UDBMS: UNIFIED DBMS FOR BOTH RELATION AND NOSQL DATA

Jiaheng Lu[1],
Heli Helskyaho[2],
Pengfei Xu[3]

[1] jiaheng.lu@helsinki.fi,
[2] heli.helskyaho@miracleoy.fi,
[3] pengfei.xu@helsinki.fi

INTRODUCTION

This project investigate a new approach for the unified storage and querying for both relational and NoSQL data. Our approach will reduce integration issues, simplify operations, and eliminate migration issues between relational and NoSQL data. We will develop a prototype to store and query both relational and NoSQL data, based on the techniques proposed, to support emerging applications such as social media marketing, social search and knowledge bases. A breakthrough in this subject will advance several fields, including big data, theory of computation, parallel computation, and social network analysis.

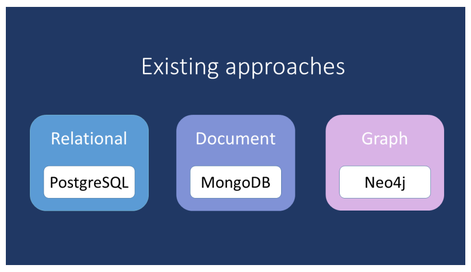

Figure 1: Existing approaches handle data separately by type

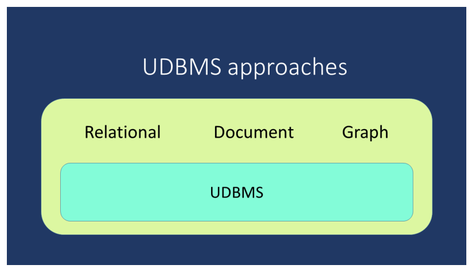

Figure 2: Our approach handles multiple data type in unified entity

Relational databases support the relational data model, generally with SQL as query language. In a similar way, the first-generation NoSQL databases each support a single data model, such as a document, graph, or column-oriented model, along with a specialized query language. In these systems, features tend to be interwoven among all the levels, with the data model hard-wired in the middle. The restriction to a single data model limits the range of use cases the database can handle well: different data models are better suited to different use cases, and applications often require more than one data model for different data types. As a result, developers face a choice between shoehorning all of their application's data into an inappropriate model and integrating multiple databases into their application's backend. However, rather than having to integrate multiple databases, it's far better if a developer can build multiple data models easily and efficiently on a single storage substrate, which becomes our research target.

This project is divided into three main stages and five tasks, as in Figure 3.

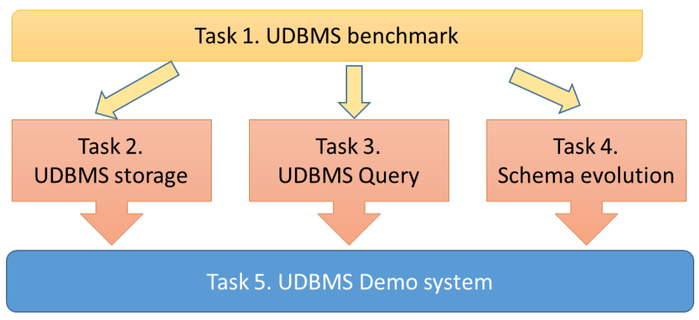

Figure 3: Stages and tasks of UDBMS project

TASK 1 AND 2: BENCHMARKING AND DATA STORAGE

We will develop a benchmark system for unified databases. Our proposal will cover data model not only account for the velocity and volume, but also account for the variety, for which the 3Vs features are indivisible in big data. In particular, we will develop a scenario of e-commerce application to meet the needs of different user involving structured data, semi-structured data and unstructured data. A high-level overview of the data model is presented in Figure 4.

The data model for UDBMS Benchmark includes Relational data, XML data, Graph data, and JSON data. Since the UDBMS Benchmark provides a variety of data types, it can also generate varied SQL query tailored to user's special needs. We will create more than 30 business questions for the UDBMS workload, the format will like "Given a product, find the top 30 customers who ordered the largest amount of this product".

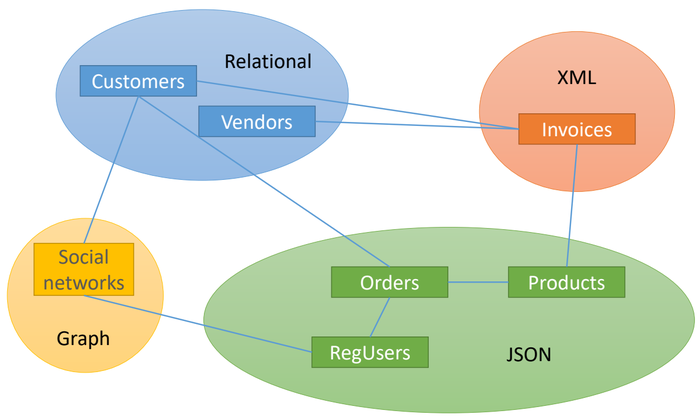

Figure 4: Example of UDBMS Storage with four types of data

TASK 3: DATA QUERY

The new system structure will have a middleware layer (as in Figure 5) that provides CRUD interfaces, while every data source has a wrapper that provide solutions for those interfaces. The middleware also maintains a schema table for each known elements in the whole system, and is responsible for updating the schema to fit any new incoming data. We will also propose a new indexing structure that is generic for relational, semi-structured and graph data. The new structure is able to expose semantic relations within data, enabling a rich validation feature for incoming queries.

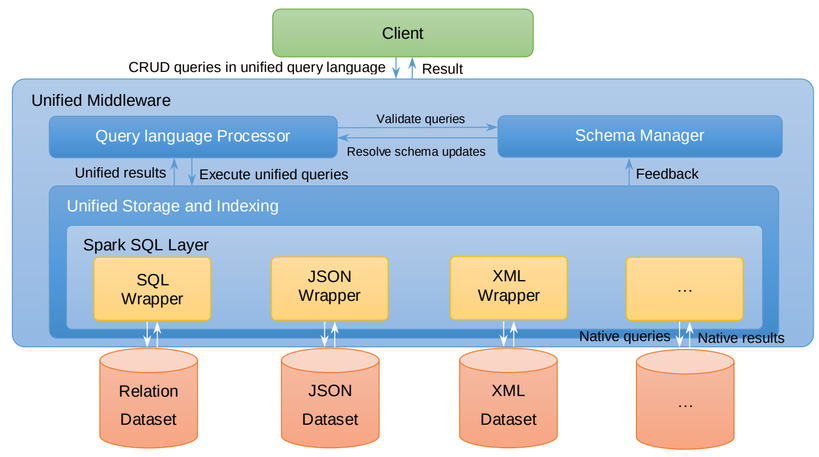

Figure 5: The UDBMS query middleware

TASK 4: SCHEMA EVOLUTION

Different data sources provide different formulas and constraints. Our further research will develop an effective schema that be able to:

- Fit different schemas in all data sources.
- Provide views for data sources do not have schema.
- Semantic-level validation for queries at compile time.
- Maintain update to schema according to incoming data.
- Resolve or provide suggestions for query errors brought by schema updates.

BRIEF COMPARISONS

Name of DBMS	Models	Query language	Flexible Schema	Query Validations
UDBMS	Relational, JSON, XML, graph	SQL-like	Yes	Yes
Oracle	Relational, JSON, XML	SQL, JSONPath, XQuery	Yes	No
MongoDB	JSON	API	Yes	Yes
Marklogic	Serializable, RDF, binary	XPath, XQuery, SQL-like	No	No

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
MATEMATISK-NATURVETENSKAPLIGA FAKULTETEN
FACULTY OF SCIENCE