Parameter Curation and *Data generation* for Benchmarking Multi-model Queries

Chao Zhang
Supervised by Jiaheng Lu
University of Helsinki, Finland

ABSTRACT

Unlike traditional database management systems which are organized around a single data model, a multi-model database is designed to support multiple data models against a single, integrated backend. For instance, document, graph, relational, and key-value models are examples of data models that may be supported by a multi-model database. As more and more platforms are proposed to deal with multi-model data, it becomes important to have benchmarks that can be used to evaluate performance and usability of the next generation of multi-model database systems. In this paper, we discuss the motivations and challenges for benchmarking multi-model databases, and then present our current research on the data generation and parameter curation for benchmarking multi-model queries. Our benchmark can be found at *http://udbms.cs.helsinki.fi/bench/*.

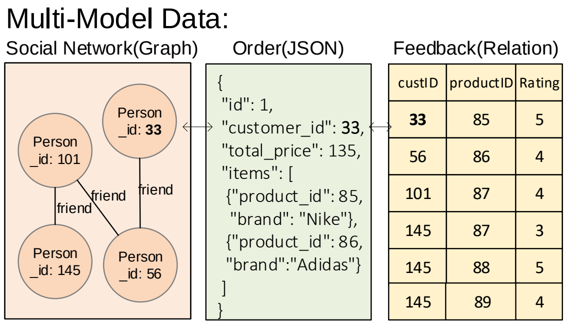
1. INTRODUCTION

Recently, there is a new trend [12, 11, 13, 3] for data management, namely, the multi-model approach, which mainly aims to utilize a single platform to manage data in different models, e.g., key-value, document, table, and graph. Compared to the polyglot persistence technology in NoSQL world which entails managing separate data stores to satisfy various use cases, the multi-model approach has been considered as the next generation of data management technology combining flexibility, scalability, and consistency.

The multi-model query is a unique operation in multi-model databases which allows users to retrieve multi-model data by using a single query language. Figure 1 depicts an example of a typical multi-model query in the social commerce [17]: *For a given person p(id=56) and product brand b("Nike"), find p's friends who have bought products in brand b, and return their feedback which contains product's reviews with the 5-star rating.* This query involves three data models: customer with friends (*Graph*), order embedded with an item list (*JSON*), and customer's feedback (*Relation*).

Database benchmark becomes an essential tool for the evaluation and comparison of DBMSs since the advent of Wisconsin benchmark [5] in the early 1980s. Since then, many database benchmarks have been proposed by academia

Proceedings of the VLDB 2018 PhD Workshop, August 27, 2018. Rio de Janeiro, Brazil. Copyright (C) 2018 for this paper by its authors. Copying permitted for private and academic purposes



```
{
 "id": 1,
 "customer_id": 33,
 "total_price": 135,
 "items": [
  {"product_id": 85,
   "brand": "Nike"},
  {"product_id": 86,
   "brand":"Adidas"}
 ]
}
```

custID	productID	Rating
33	85	5
56	86	4
101	87	4
145	87	3
145	88	5
145	89	4

ArangoDB AQL:

```
FOR friend IN 1..1 OUTBOUND PersonId/56 KnowsGraph
FOR order IN Order
FOR feedback IN Feedback
  FILTER order.customer_id==friend._id AND
   BrandName/"Nike" IN order.items[*].brand AND
   friend._id==feedback.custID AND
   feedback.Rating==5
RETURN {person:friend, feedback:feedback}
```

Figure 1: An Example of Multi-Model Query. *Top*: an excerpt of joined multi-model data involving Graph, JSON, and Relation. *Bottom*: the same query as above, written in the ArangoDB AQL query language.

and industry for various evaluation goals, such as TPC-X series for RDBMSs and data warehouses, OO7 [2] benchmark for object-oriented DBMSs, and XMark [16] for XML DBMSs. More recently, The NoSQL and big data movements in the late 2000s brought the arrival of the next generation of benchmarks, such as YCSB benchmark [4] for cloud serving systems, LDBC [6] and Rbench [15] benchmarks for Graph and RDF DBMSs, BigBench [8] benchmark for big data systems. Unfortunately, these benchmarks are not well suited for the evaluation of multi-model databases due to the lack of consideration of multiple data models, e.g., multi-model storage, multi-model query processing, multi-model query evaluation. Motivated by this, my Ph.D. dissertation will focus on the holistic evaluation of multi-model databases. In general, there are three main challenges on evaluating multi-model databases:

First, existing data generators that only involve single model cannot be directly adopted to evaluate the multi-model databases, and how to design a meaningful data models to mimic most cases of multi-model application remains an open question. In this regard, we develop a new data gen-

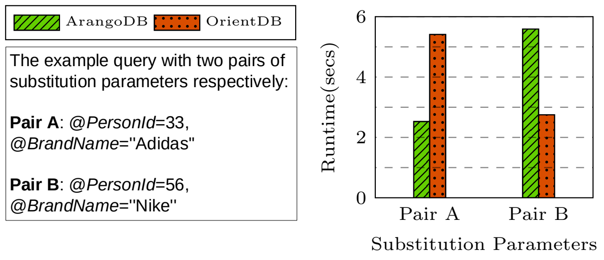


Figure 2: The Motivation of Parameter Curation

erator to generate the correlated data in diverse data models, including Graph, JSON, XML, key-value, and tabular. Furthermore, to simulate the data distributions in real life, we propose a three-phase framework to generate the data in the scenario of social commerce. Our data generator also has good scalability because it is implemented on the top of Hadoop and Spark, which enables us to generate the data in parallel.

The second benchmarking challenge is the problem of *Parameter Curation* [9], with the goal of selecting the *substitution parameters* for the multi-model query template to yield stable runtime behaviors. The rationale is that, the different parameter values for same query template would result in high runtime variance. For instance, in Figure 1, *PersonId/56* and *BrandName/"Nike"* with the orange color in the AQL query are two *substitution parameters* that can be replaced by other values for the example query template. Figure 2 illustrates our experiment results with two different pairs of *substitution parameters* on two representative multi-model databases: ArangoDB [1] and OrientDB [14]. As shown in Figure 2, these parameters lead to the opposite evaluation results to compare the performance between ArangoDB and OrientDB. Interestingly, we observed the query runtime mainly depends on the domination of data models. For example, pair A involves relative larger intermediate results of JSON while pair B takes in the larger size of Graph. Therefore, the problem of parameter curation for benchmarking the multi-model queries requires answering three interesting questions: (i) how to select parameters from the data model perspective, (ii) how to cover different workloads concerning the data model, and (iii) how to guarantee the stable distribution of *substitution parameters*. In light of this, we formalize this problem as the top-k parameter groups curation, and then propose a new algorithm, *MJFast*, to select the ideal parameter groups.

The third challenge corresponds to the metrics of the benchmark. As expected, both metrics for evaluating the multi-model dataset (e.g., how closely the data mimic the real heterogeneous datasets) and multi-model query (e.g., to what extent the queries capture the diverse multi-model patterns) are needed. However, the disparity between data models in the data structure and workload complexity is a major hurdle when trying to define these new metrics. For the dataset evaluation metrics, we intend to use the dataset coherence and relationship specialty [15], as well as the multi-model complexity. Regarding the metrics of multi-model query, we define a unified metric, characterizing the query processing concerning the data models. For instance, the metric can be used to either measure the cost of the nested-loop join for the relational model or assess the cost of the shortest path matching for the graph model.

The rest of this paper is divided as follows. Section 2 presents the overview of our approach. Section 3 introduces the methods and techniques for the data generation. Section 4 gives our method for the parameter curation. Section 5 shows the preliminary experimental results. Finally, the last chapter summarizes this paper and outlines our future work.

2. OVERVIEW OF OUR APPROACH

Figure 3 gives an overview of our benchmarking approach, which consists of three key components to evaluate the multi-model query. The metadata in the repository is first passed into the **Data Generation** (Section 3) component that generates the data in a unified multi-model form based on our developed data generator. Next, the **Workload Generation** component generates the multi-model queries against the data models. These multi-model queries consist of a set of complex read-only queries that involve at least two data models, aiming to cover different business cases and technical perspectives. More specifically, as for business cases, these queries fall into four main levers [10] : *individual*, *conversation*, *community*, and *commerce*. In these four levers, common-used business cases in different granularities are rendered. Regarding technical perspectives, these queries are designed based on the *choke-point* concept which combines usual technical challenges to process the data in multiple data models, ranging from the conjunctive queries (OLTP) to analysis (OLAP) workloads. The final part is the **Parameter Curation** (Section 4) component. It first characterizes the multi-model query by identifying the parameters and corresponding involved data models. Then model vectors corresponding to each parameter value are generated. Finally, the top-k parameters for the multi-model query are selected based on the proposed MJFast algorithm.

3. DATA GENERATION

The data generation is the cornerstone of our benchmark and comprised of two main parts: social network generation and e-commerce data generation. The former part is to generate the social graph, including the person entities and knows relations, as well as their activities such as posts, comments, and likes. This generation is based on the LDBC SNB[6] data generator, which is a representative tool of generating data in the social network with rich semantics and scalability. The latter one is to generate the e-commerce data. Specifically, we propose a three-phase framework to generate the transactions by taking into account person's interests, friendship, and social engagement. The three-phase framework consists of *purchase, propagation-purchase, repurchase* in the context of social commerce.

Purchase. In this phase, we consider two factors when generating the transaction data. First, persons usually buy products based on their interests. Second, persons owning more interests are more likely to buy products than others. This phase is implemented on the top of Spark SQL using scala, which utilizes a plentiful APIs and UDFs to output the various model simultaneously without any additional operations. Consequently, our data include five models: social network (Graph), vendor and feedback (Relation), order (JSON), invoice (XML), product (Key-value).

Propagation-Purchase. In this phase, we incorporate two ingredients from previous data generation: (i) person's basic demographic data, e.g., gender, age, location. (ii)

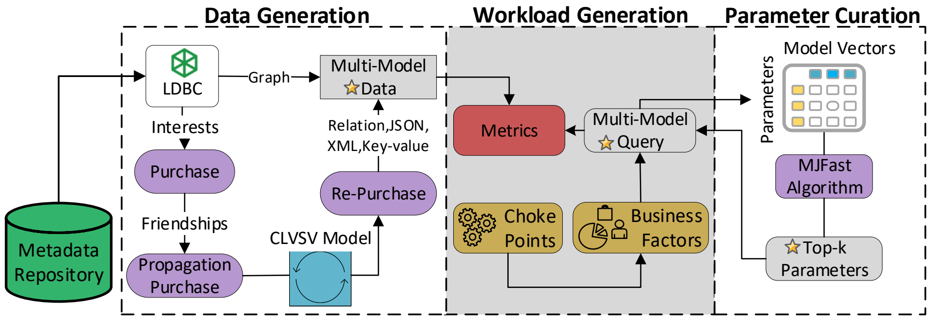


Figure 3: Overview of our approach

friends' transaction. The scoring function is defined as follow:

$$S_{ui} = \sum_k k \times Pr(R_{ui} = k|A = a_u) + E(R_{vi} : \forall v \in N(u)) \quad (1)$$

where $\sum_k k \times Pr(R_{ui} = k|A = a_u)$ is the expectation of the probability distribution of the target user u's rating on the target item i, and $A = \{a_1, a_2, ..., a_m\}$ is user attribute set, we compute this part based on naive bayes method. The latter part $E(R_{vi} : \forall v \in N(u))$ is the expectation of u's friends' rating distribution on the target item, in which $N(u)$ is the friends set of user u, and the item i is from the purchase transaction of friends.

Re-Purchase. To make fine-grained predictions by incorporating the customer's social activities, we propose a new probabilistic model, CLVSC (Customer Lifetime Value in Social Commerce), to generate the transactions based on the history of customer's purchases and social activities:

$$\begin{aligned} CLVSC_{ib} &= E(X^* \,|\, n^*, x', n, m, \alpha, \beta, \gamma, \delta) \\ &\times (E(M \,|\, p, q, \upsilon, m_x, x) + E(S \,|\, \bar{s}, \theta, \tau)) \end{aligned} \quad (2)$$

where i and b are the customer and brand index, respectively, $E(X^* \,|\cdot)$ is the expected number of behaviors, and $E(M \,|\cdot)$ is the expected monetary value, parameters in these two parts are for the beta-geometric/beta-binomial model [7], and $E(S \,|\cdot)$ is the expected number of customer's social activities, in which the parameters are for the Poisson-gamma model.

4. PARAMETER CURATION

4.1 Preliminaries

In this section, we describe the preliminaries for the parameter curation problem. We assume that the selection of parameters for a multi-model query should guarantee the following properties: (i) the query result should correspond to involved data models and their combinations, (ii) the size of involved data models should be bounded in each class, (iii) the selected parameters should cover different classes in the whole parameter space.

To satisfy the property (i), we propose a vector-based approach to represent parameter values from the multi-model perspective. Specifically, we compute sizes of all intermediate results correspond to a parameter value based on the permutation of data models. For instance, given the query in the Section 1, and a parameter pair *(p, b)*, we compute a non-zero vector *(G,J,GJ,GJR)*, where G stands for Graph, J for JSON,R for Relation, GJ refers to the combination of these two models, i.e., persons who are p's friends and have bought products in brand b. This method allows us to represent the model-oriented results independently of the databases. The definition of the model vector is as follow:

Definition 1. ***Model Vector****: In a multi-model query, each model vector is defined as ω $\{c_1, ., c_k, ., c_n\}$, where c_k is k-th intermediate result size against involved data models or their combinations, c_n is the final result size. The length of ω is between [3, 2^m-1], where m is the number of the data model.*

Regarding property (ii), we assume that a representative class in the whole parameter space consists of two traits: the considerable number of model vectors, and the bounded distance between these vectors. Hence, we define the qualified class as the candidate parameter group:

Definition 2. ***Candidate parameter group****: In the parameter space, the candidate parameter group is the space with radius ϵ covering at least ν model vectors.*

To fulfill the property (iii), we find the k farthest candidate parameter groups. Therefore, the parameter curation problem boils down to finding the top-k candidate parameter groups, with the maximum number of model vectors, and maximum distance between groups.

4.2 Problem Definition and Algorithm

We now formalize the problem as follow:

Top-*k* Parameter Groups Curation: Given the multi-model query MQ with parameter space P that is a set of N points in $\mathbb{R}^d$, each point in P is a d-dimensional multi-model vector, the distance between two groups is the Euclidean distance between centroid of groups. The objective is to select k disjoint candidate parameter groups $S_k \subset P$ such that the score S

$$S(S_k) = \alpha \sum_1^k Density(S_k)/N_1 + \beta \sum_1^k Distance(S_k)/N_2 \quad (3)$$

is maximized, where the α and β reflects the importance of density and distance, which has $\alpha + \beta = 1$. N_1 and N_2 are the normalization constants that normalize the sum of density and the sum of distance between 0 to 1, respectively.

The problem of parameter curation is non-trivial because it includes two NP-COMPLETE problems: *top-k highest dense regions problem* and *top-k weighted maximum vertex cover problem*. Therefore, we propose a greedy algorithm, called *MJFast*, to tackle this problem. The main idea of *MJFast* is first to gather the similar candidate parameter groups into the cluster, then chooses the top-k densest candidate parameter groups from each cluster. Finally, the top-k farthest groups from all groups are returned. In specific, we propose a new data structure, *snowball*, to store the strongly closed candidate parameter groups. Snowball starts with an arbitrary candidate parameter group in which the centroid is a model vector. Then it recursively rolls if other qualified centroid vectors exist among the group. Since each snowball only maintains top-k densest groups at each iteration, the search space will be reduced dramatically. In *MJFast*, we build a k-d tree to speed up the searches for nearest neighbor, thus the average search time is O(log$n$). In the worst case that all points of P are within the same group, the time complexity is O($n$log$n$).

5. EXPERIMENT RESULTS

5.1 Data Generation

In the case of efficiency, experiment result suggests the data generator can generate 1G multi-model dataset in 5 minutes, on a single 8-core machine running MapReduce and Spark in "pseudo-distributed" mode. In terms of scalability, we successfully generate 10G dataset within 20 minutes on a cluster with three nodes.

5.2 Parameter Curation

We use two metrics to compare the *MJFast* with the random method. First, to measure the stability of the method, we compare the KL-divergence D_{KL} between two groups of parameters for a fixed k. The distribution is the discrete distribution of model-dominating. For example, when k is 5, the two model-dominating distributions are (G:4, J:1, R:0) and (G:1, J:1, R:3) respectively, so the D_{KL} is 1.11. Second, we proceed with experiments on ArangoDB to compare the total runtime variance (TRV) between two groups of parameters. As shown in Table 1b, the parameters curated by *MJFast* can not only yield the small value of the D_{KL}, but also result in low runtime variance as the k increase.

5.3 Preliminary Benchmarking Results

Table 1a illustrates the preliminary results for benchmarking the multi-model queries on ArangoDB and OrientDB. We conduct the experiments on a quad-core Xeon E5540 server with 32GB of RAM and 500GB of disk. All of benchmark queries involve at least two models, in particular, Q1, Q2, Q5 are Graph-dominating workloads, and Q3, Q4 are JSON-dominating workloads. The results show that, in multi-model context, OrientDB outperform ArangoDB regarding the Graph-dominating workload, and ArangoDB is better at JSON-dominating workload. This also suggests that the multi-model capacities of these two databases depend on their main models. i.e., ArangoDB is originally a document-oriented database, and OrientDB is a natively graph database.

Query	Q1(G&J)	Q2(G&R)	Q3(J&G)	Q4(J&R)	Q5(G&J&R)
ArangoDB	0.452	7.134	**16.566**	**3.341**	88.534
OrientDB	**0.187**	**2.360**	24.653	6.324	**12.653**

(a) Mean runtime of multi-model queries (s)

	Random		MJFast	
	D_{KL}	TRV	D_{KL}	TRV
k=5	0.89	23.0 s	**0**	**0.5 s**
k=10	0.19	56.7 s	**0.02**	**1.2 s**
k=20	0.11	78.6 s	**0.03**	**2.5 s**

(b) Results for Parameter Curation

Table 1: Preliminary Experiment Results

6. CONCLUSION AND FURTURE WORK

Benchmarking the multi-model databases is a challenging task since current public data and workloads can not well match the various cases of real applications. To date, we have developed a scalable data generator to provide data in multiple data models, involving Graph, JSON, XML, key-value, and tabular. *MJFast* algorithm, which is proposed to address the problem of parameter curation, ensures that our performance analysis is holistic and valid.

The general plan to complete my Ph.D. dissertation is to focus on the three components shown in Figure 3. First, the data schema and corresponding model in the real application could be changed, we will introduce this process in data generation. Second, we will optimize the *MJFast* algorithm by incorporating the sampling-based method to avoid the computation of whole parameter space. Finally, we will finalize the multi-model query template and the unified metric, and then conduct a set of experimental study on multi-model databases. Another extension is to investigate the ACID guarantees of multi-model transactions.

Acknowledgement This work is supported by Academy of Finland (310321), China Scholarship and CIMO Fellowship.

7. REFERENCES

[1] ArangoDB. Highly available multi-model NoSQL database. https://www.arangodb.com/, 2017.

[2] M. J. Carey, D. J. DeWitt, and J. F. Naughton. The oo7 benchmark. In *ACM SIGMOD*, pages 12–21, 1993.

[3] J. Chen, Y. Chen, X. Du, C. Li, J. Lu, S. Zhao, and X. Zhou. Big data challenge: a data management perspective. *Frontiers Comput. Sci.*, 7(2):157–164, 2013.

[4] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *ACM SoCC*, pages 143–154, 2010.

[5] D. J. DeWitt. The wisconsin benchmark: Past, present, and future. In *The Benchmark Handbook*, pages 119–165. 1991.

[6] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat-Pérez, M. Pham, and P. A. Boncz. The LDBC Social Network Benchmark: Interactive Workload. In *SIGMOD 2015*.

[7] P. S. Fader. *Customer-base analysis with discrete-time transaction data*. PhD thesis, University of Auckland, 2004.

[8] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H. Jacobsen. BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. In *ACM SIGMOD*, 2013.

[9] A. Gubichev and P. Boncz. Parameter curation for benchmark queries. In *TPCTC*, pages 113–129, 2014.

[10] Z. Huang and M. Benyoucef. From e-commerce to social commerce: A close look at design features. *ECRA*, 2013.

[11] J. Lu. Towards Benchmarking Multi-Model Databases. In *CIDR*, 2017.

[12] J. Lu and I. Holubová. Multi-model data management: What's new and what's next? In *EDBT*, 2017.

[13] J. Lu, Z. H. Liu, P. Xu, and C. Zhang. UDBMS: road to unification for multi-model data management. *CoRR*, abs/1612.08050, 2016.

[14] OrientDB. Distributed Multi-model and Graph Database. http://orientdb.com/orientdb/, 2017.

[15] S. Qiao and Z. M. Özsoyoglu. Rbench: Application-specific RDF benchmarking. In *ACM SIGMOD*, 2015.

[16] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *VLDB*, pages 974–985, 2002.

[17] K. Z. Zhang. Consumer behavior in social commerce: A literature review. *Decision Support Systems*, 2016.