Safety Database.*** The Disaster & Safety database is created partially from real-world datasets. The *Earthquakes* and *Shelters* tables and the *Sites* document collection are derived from the U.S. Geological Survey (USGS) [43], the Homeland Infrastructure Foundation Level Data (HIFLD) [42], and the OpenStreetMap (OSM) [29], respectively. The *Road Network* graph is derived from the 9th DIMACS Implementation Challenge [10]. The *GPSes* table and the *Finedust* array are generated synthetically based on real-world datasets. Specifically, the GPS coordinates of *GPSes* follow the population distribution of the Gridded Population of the World (GPW) dataset [11], and the *Finedust* array is generated from the atmospheric scans made by the Lidar equipment installed in Siheung, Korea [24] (with the coordinates remapped to the state of California, USA so that the *Finedust* array is placed in the same geographic location with the other aforementioned datasets). The primary-foreign key relationships between the different sources of data are established based on spatial proximity.

Table 4: Databases in *M2Bench*

Scenario	*Data Name*	*Source dataset*	*Scalability*
E-Commerce	Customers	TPC-DS [41]	✓
	Products	UniBench [51]	✓
	Brands	UniBench	
	Orders	UniBench	✓
	Reviews	UniBench	✓
	Social Network	TPC-DS & UniBench	✓
Healthcare	Patients	MIMIC-III [23]	✓
	Prescriptions	MIMIC-III	✓
	Diagnoses	MIMIC-III	✓
	Drugs	Drugbank [47]	
	Disease Network	SNOMED [35]	
Disaster & Safety	Earthquakes	USGS [43]	
	Shelters	HIFLD [42]	
	GPSes	GPW v4 [11]	✓
	Sites	OSM [29]	
	Road Network	DIMACS [10]	
	Finedust	MISE [24]	✓

5.2 Scaling Databases

The *M2Bench* databases can be scaled up by a given factor. Since not every component of the database is scalable, the scalable components of each database are clearly marked with a ✓symbol in Table 4. The detailed steps of the scaling-up process for each data model are depicted below.

5.2.1 ***Scaling Tables.*** When a table is scaled up, the value domain of the primary key (PK) attribute is expanded by a given scale factor. So is the value domain of a foreign key (FK) attribute referencing the PK of the table being scaled up. For example, if the *Patients* table is scaled up by a factor k, then each *Patients* tuple with the primary key p is duplicated to k tuples with their primary keys from $p \times k$ to $p \times k + (k - 1)$. If a table being scaled up contains a foreign key attribute referencing the PK of a non-scalable table (*e.g.*, categorical values), each PK value is repeated k times for the FK attribute. For value domains such as timestamps and emails, perturbed values are added to increase the cardinality of the domains.

5.2.2 ***Scaling Documents.*** Document collections are scaled up in the same manner as the tables regardless of their fields being nested or not. The number of documents increases by a given scale factor for a collection being scaled up. Conversely, the array fields of a document are not subject to scaling up and their lengths remain the same. In the E-Commerce database, for example, the *Orders* collection contains more documents as it gets scaled up, but the length of *orderline* in the document does not change.

5.2.3 ***Scaling Graphs.*** Most real world graphs including social networks grow denser over time with the number of edges growing super-linearly to the number of nodes. Thus, the (in/out) degree distribution is considered an important feature to be preserved by graph scaling. A graph scaling tool called EvoGraph is known to preserve various graph properties such as degree distribution [32]. Thus, for example, the *person-follows-person* graph of the *Social Network* can be scaled up by EvoGraph. However, EvoGraph cannot be

applied to bipartite graphs such as the *person-InterestedIn-hashtag* graph. Therefore, the *person-InterestedIn-hashtag* graph is scaled up by using our own generator. This graph is scaled up by increasing the degree of the *person* nodes linearly so that people become more interested in diverse hashtags as the scale factor increases.

5.2.4 ***Scaling Arrays.*** The *Finedust* array is scaled up by increasing the resolution of the time dimension. The time domain itself remains unchanged, but the time interval between two adjacent observations is sliced further so that the size of the array increases by a given scale factor. The scalar values stored in the array (*e.g.*, dust concentration measurements) are linearly interpolated between the values stored in two adjacent observations.

6 EVALUATION

To demonstrate the efficacy of *M2Bench*, all the tasks were implemented and evaluated on a few chosen multi-model database systems: Polyglot persistence [46], ArangoDB [2], and AgensGraph [5]. For analytic workloads, the elapsed time taken for executing an individual task is commonly used as an evaluation metric for benchmarking purposes. Therefore, in this section, we focus on the end-to-end elapsed time of an individual task executed by each database system. The databases were scaled-up with the *M2Bench* script to evaluate the scalability of the database systems.

6.1 Database Systems for Evaluation

6.1.1 ***Polyglot Persistence.*** A DBMS is said to *natively support* a data model X if the DBMS has a storage engine designed specifically for the model X. For example, SciDB's storage engine is designed to store array data. It provides efficient access to a group of adjacent array cells by storing them as a set of partitioned arrays called chunks. Polyglot persistence refers to a system that exploits multiple DBMSes to solve complex problems. In our evaluation, polyglot persistence relies on the storage engines of four popular DBMSes to perform multi-model tasks, namely, MySQL [30], MongoDB [27], Neo4j [28], and SciDB [38]. Each of them *natively* supports one of the four data models. A model-specific operation is assumed to be performed by the corresponding DBMS. For example, an aggregation of relational data is executed by MySQL.

To access data from multiple systems simultaneously, a coordinating client is created atop the DBMSes. The client is involved in collecting intermediate results from those DBMSes and processing them for the next intermediate or the final result. The client is assumed to have no memory space enough to store a large quantity of intermediate data. When an intermediate result is required by the next operation, the result may have to be written back to the DBMS that produced it and fetched again from the DBMS. For a join operation between different data models, the client orchestrates a nested loop join between the DBMSes.

6.1.2 ***ArangoDB.*** ArangoDB is a DBMS that natively supports two data models: the document model and the graph model. While JSON documents are stored in RocksDB [16] in its own internal format, graph data are stored in *almost* index-free adjacency storage. It is almost index-free in that a hash index is also provided in the form of modified adjacency storage. The relational and the array data of *M2Bench* are stored as documents in ArangoDB for evaluation. In particular, the array data are stored in the coordinate list format (or COO format in short), which represents an array cell x as an n-dimensional tuple $(d_1, \ldots, d_n, x)$ with its i^{th} coordinate denoted by d_i.

6.1.3 ***AgensGraph.*** AgensGraph is a DBMS that supports the relational model and the graph model. As an extension of PostgreSQL [40], AgensGraph natively supports the relational model. AgensGraph relies on an additional storage engine for graph data, but its internal architecture is similar to the relational storage engine. Thus, AgensGraph is not considered natively supporting the graph model. The document collections of *M2Bench* are stored in the JSONB format, and the array data are stored in the COO format.

6.2 Settings

All the experiments were conducted on a standalone machine with Intel i7-9700K CPU, 32GB RAM, and a 512GB SSD running Ubuntu 18.04.4 LTS. We configured the three chosen database systems to use the same amount of memory insofar as possible. For the polyglot system, 4GB memory was allotted as a page cache for each of the four underlying DBMSes and additional 8GB memory was allotted for the Java heap space of Neo4J following the recommendation of Neo4J. Similar to the polyglot system, a total of 24GB memory was allotted separately for AgensGraph and ArangoDB.[2]

Indexes were built on the same or closely related attributes on all the database systems. Queries were executed in a single thread, and the elapsed times were measured by taking the average of five runs per query. To avoid the cache effect, all the DBMSes were restarted before each run to flush the page cache. For the geospatial queries from T11 to T14, a new collection called *Site_centroid* was created to include the centroid values additionally. This new collection was used instead of the *Sites* collection because MongoDB did not provide a centroid function.

6.3 Baseline Evaluation

Figure 7 shows the elapsed times of all tasks performed by the three database systems, namely, the polyglot system, ArangoDB, and AgensGraph. (Refer to Table 2 for the description of tasks.) Each bar shows a measurement of elapsed time, and it is broken down into different colors to identify separately the times consumed for different data models.[3] Profiling was done by the following simple rules listed below.

(1) The time consumed by a model-specific operation (shown in Figure 3) accounts solely for the corresponding model.
(2) If two datasets of different models are joined or accessed simultaneously, the times consumed to fetch the datasets account separately for their corresponding data models.
(3) When the data model of intermediate data needs to be determined for profiling, the relational model is chosen if intermediate data do not have any nested attribute or any array attribute. Otherwise, the document model is chosen.

[2]For ArangoDB, three-quarters of the total memory was used by RocksDB.
[3]The legend "Others" indicates the elapsed time consumed by the client in the polyglot system or the elapsed time that cannot be classified into a specific model.

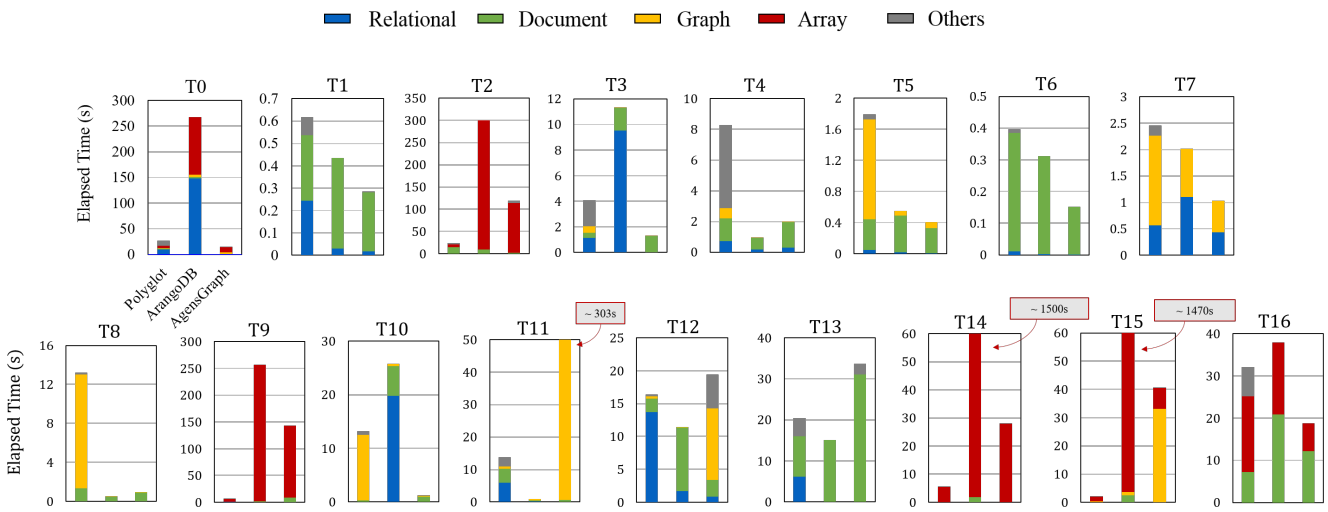


Figure 7: Baseline evaluation

For the polyglot system with a coordinating client, the connection delay between the client and each underlying DBMS is added to the elapsed time of the corresponding data model.

6.3.1 ***Polyglot Persistence.*** The polyglot system outperformed AgensGraph and ArangoDB for tasks that required intensive array computations (*e.g.*, T2, T9, T14, and T15). This is attributed to SciDB with a native storage engine that stores an array in chunks such that the locality of array cells is preserved. On the other hand, AgensGraph and ArangoDB store an array in a table and a collection, respectively, where each row or each document represents an array cell. Consequently, the locality of array cells is not preserved. With the lack of locality, array operations such as matrix multiplication would have to access randomly scattered rows, which causes excessive disk I/O operations. Task T16 also requires array operations, but the performance of the polyglot system was not so impressive for it. This is because T16 requires accessing array cells randomly and iteratively. This access pattern was not particularly beneficial for the polyglot system.

For the rest of the tasks, the performance of the polyglot system was not the best and it was actually much worse than that of the other database systems for a few tasks. This is due to what we call *polyglot latency*, which can take up a large portion of the elapsed time. It occurs when many data objects are fetched individually to the client, for example, to be joined with data from another database system. The polyglot latency was almost 42% of the total elapsed time of T1. The polyglot latency will be discussed in more detail in Section 6.4.

6.3.2 ***ArangoDB.*** ArangoDB outperformed the other database systems for T4, T8, T11, T12, and T13. Among those, T11 and T12 require intensive computations over graphs, and T13 requires intensive computations over documents. Due to the lack of a native storage engine for arrays, ArangoDB was outperformed by the polyglot system for T2, T9, T14, and T15 that involve array operations. For these tasks, ArangoDB was slower than AgensGraph as well due to their differential index performance. (Refer to Section 6.5.4.)

6.3.3 ***AgensGraph.*** AgensGraph outperformed the other database systems for T0, T1, T3, T5 to T7, T10, and T16. Overall, it performed well for tasks that involved intensive access to relational tables. This is because AgensGraph relies on the relational storage engine of PostgreSQL to support both the relational and the graph models. Surprisingly, however, for some tasks with graph operations, AgensGraph was faster than ArangoDB which has a native graph engine. (This is discussed in greater detail in Section 6.5.) On the other hand, for tasks with shortest path queries (T11, T12, and T15), AgensGraph was slower than the other database system. AgensGraph yielded poorer performance than the polyglot system for tasks with array operations (T2, T9, T14, and T15).

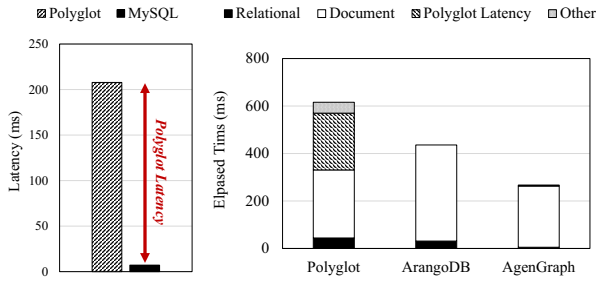
6.4 Polyglot Latency

Although the polyglot system takes advantage of native storage engines for all the data models, its performance was poor for several tasks. In the polyglot system, the client interacts with an individual DBMS through a database connection such as ODBC or JDBC. Whenever the client processes a query, the query is transmitted to the DBMS and the result is returned back to the client. Both the query and result are transmitted through the connection, and the intrinsic overhead is not trivial especially when the client interacts with the DBMS frequently. We call the additional latency caused by this client-DBMS interaction the *polyglot latency*, because it is inherent only in the polyglot system due to its peculiar architecture.

Consider a subquery *Q* of task T1 shown below, which joins a relational table with a document collection. The *Products* table is stored in MySQL, and the *OrderLine* collection produced by UNNESTing the *order_line* field of the *Orders* collection is stored in MongoDB.

```
Subquery Q = SELECT product_name
             FROM Products, OrderLine
             WHERE OrderLine.product_id = Products.product_id
```

To process the query, the client of the polyglot system repeatedly invokes a selection query to MySQL for each *OrderLine.product_id* value. The repeated interaction with MySQL through the database connection leads to significant performance degradation.



(a) Subquery Q **(b) Elapsed times of the task T1**

Figure 8: Polyglot latency in task T1

Figure 8a compares the total elapsed times taken by the polyglot system and MySQL for the subquery *Q*. MySQL processed the subquery *Q* by importing the *OrderLine* collection to a relational table and performing a nested loop join operation. Therefore, the difference in the elapsed times may be considered an upperbound of the polyglot latency of *Q*. The polyglot latency was approximately 95% of the total elapsed time for *Q*. Figure 8b shows the breakdown of the elapsed times for task T1 by the three systems in comparison. The polyglot latency was approximately 42% of the elapsed time taken by the polyglot system. Without the polyglot latency, the polyglot system would outperform ArangoDB.

The polyglot latency was incurred in all the tasks carried out by the polyglot system. Although the polyglot latency can be eliminated for the subquery *Q* by importing the *OrderLine* collection, this sort of optimization cannot be applied to every task. SciDB and Neo4j do not support this kind of join for one. For the tasks amenable to the reduction of polyglot latency, we measured the potential improvement in Table 5.

Table 5: Improvement by avoiding the polyglot latency

Task	T0	T1	T3	T5	T6	T11	T12
Improv. Rate	33.08%	41.92%	7.44%	2.66%	4.94%	5.89%	0.76 %

Improvement Rate = (*Original Time* − *Improved Time*)/*Original Time* × 100

6.5 Effect of Storage Engines

This section analyzes the effect of storage engines on the performance of the database systems with respect to model-specific operations.

6.5.1 ***Relational Model.*** Figure 9 compares the elapsed times of ArangoDB and AgensGraph for the tasks that involve the relational model. Among those tasks, AgensGraph outperformed ArangoDB for nine of them (T0, T1, T3, T6, T7, and T9 to T12). Overall, AgensGraph achieved higher performance in processing selection, aggregation, and join operations. This is primarily because AgensGraph is supported by a relational storage engine, while ArangoDB is not.

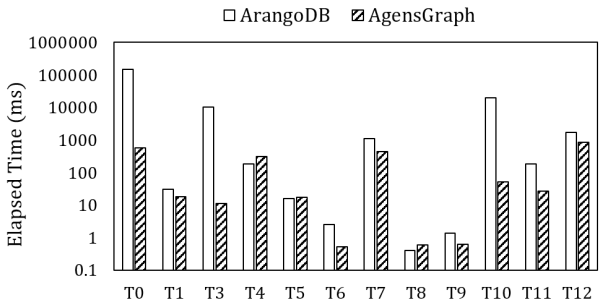


Figure 9: Elapsed times of relational operations

6.5.2 ***Document Model.*** There was no clear winner between ArangoDB and AgensGraph for the document model, despite the fact that ArangoDB is said to support the document model natively while AgensGraph does not. We conjecture that this is because ArangoDB stores JSON documents in RocksDB, which is a key-value store but not a genuine storage engine for the document model. The polyglot system with MongoDB was expected to yield superior performance for processing document operations, but its potential performance gain was canceled out by the polyglot latency.

6.5.3 ***Graph Model.*** ArangoDB and AgensGraph are compared in Figure 10 with respect to the elapsed times taken by graph operations. Tasks T0, T3 to T5, T7, T8, and T10 include pattern matching queries, and T11, T12 and T15 include shortest path queries. ArangoDB outperformed AgensGraph for the shortest path queries up to three orders of magnitude. Recall that ArangoDB is supported by a native storage engine for the graph model.

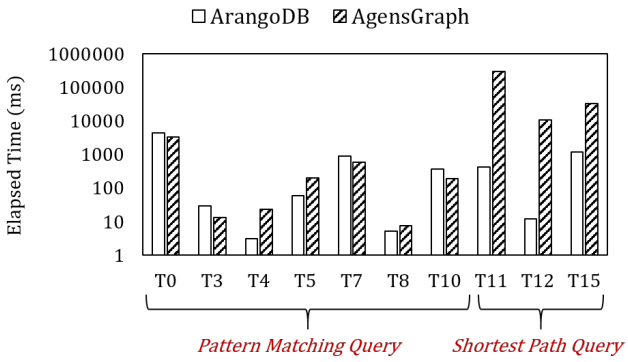


Figure 10: Elapsed times of graph operations

For pattern matching queries that require accessing the adjacent vertices of a given vertex, the performance of ArangoDB and AgensGraph was dependent on the degree (*i.e.* the number of neighbors) of the given vertex. ArangoDB relied on a hash index for vertices, and the degree of the given vertex was its dominant cost factor. In contrast, AgensGraph used a B-tree index for edges (clustered

by vertices), and the cost of B-tree traversal was dominant. Consequently, ArangoDB outperformed (or underperformed) AgensGraph when the degree was low (or high).

6.5.4 ***Array Model.*** Among the three database systems, the polyglot system is the only one that is supported by the array storage engine of SciDB. Figure 11 shows the elapsed times consumed by array operations. Despite the polyglot latency included in the measurement, the polyglot system outperformed the other systems for all the tasks (except for T16). The array operations of those tasks (*e.g.*, matrix multiplications) access neighboring cells together, the high locality of which can benefit from the chunking design of the SciDB storage engine. For T16, however, the polyglot system was slower than AgensGraph because the array cells are accessed randomly. Moreover, cell-accessing queries are invoked iteratively by the client, which elongates the polyglot latency too.

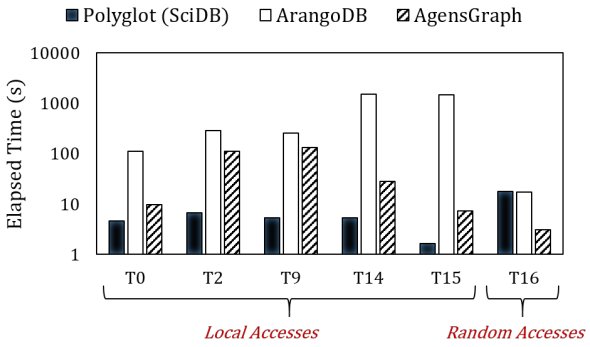


Figure 11: Elapsed times of array operations

AgensGraph outperformed ArangoDB for the array workloads, due to the different ways their indexes were utilized. The index of AgensGraph returns the physical address of a record. Hence AgensGraph requires just one index search and one direct access to the record. In contrast, the index of ArangoDB returns the key of an entry stored in RocksDB, which requires another index search to retrieve the record itself. Hence ArangoDB requires two index searches rather than just one. The performance gap was wider for T14 and T15 requiring range searches, which the compound index of ArangoDB was not so effective for.

6.6 Changing Scale Factors

The performance of the three database systems was measured with a few varying scale factors. For visual clarity, in Figure 12, a linear scale is used to show the elapsed times for the first half of the chosen tasks and a log scale for the rest. As the scale factor increased, the elapsed time of all the systems increased for all the tasks. There was no crossover for any pair of database systems with respect to their relative processing speed. It is notable that the elapsed time of the polyglot system for T1, T4, and T5 increased faster than for the other tasks. This is because, as the scale factor increased, so did the number of records to join from different data models. This increased the join cost and the polyglot latency faster than linearly. For T11 and T12, AgensGraph yielded low performance due to the high cost of the shortest path queries.

6.7 Summary of Evaluation

There was no single winner that outperformed the other database systems for all the tasks. Each database system performed differently for different tasks due to the disparate characteristics intrinsic to the benchmarking tasks. A few key observations made in the evaluation are summarized below.

- The performance of the polyglot system was degraded significantly due to the polyglot latency when data from different models are joined. It was further aggravated by increased scale factors.
- The polyglot latency could be avoided by importing data from one model to another, but this optimization is not applicable to all the multi-model tasks.
- AgensGraph outperformed ArangoDB for relational workloads and array workloads as well (with the effective use of indexes).
- AgensGraph tended to yield better performance when the degree of vertices was high, but it yielded inferior performance for tasks with shortest path queries.
- The storage engine of SciDB played a key role in delivering superior performance for array workloads with high locality in the access patterns.

7 RELATED WORK

Benchmark programs have been an invaluable tool for the database community for decades. This section reviews those benchmark programs with respect to the data models they support and differentiates them from *M2Bench* presented in this paper.

Single Model Benchmark Programs. TPC-H and TPC-DS are among the most popular benchmarks aimed at evaluating a relational database system for analytic workloads [41]. They provide relational database schemas and numerous analytic workloads arising in the E-commerce applications.

XMark and NOBENCH have been used for document-oriented semi-structured databases. While XMark [34] provides pattern matching queries as a benchmark for XML data, NOBENCH [12] provides basic NoSQL queries such as selection, projection, and aggregation for JSON documents.

Linkbench [3] is developed as a benchmark for evaluating a graph database system. It provides simple CRUD operations that reproduce query traces in Facebook's graph database TAO [7]. LDBC-SNB [17] provides an analytic workload that consists of pattern matching queries and aggregation queries on a graph database. GooDBye [26] is another benchmark that consists of graph analytic queries similarly to LDBC-SNB.

SS-DB [15] is a benchmark for array database systems. Its workload is designed for astronomical data, and consists of subarray queries and mulitidimensional aggregation queries. GenBase [39] is another benchmark for the array data model, and provides a workload for microarray genomic data. The workload of GenBase is different from that of SS-DB in that the former consists mainly of linear algebra queries such as matrix multiplications.

Multi-Model Benchmark Programs. Relatively recently, BigBench [21, 22] and UniBench [50, 51] have been proposed to support multiple data models. BigBench provides workloads for big data

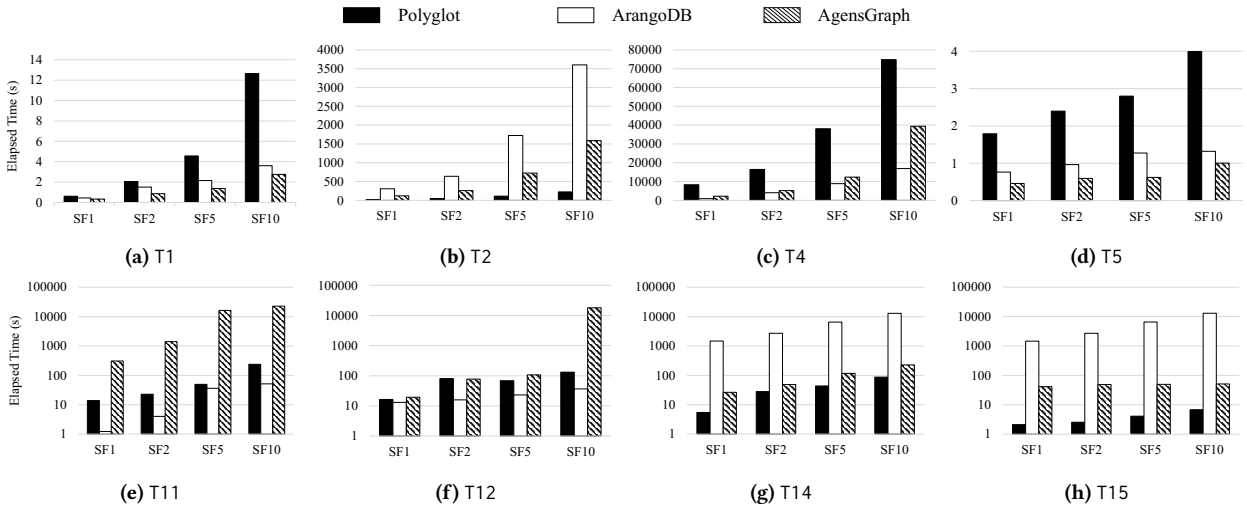


Figure 12: Elapsed times with varying scale factors

analytics based on both the relational and the document models. BigBench V2, an improved version of BigBench, adds a key-value data model to BigBench. UniBench provides OLTP and OLAP workloads for E-Commerce databases based on the relational, document and graph data models.

These multi-model benchmarks, namely BigBench and UniBench, are different from our *M2Bench* in a few ways. First, they do not support the array data model, which has become one of the essential tools for machine learning applications. *M2Bench* supports the array model as one of its core data models. Second, they focus on evaluating different aspects of a database system. BigBench and BigBench V2 focus mainly on big data processing as their names suggest, and UniBench focuses on the impact of join orders, complex aggregations, and ACID properties. On the other hand, the main focus of *M2Bench* is on identifying the impact of individual data models on the performance. Third, while they are designed around the E-Commerce use cases only, *M2Bench* provides more diverse workloads arising from three domain categories, namely, E-Commerce, Healthcare and Disaster & Safety.

The multi-model database systems, AgensGraph and ArangoDB, have already been evaluated comparatively by UniBench [50]. It should be noted that the UniBench's verdict on these two systems is not entirely equivalent to that of *M2Bench* presented in this paper. UniBench reports that ArangoDB performs queries on the JSON documents more efficiently than AgensGraph, while *M2Bench* reports that AgensGraph and ArangoDB deliver comparable performance for the document model. The discrepancy in these two bodies of study is due to the fact that UniBench included a *cross join* operation for AgensGraph to flatten the nested arrays of documents while *M2Bench* did not. Without the additional *cross join* operation, they both would arrive at the same conclusion. UniBench and *M2Bench* report equivalent assessments for the relational and graph data models. Both of them observed that AgensGraph outperforms ArangoDB for the relational model and ArangoDB outperforms AgensGraph for tasks with a shortest path operation.

Table 6 summarizes the benchmark programs discussed in this section with the data models they support.

Table 6: Benchmark programs and data models

Benchmark Programs	*Relational*	*Document*	*Graph*	*Array*
TPC-H, TPC-DS [41]	✓			
XMark [34]		✓		
NOBENCH [12]		✓		
LinkBench [3]			✓	
GooDBye [26]			✓	
LDBC-SNB [17]			✓	
SS-DB [15]				✓
GenBase [39]				✓
BigBench [21, 22]	✓	✓		
UniBench [50, 51]	✓	✓	✓	
M2Bench	✓	✓	✓	✓

8 CONCLUSION

We propose a new benchmark program called *M2Bench* to evaluate database management systems aimed for multi-model analytic workloads. *M2Bench* covers four important data models: relational, document-oriented, property graph, and array. *M2Bench* help users identify the impact of specific data models on the performance of a DBMS. It provides a diverse range of workloads inspired by three representative real-world scenarios. By executing the *M2Bench* workloads on a few chosen multi-model database systems, we have demonstrated that *M2Bench* can effectively pinpoint the strengths and weaknesses of each system in regard to individual data models. We believe that these features will make *M2Bench* an essential tool for evaluating multi-model database systems.

ACKNOWLEDGMENTS

This work was supported in part by the National Research Foundation (NRF) of Korea (Grant No. NRF-2020R1A2C1010358). The authors assume all responsibility for the content of the paper.

REFERENCES

[1] Renzo Angles and Claudio Gutierrez. 2008. Survey of Graph Database Models. *ACM Comput. Surv.* 40, 1, Article 1 (Feb 2008), 39 pages.

[2] ArangoDB, Inc. 2022. *ArangoDB.* Retrieved December 12, 2022 from https://www.arangodb.com/

[3] Timothy G. Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. 2013. LinkBench: A Database Benchmark Based on the Facebook Social Graph. In *Proceedings of the 2013 ACM SIGMOD Conference*. New York, NY, USA, 1185–1196.

[4] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann. 1998. The Multidimensional Database System RasDaMan. *SIGMOD Rec.* 27, 2 (June 1998), 575–577.

[5] Bitnine Global Inc. 2021. *AgensGraph.* Retrieved December 12, 2022 from https://bitnine.net/agensgraph/

[6] Matthias Boehm, Michael W. Dusenberry, Deron Eriksson, Alexandre V. Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick R. Reiss, Prithviraj Sen, Arvind C. Surve, and Shirish Tatikonda. 2016. SystemML: Declarative Machine Learning on Spark. *Proc. VLDB Endow.* 9, 13 (Sep 2016), 1425–1436.

[7] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song, and Venkat Venkataramani. 2013. TAO: Facebook's Distributed Data Store for the Social Graph. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*. San Jose, CA, USA, 49–60.

[8] D. Brown and N. Hayes. 2008. *Influencer Marketing: Who Really Influences Your Customers?* Elsevier/Butterworth-Heinemann.

[9] Peter Buneman. 1997. Semistructured Data. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. Tucson, Arizona, USA, 117–121.

[10] C. Demetrescu, A. Goldberg, D. Johnson. 2006. *9th DIMACS Implementation Challenge - Shortest Paths.* Retrieved October 10, 2022 from http://www.diag.uniroma1.it/~challenge9/index.shtml

[11] Center for International Earth Science Information Network - CIESIN - Columbia University. 2018. *Gridded Population of the World, Version 4 (GPWv4): Administrative Unit Center Points with Population Estimates, Revision 11.* Palisades, NY, USA. Retrieved October 10, 2022 from https://doi.org/10.7927/H4BC3WMT

[12] Craig Chasseur, Yinan Li, and Jignesh M. Patel. 2013. Enabling JSON Document Stores in Relational Systems. In *Proceedings of the 16th International Workshop on the Web and Databases*. New York, NY, USA, 1–6.

[13] E. F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13, 6 (June 1970), 377–387.

[14] Douglas Crockford and Chip Morningstar. 2017. Standard ECMA-404 The JSON Data Interchange Syntax. https://doi.org/10.13140/RG.2.2.28181.14560

[15] Philippe Cudre-Mauroux, Hideaki Kimura, Kian-Tat Lim, Jennie Rogers, Samuel Madden, Michael Stonebraker, Stan Zdonik, and Paul Brown. 2010. SS-DB: A standard science DBMS benchmark. In *4th Extremely Large Databases Conference*. Menlo Park, California, USA, 11.

[16] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. RocksDB: Evolution of Development Priorities in a Key-Value Store Serving Large-Scale Applications. *ACM Trans. Storage* 17, 4, Article 26 (Oct 2021), 32 pages.

[17] Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat, Minh-Duc Pham, and Peter Boncz. 2015. The LDBC Social Network Benchmark: Interactive Workload. In *Proceedings of the 2015 ACM SIGMOD Conference*. Melbourne, Victoria, Australia, 619–630.

[18] Reza Ferdousi, Reza Safdari, and Yadollah Omidi. 2017. Computational prediction of drug-drug interactions based on drugs functional similarities. *Journal of Biomedical Informatics* 70 (2017), 54–64.

[19] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. An Overview of the HDF5 Technology Suite and Its Applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases* (Uppsala, Sweden) *(AD '11)*. Association for Computing Machinery, New York, NY, USA, 36–47. https://doi.org/10.1145/1966895.1966900

[20] 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 *Proceedings of the 2018 ACM SIGMOD Conference*. Houston, TX, USA, 1433–1445.

[21] Ahmad Ghazal, Todor Ivanov, Pekka Kostamaa, Alain Crolotte, Ryan Voong, Mohammed Al-Kateb, Waleed Ghazal, and Roberto V. Zicari. 2017. BigBench V2: The New and Improved BigBench. In *33rd IEEE ICDE Conference*. San Diego, CA, USA, 1225–1236.

[22] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. In *Proceedings of the 2013 ACM SIGMOD Conference*. New York, New York, USA, 1197–1208.

[23] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. MIMIC-III, a freely accessible critical care database. *Scientific data* 3, 1 (2016), 1–9.

[24] Kyoseung Koo, Juhun Kim, and Bongki Moon. 2021. MISE: An Array-Based Integrated System for Atmospheric Scanning LiDAR. In *Proceedings of the 33rd SSDBM Conference*. Tampa, FL, USA, 265–269.

[25] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37.

[26] Piotr Matyjaszczyk, Przemyslaw Rosowski, and Robert Wrembel. 2020. GooDBye: a Good Graph Database Benchmark - an Industry Experience. In *EDBT/ICDT Workshops*. Copenhagen, Denmark, 6.

[27] MongoDB, Inc. 2022. *MongoDB.* Retrieved December 12, 2022 from https://www.mongodb.com/

[28] Neo4j, Inc. 2022. *Neo4j.* Retrieved December 12, 2022 from https://neo4j.com/

[29] OpenStreetMap contributors. 2022. *Planet dump October 10, 2022. Retrieved from https://planet.osm.org* .

[30] Oracle. 2022. *MySQL.* Retrieved December 12, 2022 from https://www.mysql.com/

[31] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy Mattson. 2016. The TileDB Array Data Storage Manager. *Proc. VLDB Endow.* 10, 4 (Nov 2016), 349–360.

[32] Himchan Park and Min-Soo Kim. 2018. EvoGraph: An Effective and Efficient Graph Upscaling Method for Preserving Graph Properties. In *Proceedings of the 24th ACM SIGKDD Conference*. London, United Kingdom, 2051–2059.

[33] Abdul Quamar, Jannik Straube, and Yuanyuan Tian. 2020. Enabling Rich Queries Over Heterogeneous Data From Diverse Sources In HealthCare.. In *10th Conference on Innovative Data Systems Research, CIDR, Online Proceedings*. Amsterdam, The Netherlands, 6.

[34] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. 2002. XMark: A Benchmark for XML Data Management. In *Proceedings of the 28th VLDB Conference*. Hong Kong, China, 974–985.

[35] SNOMED International. 2022. *SNOMED Terminolody.* Retrieved October 10, 2022 from https://www.snomed.org/

[36] SNU-DBS. 2022. *M2Bench Github Repository.* Retrieved December 12, 2022 from https://github.com/snu-dbs/m2bench

[37] SNU-DBS. 2022. *M2Bench Task Explanation.* Retrieved December 12, 2022 from https://github.com/snu-dbs/m2bench/blob/publish/Tasks/alltasks.md

[38] Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. 2011. The Architecture of SciDB. In *Proceedings of the 23rd SSDBM Conference*, Vol. 6809. Portland, OR, USA, 1–16.

[39] Rebecca Taft, Manasi Vartak, Nadathur Rajagopalan Satish, Narayanan Sundaram, Samuel Madden, and Michael Stonebraker. 2014. GenBase: A Complex Analytics Genomics Benchmark. In *Proceedings of the 2014 ACM SIGMOD Conference*. Snowbird, Utah, USA, 177–188.

[40] The PostgreSQL Global Development Group. 2022. *PostgreSQL.* Retrieved December 12, 2022 from https://www.postgresql.org/

[41] TPC. 2022. *TPC Benchmarks.* Retrieved December 12, 2022 from https://www.tpc.org/

[42] U.S. Department of Homeland Security. 2021. *National Shelter System Facilities.* Retrieved October 10, 2022 from https://hifld-geoplatform.opendata.arcgis.com/datasets/geoplatform::national-shelter-system-facilities/about

[43] U.S. Geological Survey. 2022. *Earthquakes.* Retrieved October 10, 2022 from https://www.usgs.gov/natural-hazards/earthquake-hazards/earthquakes

[44] Santiago Vilar, Eugenio Uriarte, Lourdes Santana, Tal Lorberbaum, George Hripcsak, Carol Friedman, and Nicholas P Tatonetti. 2014. Similarity-based modeling in large-scale prediction of drug-drug interactions. *Nature protocols* 9, 9 (2014), 2147–2163.

[45] W3C. 1998. *Extensible Markup Language (XML) 1.0.* Retrieved December 12, 2022 from https://www.w3.org/TR/1998/REC-xml-19980210.html

[46] Wikipedia contributors. 2022. *Polyglot persistence — Wikipedia, The Free Encyclopedia.* Retrieved December 12, 2022 from https://en.wikipedia.org/wiki/Polyglot_persistence

[47] David S Wishart, Craig Knox, An Chi Guo, Savita Shrivastava, Murtaza Hassanali, Paul Stothard, Zhan Chang, and Jennifer Woolsey. 2006. DrugBank: a comprehensive resource for in silico drug discovery and exploration. *Nucleic acids research* 34, suppl_1 (2006), D668–D672.

[48] Jinhong Xian, Dongsong Sun, Wenjing Xu, Yuli Han, Jun Zheng, Jiancao Peng, and Shaochen Yang. 2020. Urban air pollution monitoring using scanning Lidar. *Environmental Pollution* 258 (2020), 113696.

[49] Manzhu Yu, Chaowei Yang, and Yun Li. 2018. Big Data in Natural Disaster Management: A Review. *Geosciences* 8, 5 (2018), 26.

[50] Chao Zhang and Jiaheng Lu. 2021. Holistic evaluation in multi-model databases benchmarking. *Distributed and Parallel Databases* 39, 1 (Mar 2021), 1–33.

[51] Chao Zhang, Jiaheng Lu, Pengfei Xu, and Yuxing Chen. 2018. UniBench: A Benchmark for Multi-model Database Management Systems. In *Proceedings of the Technology Conference on Performance Evaluation and Benchmarking (TPCTC 2018)*. Rio de Janeiro, Brazil, 7–23.