# Unified Database API

Amit Kumar Gupta, Sanjitha Singh, Mohammed Ataaur Rahaman, Ravichandra H.
Computer Science Department, Dayananda Sagar College of Engineering
theamit97@gmail.com, sanjitha.singh@gmail.com, ataago7@gmail.com, ravichandra-cs@dayanandasagar.edu

*Abstract*— **In the last few years, cloud computing has gained a lot of traction. It has spearheaded a huge paradigm shift in the IT industry with respect to management of resources. Computing resources from storage to servers are offered on a pay-per-use basis. With the virtualization of resources and the pricing policies, development and deployment of applications over cloud has become popular by the means of Platform-As-A-Service (PAAS) architecture.**

**The emergence of cloud computing has also led to the production of a huge amount of data; therefore, a vast array of data-stores to store, access and manipulate this data. Each datastore is specialized for a specific type of data. For a truly dynamic and responsive application, multiple datastores are often used, since a single datastore cannot meet the requirements of complex applications. Each of these datastores has their own data representation model, query language, and interfacing APIs. Learning the semantics of each of these datastores and the syntax of their API is a challenging task for any programmer.**

**In addition to this, different vendors offering PAAS often provide different datastores. Anytime a developer migrates to a different vendor, he must migrate his data from the existing data store to a new unfamiliar one.**

**A simple solution to this heterogeneity is a common unified interface that enables the programmer to perform all the necessary database operations using a simple query language. Doing this decouples the actual database operations from the low-level implementation details of the numerous APIs, thus providing a comfortable level of abstraction for the user to work with.**

***Keywords—NoSQL, datastores, REST, polyglot persistence***

## I. INTRODUCTION

We adopt and improve upon the REST-based API proposed by Sellami et al.[1], called ODBAPI (OPEN-PaaS-DataBase API).ODBAPI uses its own data representation model that consists of Environment, Database, Entity Set, Entity and Attributes. An environment contains a pool of datastores and resources of type Database. Each database is made up of functional units called entity sets. Entity sets comprises entities, which in turn include attributes.

The implementation is based entirely on the REST architecture. All queries are performed as HTTP requests. The API provides a layer of abstraction since the user only interacts with the ODBAPI via REST requests. Based on the name and type of the database, ODBAPI interacts with the respective API to perform the necessary operation.

The purpose of this literature survey is to understand the other prominent approaches to solve this problem of heterogeneity of NoSQL databases and polyglot persistence. We evaluate them against our present approach and adapt our methodology by assimilating better ideas and overcoming any existing flaws.

We use the author's references as the starting point of our research. It is intuitive to look at the research process and gain insight into how the idea was conceived, so the process of redefining and optimizing it is simpler and evolutionary.

## II. LITERATURE REVIEW

### *A. Bigtable*

The paper shows that modern web applications and services have a need for a storage system that has varied demands. These demands pertain to data size, latency requirements and *also* real time processing. Thus, a flexible solution to all the above requirements are needed.
Fay Chang et al. propose the concept of Bigtable. Most Google products make use of *Bigtable* which achieve:

- Pertinency
- Quantifiability
- High performance
- High availableness

The paper states that Bigtable employs a unique model of data store, which is today similar to a No-SQL relational store. It's the user's responsibility to structure the data by themselves

### *B. PNUTS*

The paper outlines modern management challenges seen in net applications.
Tasks like:

- managing session state
- content meta-data
- user-generated content like tags and comments.

Lower consistency is usually tolerable for net applications however measurability, systematically smart latency is said to be important according to the paper.

The paper we chose proposes Yahoo! PNUTS. It solves all the above-mentioned issues and many more. Some of the key takeaways from the paper were:

- easy relative model to users
- Multi-level redundancy
- Async operations
- Service of knowledge management

The paper states however that many tradeoffs had been made between practicality, performance and measurability that have to be compelled to be created. PNUTS additionally restricted the options to those who were really required and will be provided protective dependableness and scale. There are no details on snap and info migration.

### *C. Dynamo*

Amazon DynamoDB is a No-SQL based database. It is a key-value and document type store, which is famous for being able to scale very well and deliver great performance regardless. The performance level is usually comparable to less than 10 milliseconds.

Reliability at massive scale is one of the biggest challenges faced by tech conglomerates. Consumer trust and customer satisfaction takes a hit with the smallest discrepancies in third party cloud services. These services work at a massive scale, all around the world. The count of servers could be in the tens of thousands. Having a reliable persistence state at a scale of the level mentioned before poses a monumental challenge for the providers of this service.
Some of the solutions provided by dynamo are:

- Interface based solely on primary keys.
- Consistent hashing, which is the same technology used in load balancing servers is used to achieve availability along with scalability.
- Minimal need to manually administer the database
- Decentralized architecture
- Object versioning.

One shortcoming of Dynamo is that it can only be deployed on AWS. This is essentially vendor lock in and prevents some level of freedom and flexibility that many developers prefer. It cannot be installed on individual desktops/servers, or custom containers. It has limited querying abilities. It does not make any provisions for joins, triggers, or server-side scripts. Thus, we conclude that DynamoDB does not suit the requirements of most applications as a sole data store solution.

### *D. Database Scalability, Elasticity, and Autonomy in the Cloud*

Modern web services and applications have large data requirements and those include having to be able to store a large volume and variety of data in real time. This data storage service must be scalable, reliable and also be highly accessible.

Many different techniques and guidelines have been given in the paper to navigate the above issues. Techniques for scaling such as vertical and horizontal which employs data sharding and data redundancy.
Apart from that, the paper also addresses issues seen during data migration and how we can maintain database performance when moving data storage solutions and maintain the above-mentioned requirements.

It was also stated that DBMS needs to be autonomic and have some level of intelligence to it. Reducing manual administration at a large scale is important.

### *E. Polyglot Persistence*

With modern applications that collect not only a large amount of data, but also collect and store not only a large volume of data but a large variety, a single database storage system does not suffice. Different types of data
are accessed and has different transactional requirements, such as some data needs high availability and some needs to be highly consistent in real time. This cannot be done using the traditional method of having only one type of data store.

Polyglot Persistence means that when storing data, it is best to use multiple data stores, chosen based upon the

way the components of a single application data is being used or components of a single application.
Having different data stores for different data may introduce the problem of learning and maintaining different technologies but that is well
worth it. For example, sensitive data that needs to be highly available would use a relational data store, whereas data which could need eventual consistency would use MongoDB for example.

### *F. Multiple Data Stores*

The paper states that in the field of Cloud computing, pools of services supporting different platforms and mechanisms to streamline development have been developed. These services also streamline execution environments, encouraging an upload and forget environment. This set of services is called platform as a service (PaaS).

PAAS systems strive to achieve:

- Elasticity
- Scalability
- Portability

The modern challenge of supporting multiple data stores is also addressed.
The paper provides a solution in which applications which use PAAS services do not have to worry about the underlying implementations of different data stores and servicing them. This saves big on server maintenance and servicing.
Encapsulation of many back-end services and environments has been provided by PAAS services.

### *G. Disinterest for enterprises towards NoSQL*

During the initial conception of NoSQL databases, the enterprises failed to see why anyone would want a schema less datastore that did not enforce any of the ACID properties. StoneBraker et al. interview different enterprises to arrive at reasons why the switch to NoSQL was hesitant.

#### *1) Lack of ACID enforcement*

ACID properties are a set standard for databases. They ensure that the database remains trustworthy. Specially in real time systems where a single wrong transaction could affect numerous other applications.
NoSQL uses horizontal scalability which implies that redundancy is inevitable. But redundancy has been undesired for a long time and introduction of this deliberately seems wrong.

#### *2) Low-Level Query Languages*

NoSQL datastores uses low level query languages which seem like a regression from using high-level query languages like SQL. SQL has a simpler language. You decide what you want to fetch, from where, and the conditions, if any. Low-level query languages can be unintuitive and have a lower usability. This makes reverting from SQL to NoSQL feel like downgrading.

#### *3) Lack of Standards*

There are a ton of NoSQL engines, each offering a different User Interface, and each following a different conceptual model. Some of these do not have a specific schema, which despite being a selling point for a few applications, becomes cumbersome for traditional applications. Then, having these many APIs that come along with them, the programmers can't really learn all of these interfaces and APIs. Compared to this SQL has a set standard that it follows.

### *H. JDBC - Java Database Connectivity*

JDBC is an application programming interface used for connecting Java to relational databases. It provides a common interface for accessing relational databases. That is, regardless of the implementation details and syntax of query languages for different databases, the API function calls remain exactly the same. This provides a level of abstraction for the user to work with. JDBC allows users to perform a variety of database operations from retrieving to updating to deleting.
Although it primarily supports all flavours and dialects of SQL, it does not adhere entirely to it. It can interact with a good number of tabular databases.
The working of JDBC is pretty straightforward and simple. A secure connection is established with the data source. The queries are written as function calls using the API. The results are processed and returned in the format that is expected by the programming language.
JDBC has a three-tier architecture which makes it ideal for cloud computing due to its increased performance. The middle-layer acts as the interface between the client machine (top layer) and the datastore (bottom layer). The queries written are sent to the middle-tier by means of a function call. These queries are mapped to the appropriate queries in the individual databases and are executed in the bottom layer. The results are returned to the middle-layer which returns it back to the client.
JDBC is geared only towards relational databases and cannot interface with NoSQL databases. It can sometimes incur too much overhead due to the middle-tier, which makes it hard for large-scale dynamic applications.

### *I. Big Data leading to NoSQL*

It is common knowledge that data is growing exponentially. With a tremendous amount of data coming in from vast sources, sources such as social media, weather stations, online shopping sites, customer behaviour analysis, the term BigData is used to refer to data with high-volume, high-velocity and high-variety. Obviously conventional databases fail in handling data of this nature. Which is why NoSQL and other alternative methods such as NewSQL and Search-Based systems have been developed.

NoSQL databases, unlike their relational counterpart, can deal with semi-structured and unstructured data which constitute the majority of BigData. It can also parallelly process large-scale distributed data. Instead of ACID (Atomicity, Consistency, Isolation, and Durability) properties, it makes use of CAP (Consistency, Availability, and Partition-Tolerance) theorem, to increase horizontal scalability.

Different types of NoSQL databases exist for storing different types of data. Some of the common NoSQL datastores are Key-Value, document, column and graph.

By using algorithms such as Map-Reduce in association with NoSQL databases, the data can be processed at high speeds to enable the gaining of insights for business, customer satisfaction, and many such trends.

### *J. Spring Data Framework*

The Spring Framework is an open source framework built on Java Persistent API. It is primarily used for development of applications on the Java platform. The main goal of the spring framework is to provide a unified interface for both SQL and NoSQL stores Spring initially started out as a dependency injection framework for building decoupled systems. Spring connects classes by using an XML files in a way. It makes sure that the objects are instantiated and initialized and inserted in the right places. Spring wraps excellent Java libraries in a simple, elegant way, ready to be used in application. Spring defines abstractions for each database and datastore and provides overarching abstractions for implementation. The developers of Spring Data, SpringSource, have programmed POJOs to help in the non-intrusive integration of spring with over applications. Spring has a large number of useful subprojects like Spring MVC, Spring Security and Spring WebFlow.

The major drawback in using the spring framework is that the addition of a new data store is tough due to the solution being linked strongly to the Java programming model.

### *K. The SoS Platform*

Atzeni et al. propose a common programming interface to access NoSQL and relational data stores referred to as Save Our Systems (SOS). It performs three general operations: put to insert objects, get to retrieve objects and delete to remove objects. There are three modules:

i) the common interface

ii) the meta-layer that stores the data and intermediates between the common interface the low-level handler

iii) the handlers that generate the appropriate calls to the specific database system.

Although the paper claims that it can offer extensibility and scalability, there is no proof of this included. It also does not include any means of including relational databases. The authors confess that there are multiple trade-offs with their approach when this is extended to relational databases.

### *L. ONDM*

ONDM (Object-NoSQL Datastore Mapper) is a simple extensible framework based on the Java Persistent API . It provides developers with a unified access to a variety of NoSQL systems. It enables flexible mapping of application data to the data representations of the different datastores. The data for the ONDM is represented by entities, value objects, relationships, and aggregates.

ONDM is based entirely on the NOAM architecture which is a system-independent approach for NoSQL database design. In NOAM architecture, data is organized into blocks and entries. Blocks are larger units made up of smaller entries. If we consider a key-value database, a chunk of key-value data is the block and a single key-value pair is the data. A group of blocks form a collection.

Aggregate classes are represented by collections. Aggregate objects as blocks.

The input for the NOAM model is an application dataset made of aggregate objects. The input is mapped to an intermediate representation in the data model. Datastore adapters are used for implementation according to the data structures of the target NoSQL system.

However, the prototype has a simpler architecture than the one they have described in the paper. It also does not include custom data representations. ONDM does not take into consideration relational databases.

### *M. A REST- API for database-as-a-service systems*

Haselmann et al. present a universal REST-based API concept which is very similar to the one that is used in our base-paper. This API allows to interact with different Database-as-a-Services (DaaSs) regardless of whether they are relational DBMSs or NoSQL DBMSs. Since it is a REST-based API, it implies the use of HTTP and XML. This approach is preferred over SOAP, since it is easier and more intuitive. Also, the development testing and implementation requires less tools.

Here, the data is represented as entity, container and attribute. Entities are XML documents that contain attributes. Entities are stored in containers.

The REST implementations are as follows:

- PUT is used for creating entities and querying

- Get is for viewing all the entities
- Post is for updating entities

The Querying process is done using sources and filters. Sources are containers or entities. Filters are used for refining queries to include WHERE clause and AND clause. The output of the query can be in XML, csv, or json format.
Theoretically, this API can perform either CRUD operations or complex queries execution. But the authors have remained at the conceptual level and have not provided any prototypical implementation. In addition to this, the authors highlight the other problems with this approach. Using XML makes the API more complicated since the overhead of parsing and shredding is high. There is also no support for the concepts of transaction and transaction management.

### *N. Data Integration over heterogenous data sources*

This paper deals with the arduous task of data integration. Manolescu et al. outline the process of using several independent data sources as if they were a single source with a single global schema.
A user query is formulated in terms of the global schema. The query is translated into subqueries that are expressed in terms of the local schema. It executes the subqueries, retrieves the sub- results and combines them into a final result.
It uses a mediator architecture, which makes the entire process distributed and decoupled. Xquery is used for inputting queries. The processing of queries happens in three stages:

- Normalization : the query is minimized into a form that is translatable to SQL
- Translation : the normalized query is translated
- Rewriting : the context is understood and the query is rewritten entirely.

This is a challenging task since SQL queries cannot capture all potential queries of all the datastores. The authors outline various optimization techniques for algorithms to increase their efficiency which is used in our base paper.
The queries that are parsed and generated are executed using special wrapper classes for DOM and relational purposes.
However, this proposal only deals with integration of schema and does not hold well for schemaless datastores. Hence, they cannot support NOSQL data stores. Use of XML also incurs some overhead, which despite the optimized algorithms has a high time complexity.

### *O. Integrating relational and document-centric data stores*

Roijackers et al. propose a hybrid mediator approach for integrating relational and document data stores. It is a basic framework for a coordinated and uniform querying across SQL and NoSQL stores. In this way, they intend to interrelate SQL and NoSQL stores, and bridge the gap between them. It eliminates the need for impromptu user intervention in the process of query formulation, reformulation and optimization. In their approach, they embed documents inside relational data stores to give rise to what is known as hybrid data stores. They suggest an extended-SQL language called NoSQL query pattern (NQP) to query the datastores. For the document-centric data, they allow the users to define the condition. The rest of the SQL query, however, corresponds to NoSQL data by means of variable binding. Due to the overhead that is incurred for query translation, they provide effective algorithmic solutions for these translations. In addition to this, they also present optimization strategies for processing queries. Their approach includes providing a logical representation of NoSQL data in the relational model, and an extension of query language for SQL.
In addition to this they have provided an extensive empirical study that demonstrates how feasible the proposed framework and the proposed implementation is.
This approach is used for the implementation of virtual datastores in our application.

## III. Conclusion

In this paper we have looked at different approaches to dealing with the heterogeneity of datastores. We started off from the evolution and growth of NoSQL databases to their early criticism and rise to popularity due to their advantage in building scalable applications. We looked into Bigtable, Dynamo and PNUTS as starting points which led us to exploring the goals of cloud applications and the methods to achieve them. The importance of Polyglot Persistence was understood along with the assorted tactics for working with polyglot persistence. We discovered multiple other attempts at unified APIs such as ONDM, SOS and another REST-based API. Their shortcomings were discussed namely, lack of support for relational databases, lack of implementation evidence and lack of methods to perform complex operations. Data integration techniques were evaluated, and a hybrid mediator architecture was proven to be better than a simple local-as-a-view mediator architecture.

## IV. References

[1] ODBAPI: a unified REST API for relational and NoSQL data stores, Rami Sellami, Sami Bhiri, and Bruno Defude

[2] C. Fay and et al., "Bigtable: A distributed storage system for structured data," ACM Trans. Comput. Syst., vol. 26, no. 2, 2008.

[3] B. F. Cooper and et al., "Pnuts: Yahoo!'s hosted data serving platform," PVLDB, vol. 1, no. 2, pp. 1277–1288, 2008.

[4] G. DeCandia and et al., "Dynamo: amazon's highly available key-value store," in Proceedings of the 21st ACM Symposium on Operating Systems Principles, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007, pp. 205–220.

[5] D. Agrawal, A. El Abbadi, S. Das, and A. J. Elmore, "Database scalability, elasticity, and autonomy in the cloud - (extended abstract)," in Database Systems for Advanced Applications - 16th International Conference, DASFAA 2011, Hong Kong, China, April 22-25, 2011, Proceedings, Part I, 2011, pp. 2–15.

[6] P. Sadalage and M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, ser. Always

[7] R. Sellami and B. Defude, "Using multiple data stores in the cloud: Challenges and solutions," in Data Management in Cloud, Grid and P2P Systems - 6th International Conference, Globe 2013, Prague, Czech Republic, August 28-29, 2013. Proceedings, 2013, pp. 87–98.

[8] M. Stonebraker, "Stonebraker on nosql and enterprises," Com- mun. ACM, vol. 54, no. 8, pp. 10–11, 2011.

[9] M. Fisher, J. Ellis, and J. C. Bruce, JDBC API Tutorial and Reference, 3rd ed. Pearson Education, 2003.

[10] A. B. M. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics - classification, characteristics and comparison," CoRR, vol. abs/1307.0191, 2013.

[11] M. Pollack, O. Gierke, T. Risberg, J. Brisbin, and M. Hunger, Eds., Spring Data. O'Reilly Media, October 2012.

[12] P. Atzeni, F. Bugiotti, and L. Rossi, "Uniform access to non- relational database systems: The sos platform," in Advanced Information Systems Engineering - 24th International Con- ference, CAiSE 2012, Gdansk, Poland, June 25-29, 2012. Proceedings, 2012, pp. 160–174.

[13] L. Cabibbo, "Ondm: an object-nosql datastore mapper," Faculty of Engineering, Roma Tre University. Retrieved June 15th, 2013.

[14] T. Haselmann, G. Thies, and G. Vossen, "Looking into a rest- based universal api for database-as-a-service systems," in 12th IEEE Conference on Commerce and Enterprise Computing, CEC 2010, Shanghai, China, November 10-12, 2010, 2010, pp. 17–24.

[15] I. Manolescu, D. Florescu, and D. Kossmann, "Answering xml queries on heterogeneous data sources," in Proceedings of the 27th International Conference on Very Large Data Bases, ser. VLDB '01, 2001, pp. 241–250.

[16] J. Roijackers and G. H. L. Fletcher, "On bridging relational and document-centric data stores," in Big Data - 29th British National Conference on Databases, BNCOD'13, 2013, pp. 135–148.