CoreDB: a Data Lake Service

Amin Beheshti
University of New South Wales, Sydney, Australia
sbeheshti@cse.unsw.edu.au

Boualem Benatallah
University of New South Wales, Sydney, Australia
boualem@cse.unsw.edu.au

Reza Nouri
University of New South Wales, Sydney, Australia
snouri@cse.unsw.edu.au

Van Munin Chhieng
University of New South Wales, Sydney, Australia
vmc@cse.unsw.edu.au

HuangTao Xiong
University of New South Wales, Sydney, Australia
htxi394@cse.unsw.edu.au

Xu Zhao
University of New South Wales, Sydney, Australia
xzha451@cse.unsw.edu.au

ABSTRACT

The continuous improvement in connectivity, storage and data processing capabilities allow access to a data deluge from sensors, social-media, news, user-generated, government and private data sources. Accordingly, in a modern data-oriented landscape, with the advent of various data capture and management technologies, organizations are rapidly shifting to datafication of their processes. In such an environment, analysts may need to deal with a collection of datasets, from relational to NoSQL, that holds a vast amount of data gathered from various private/open data islands, i.e. Data Lake. Organizing, indexing and querying the growing volume of internal data and metadata, in a data lake, is challenging and requires various skills and experiences to deal with dozens of new databases and indexing technologies: How to store information items? What technology to use for persisting the data? How to deal with the large volume of streaming data? How to trace and persist information about data? What technology to use for indexing the data? How to query the data lake? To address the above mentioned challenges, we present CoreDB - an open source data lake service - which offers researchers and developers a single REST API to organize, index and query their data and metadata. CoreDB manages multiple database technologies and offers a built-in design for security and tracing.

CCS CONCEPTS

• **Information systems → Data management systems**; **Web services**;

KEYWORDS

Data Lake, Database Service, Data API

ACM Reference format:
Amin Beheshti, Boualem Benatallah, Reza Nouri, Van Munin Chhieng, HuangTao Xiong, and Xu Zhao. 2017. CoreDB: a Data Lake Service. In *Proceedings of CIKM'17 , Singapore, Singapore, November 6–10, 2017,* 4 pages.
https://doi.org/10.1145/3132847.3133171

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM'17 , November 6–10, 2017, Singapore, Singapore
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-4918-5/17/11…$15.00
https://doi.org/10.1145/3132847.3133171

1 INTRODUCTION

The production of knowledge from ever increasing amount of private/open data is seen by many organizations as an increasingly important capability that can complement the traditional analytics sources [2]. In this context, modern data-oriented applications are dealing with various types of data - unstructured, semi-structured and structured - such as emails, tweets, documents, videos and images. For example, consider an analyst who is interested in analyzing the Government Budget through engaging public's thoughts and opinions on social networks. To achieve this, the analyst may need to deal with a wealth of digital information generated through social networks, blogs, online communities and mobile applications which forms a complex data lake [6]: a collection of datasets that holds a vast amount of data gathered from various private/open data islands. Organizing and indexing the growing volume of internal data and metadata, in the data lake, is challenging and requires vast amount of knowledge to deal with dozens of new databases and indexing technologies.

In particular, for an analyst who is dealing with the data layer for organizing, indexing and querying different types of data - from structured entities to be stored in relational databases to large volume of open data to be organized using appropriate NoSQL databases such as MongoDB or CouchDB - various skills and experiences may be required: How to store information items (from structured entities to unstructured documents)? What technology to use for persisting the data (from Relational to NoSQL databases)? How to deal with the large volume of data being generated on a continuous basis (from Key-value and document to object and graph store)? How to trace and persist information about data (from descriptive to administrative)? What technology to use for indexing the data/metadata? How to query the data lake (from SQL to full-text search)?

To address the above mentioned challenges, we present CoreDB - an open source data lake service - which offers researchers/developers a single REST API to organize, index and query their data and metadata. CoreDB manages multiple database technologies (from Relational to NoSQL databases), exposes the power of Elasticsearch [5] and weave them together at the application layer. CoreDB offers a built-in design to support: (i) Security and Access Control: to provide a database security threats including authentication, access control and data encryption; and (ii) Tracing and Provenance [3, 8]: to collect and aggregate tracing metadata including descriptive, administrative and temporal metadata and build a provenance graph.

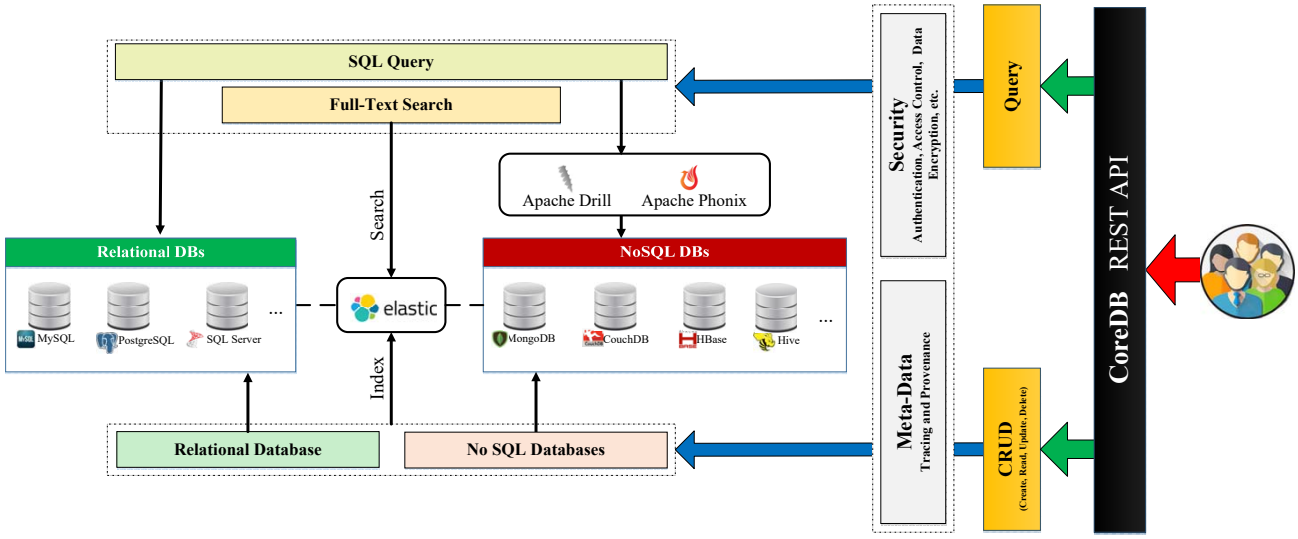

Figure 1: CoreDB Architecture.

The CoreDB API is available as an open source project on GitHub[1]. The rest of the paper is organized as follows. In Section 2, we present an overview of the CoreDB, while in Section 3 we describe our demonstration scenario.

2 COREDB OVERVIEW

CoreDB is an open source complete Database Service that powers multiple relational and NoSQL (key/value, document and graph stores) database-as-a-service for developing Web data applications, i.e. data-driven Web applications. CoreDB enables analysts to build a data lake, create relational and/or NoSQL datasets within the data lake and CRUD (Create, Read, Update and Delete) and query entities in those datasets. CoreDB exposes the power of Elasticsearch, a search engine based on Apache Lucene (lucene.apache.org/), to support a powerful index and full-text search. CoreDB has a built-in design to enable top database security threats (Authentication, Access Control and Data Encryption) along with Tracing and Provenance support. CoreDB weave all these services together at the application layer and offers a single REST API to organize, index and query the data and metadata in a data lake. Figure 1 illustrates the architecture and the main components of the CoreDB.

2.1 CRUD Data Lake, Dataset and Entity

The top-level organizing concept in CoreDB is the Data Lake: a collection of datasets that holds a vast amount of data gathered from various private/open data islands. Within the Data Lake one can create a dataset of type relational and/or NoSQL database. CoreDB offers a single REST API to create a set of datasets and weave them together at the application layer. To create a relational database, a database connection configuration operation has been provided to enable access to many of relational databases such as MySQL, PostgreSQL and Oracle. Moreover, CoreDB leverages appropriate NoSQL database such as MongoDB, HBase and HIVE to organize key-value, document and graph stores requirements. CoreDB persists the entities (structured and unstructured) in a JSON format, an easy-to-parse structure, for its growing adoption in the Web data applications. Considering the self-describing nature of JSON documents, in CoreDB we extend JSON with the option of defining a schema for all or part of the data. The following statements illustrate how to call the CoreDB service to create a Data Lake and a Dataset (Relational or NoSQL):

```
Create a Data lake:
curl -H "Content-Type: application/json" -X POST -d
'{"name":"DataLake_NAME"}' http://CoreDB/api/clients

Create a Dataset:
curl -H "Content-Type: application/json" -H "Authorization:
Bearer ACCESS_TOKEN" -X POST -d '{"name":"Dataset_NAME",
"type": "Database_NAME"}' http://CoreDB/api/databases
```

When creating a data lake, the 'DataLake NAME' parameter needs to be replaced by the user. When calling this service, an access token ('ACCESS TOKEN') will be returned which enables the user to access the data lake; and will be required for creating, reading, updating and deleting a dataset or an entity in the data lake. For example, to create a dataset (named 'dsTweets') for storing a set of tweets in MongoDB, the 'Dataset NAME' parameter can be replaced with 'dsTweets' and the 'Database NAME' parameter should be replaced with 'MongoDB'. CoreDB supports various relational and NoSQL databases such as MySQL, PostgreSQL, Oracle, MongoDB, HBase and HIVE. The URL ('http://CoreDB/') illustrates the Web address where the CoreDB service is deployed. The next step is to use the CoreDB service to CRUD entities:

```
Create an Entity:
curl -H "Content-Type: application/json" -H "Authorization:
Bearer ACCESS_TOKEN" -X POST -d '{"Param1":"Value1",
"Param2": "Value2", ...}' http://CoreDB/api/entity/
{Database_NAME}/{Dataset_NAME}/{Entity_TYPE}

Read an Entity:
curl -H "Content-Type: application/json" -H "Authorization:
Bearer ACCESS_TOKEN" -X GET http://CoreDB/api/entity/
{Database_NAME}/{Dataset_NAME}/{Entity_TYPE}/{id}
```

[1]https://github.com/unsw-cse-soc/CoreDB

```
Update an Entity:
curl -H "Content-Type: application/json" -H "Authorization:
Bearer ACCESS_TOKEN" -X PUT -d '{"Param1":"Value1",
"Param2": "Value2"}' http://CoreDB/api/entity/
{Database_NAME}/{Dataset_NAME}/{Entity_TYPE}/{id}

Delete an Entity:
curl -H "Content-Type: application/json" -H "Authorization:
Bearer ACCESS_TOKEN" -X DELETE  http://CoreDB/api/entity/
{Database_NAME}/{Dataset_NAME}/{Entity_TYPE}/{id}
```

When creating, reading, updating or deleting an entity, the 'Database NAME', 'Dataset NAME' and 'Entity TYPE' parameters need to be replaced by the user. For example, to CRUD a tweet in 'dsTweets' dataset stored in MongoDB the 'Database NAME', 'Dataset NAME' and 'Entity TYPE' parameters should be replaced with 'MongoDB', 'dsTweets' and 'Tweet' respectively.

2.2 Index and Query

Index. Full-text search is distinguished from searches based on metadata or on parts of the original texts represented in databases (such as titles, abstracts, selected sections, or bibliographical references). CoreDB exposes the power of Elasticsearch without the operational burden of managing it by developers. In particular, when the user enables indexing while creating a dataset, the entities will be automatically indexed for powerful Lucene queries.

Query. CoreDB enables the power of standard SQL with full ACID transaction capabilities for querying data held in not only relational databases but also in NoSQL databases. In particular, using a simple REST API, it is possible to send a SQL query to be applied to the datasets created in the data lake. To achieve this, in CoreDB, we leverages Apache Phoenix (phoenix.apache.org/) to take the SQL query and compiles it into native NoSQL store APIs. Moreover, to support queries that need to join data from multiple datastores in the data lake, we leverage Apache Drill (drill.apache.org/). Considering that Elasticsearch is a search engine based on Lucene, it is also possible to apply Wildcard, Fuzzy, Proximity and Range search queries in CoreDB. The following statement illustrate how to call the CoreDB service to apply a query (SQL or Full-text search) to the data lake:

```
curl -H "Content-Type: application/json" -H "Authorization:
Bearer ACCESS_TOKEN" -X GET http://coredbapi/entity/
{Database_NAME}/{Dataset_NAME}/{Entity_TYPE}?query={query}
```

For example, to find the tweets (stored in 'dsTweets' dataset persisted in the MongoDB database) that contains the keyword 'CIKM' the following query can be used.

```
curl -H "Content-Type: application/json" -H "Authorization:
Bearer ACCESS_TOKEN" -X GET http://coredbapi/entity/
dsTweets/MongoDB/Tweet?query="{"match": {"text":"CIKM"}}"
```

2.3 Security and Access Control

Database servers are the most important systems in virtually all organizations. They store critical information (e.g. Email, Financial data, Personal data, etc.) that is vital for organizations. CoreDB has a built-in design to support top database security threats (e.g. Weak Authentication and Weak system configuration). In particular, CoreDB supports: Identification and Authentication requirements, System Privilege and Object Access Control and Data Encryption. For example, each user may be identified and authenticated by the database system and has different access levels (e.g. create, read, update, and delete) to system entities by supporting Roles, Responsibilities and Privileges, System Privileges and Object Privileges. In CoreDB, privileges are provided directly to users or through roles. The following statements illustrate how to use CoreDB service to create a user and get an access token.

```
Create a User:
curl -H "Content-Type: application/json" -X POST -d '{"userName":
"USER_NAME",  "password": "PASSWORD", "role":"ROLE", "clientName":
"DataLake_NAME", "clientSecret":"DataLake_SECRET"}'
http://CoreDB/api/account

GET Access Token:
curl -H "Content-Type: application/json" -X POST -d '{"userName":
"USERNAME", "password": "PASSWORD", "grant_type": "PASSWORD",
"clientName":"YOUR_CLIENT", "clientSecret":"YOUR_CLIENT_SECRET"}'
http://CoreDB/api/oauth
```

After creating the user and receiving the access token, it is possible to use the following statement to grant an action (create, read, update, delete and query) to a specific role:

```
Define Access Control:
curl -H "Content-Type: application/json" -H "Authorization:
Bearer ACCESS_TOKEN" -X POST -d '{"role":{"action":"TRUE/FALSE}}'
http://coredbapi/api/{Database_NAME}/{Dataset_NAME}/{Entity_TYPE}
```

2.4 Tracing and Provenance

Tracing the entities over time is very important and assists analysts in understanding what time an entity was created, read, updated, deleted or queried (and who did this)? Where was the location (e.g. IP Address)? What was the platform (e.g. mobile or PC)? To address this important requirement, the CoreDB API provides a very useful and powerful functionality involving tracing historical data back to users. CoreDB offers a built-in design to collect and aggregate tracing metadata including descriptive, administrative and temporal metadata. We use the tracing metadata to build a provenance graph [3]: a directed acyclic attributed graph where the nodes are users/roles and entities and the relationships among them represents the activities such as created, read, updated, deleted or queried. The relationships will be tagged with metadata such as timestamp and location. The following statements illustrate how to call the CoreDB service to receive the provenance graph of a particular entity:

```
curl -H "Content-Type: application/json" -H "Authorization:
Bearer ACCESS_TOKEN" -X GET http://dataapi/api/entity/trace/
{Database_NAME}/{Dataset_NAME}/{Entity_TYPE}/{id}
```

The result will be a JSON file containing a finite set of triples (subject, predicate, object) representing the relationship between two entities in the provenance graph. For example, the triple (david, read[ts:20170320;ip:100.101.102.103], tweet123) represents that a user with unique id 'david' read a tweet with the unique id 'tweet123' on 20th of March 2017 using a computer with IP address '100.101.102.103'.

3 DEMONSTRATION SCENARIO

Governments at all levels are starting to recognize the value in their budgeting process. In this context, the budget is the single most important policy document of governments, where policy objectives are reconciled and implemented in various categories such as 'Health', 'Social-Services', 'Transport' and 'Employment'. In the demonstration scenario we present the requirement for an analyst who is interested in analyzing the Government Budget - specifically

the Health program - through engaging public's thoughts and opinions on social networks. The goal here is to properly link the data objects in social networks (e.g. tweets in Twitter[2]) to the health category of the budget. The demonstration scenario consists of the following parts:

(i) Data Definition and manipulation. The budget analyst will use the CoreDB service to create a data lake. Considering that the Australian budget 2016-17 handed on Tuesday 3 May, 2016; the analyst is interested in persisting all the tweets from one month before and two months after this date. The analyst will create a NoSQL dataset to store these tweets (more than 15 million tweets) in MongoDB. We illustrate that it is possible to fill this dataset entity by entity (a simple java code to read the entities and call the CoreDB service) or read the whole tweets (uploaded somewhere on the Web in a JSON format) and persist them in the dataset using the CoreDB service. Then, the analyst will be interested to create a relational dataset to store the main entities related to the budget health program such as registered doctors and nurses in Australia, Hospitals and Pharmacies, Health funds, Medical Devices, Drugs, Diseases and keywords related to health in MySQL database. These information later can be used to filter tweets related to health. To build this dataset, the analyst will create a set of users and access tokens: these credentials will be provided to a set of users who will help in populating this dataset. This scenario will help us to illustrate the tracing capability of the CoreDB.

(ii) Index and Query. In this part we illustrate the automatic indexing capability in CoreDB. We propose to the attendee a scenario where she would be able to use full-text search and SQL queries to find tweets that contain keywords such as instances of Hospitals, Drugs and Diseases stored in the relational dataset. Consider the data lake created in the previous step; following is a sample query for linking a tweet persisted in MongoDB to a tuple (in the Hospital table) stored in PostgreSQL database:

```
SELECT tweets.summary, tweets.user_id, tweets.date FROM
mongo.budget.tweets AS tweets INNER JOIN postgresql.health
AS healthDB ON tweets.hopitalId = healthDB.hostpotal.id
WHERE tweets.summary LIKE `%health%' AND tweets.body LIKE
`%Sydney Hospital%' AND  tweet.date BETWEEN `21-05-2016' AND
`21-08-2016Š AND healthDB.hostpital.name LIKE `%Sydney Hospital%'
```

Figure 2 illustrates the performance of this query: the experiment were performed on Amazon EC2 platform using instances running Ubuntu Server 14.04. The scalability experiment was done on a single machine, four machines and eight machines of type t2.large that provides 8GB of memory, 2 virtual CPUs and 20GB EBS storage; and on 15 million tweets. Notice that the query processing comes down to three phases (parsing, plan generation and plan execution), and the scalability study (in Figure 2) shows the impact on the execution phase. The result also shows that the parsing phase is costly specially when we have several joins among different databases in the data lake.

(iii) construct relationships among the data objects stored in MongoDB and MySQL. To properly analyze the tweets, the budget analyst may require to link tweets to the health related entities related to the budget health program. For example, as the result of the querying step, the analyst identifies a set of tweets which contain the Diabetes diseases. We illustrate how it is possible

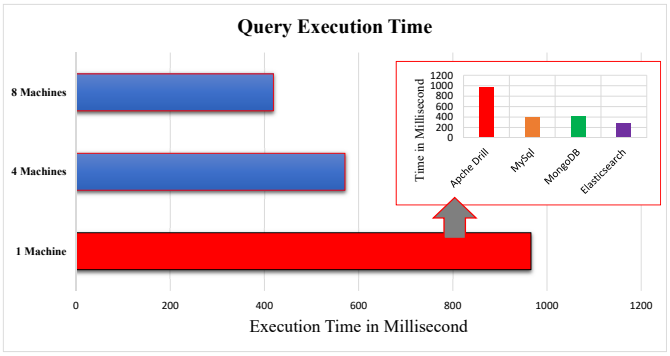

Figure 2: Sample query execution time.

to create a new graph dataset [7] in the data lake and store the '$tweet - -(contains) - - > diabetes$' relationship. (iv) security and tracing. In this part we propose to the attendee a scenario where she would be able to see the security (Identification and Authentication requirements, System Privilege and Object Access Control and Data Encryption), tracing and provenance capabilities of the CoreDB.

4 RELATED WORK AND CONCLUSION

The two closest systems to our work include AsterixDB [1] and Orchestrate (orchstrate.io/). The added value of CoreDB compare to these systems include: managing multiple database technologies (From Relational to NoSQL) and providing a built-in design for security and tracing. Moreover, CoreDB is available as an open source project and through a single REST API. As an ongoing work, we are extending the query component to support SPARQL[3] queries. We also plan to leverage our previous work [4] on data curation to enable CoreDB automatically curate the data items stored in the data lake, e.g. extracting features such as keywords and named entities and persist them in the data lake.

ACKNOWLEDGMENTS

This research was partially supported by ARC project LP0669090.

REFERENCES

[1] Apache. 2017. AsterixDB. https://asterixdb.apache.org/. (2017).
[2] Seyed-Mehdi-Reza Beheshti, Boualem Benatallah, Sherif Sakr, Daniela Grigori, Hamid Reza Motahari-Nezhad, Moshe Chai Barukh, Ahmed Gater, and Seung Hwan Ryu. 2016. *Process Analytics - Concepts and Techniques for Querying and Analyzing Process Data.* Springer.
[3] Seyed-Mehdi-Reza Beheshti, Hamid R. Motahari Nezhad, and Boualem Benatallah. 2012. Temporal Provenance Model (TPM): Model and Query Language. *CoRR* abs/1211.5009 (2012). http://arxiv.org/abs/1211.5009
[4] Seyed-Mehdi-Reza Beheshti, Alireza Tabebordbar, Boualem Benatallah, and Reza Nouri. 2017. On Automating Basic Data Curation Tasks. In *Proceedings of the 26th International Conference on World Wide Web Companion, Perth, Australia, April 3-7, 2017.* 165–169.
[5] C. Gormley. 2015. *Elasticsearch: The Definitive Guide.* " O'Reilly".
[6] Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Managing Google's data lake: an overview of the Goods system. *IEEE Data Eng. Bull.* 39, 3 (2016), 5–14.
[7] Mohammad Hammoud, Dania Abed Rabbou, Reza Nouri, Seyed-Mehdi-Reza Beheshti, and Sherif Sakr. 2015. DREAM: Distributed RDF Engine with Adaptive Query Planner and Minimal Communication. *PVLDB* 8, 6 (2015), 654–665.
[8] OPM. 2017. The Open Provenance Model. http://openprovenance.org/. (2017).

[2]https://support.twitter.com/articles/215585

[3]https://www.w3.org/TR/rdf-sparql-query/