DOI 10.29042/2018-4441-4445

Helix Vol. 8(6): 4441- 4445

A Uniform Query Framework for Heterogenous Data Management Systems

[*1]Jayant B. Karanjekar, [2]Manoj B. Chandak
[*1,2]Shri Ramdeobaba College of Engineering and Management, Katol Road, Nagpur, India
[*1]BuildDirect Technologies Inc., 401 W Georgia St, Vancouver, BC, V6B 5A1, Canada
Email: jayantkaranjekar@builddirect.com

Received: 20th September2018, Accepted: 11th October 2018, Published: 31st October 2018

Abstract
Most organizations today store their data in a variety of data storage systems such as relational databases, columnar stores, document stores, text search engines, and so forth. The adoption of multiple data storage systems tailored towards specific needs has its own challenges. The developers using these specialized systems have to often integrate several of them together. In the absence of a uniform framework, multiple engineers have to independently develop optimization logic and support different data retrieval mechanism for each system. The framework proposed in this paper intends to solve these problems by providing a uniform query framework to retrieve data from heterogenous systems. This paper also analyzes the response time of query framework with respect to individual data sources.

Keywords
Data Management, Big Data, Heterogenous Data Source, Information Retrieval, NoSQL, Query Framework, RDBMS

Introduction:

For over four decades relational database management systems have dominated the industry of data storage systems. However, there is a lot of interest and adoption of other of other types of data storage systems over the last decade such as document stores, columnar stores, stream processing engines, text search engines etc. In 2005, Stonebraker and Çetintemel [1] argued that these new types of data storage systems, also known as NoSQL databases, would bring an end to the "one size fits all" paradigm and can offer cost effective performance. Their argument today seems more relevant than ever as many specialized open-source data systems have since become popular such as Cassandra [2] (columnar store), MongoDB [3] (document store), Elasticsearch [4] (text search), etc.

As organizations have invested in data processing systems tailored towards their specific needs, they are faced with some challenges. The developers using these specialized systems have to often integrate several of them together. In the absence of a uniform framework, multiple developers have to independently develop optimization logic and support different data retrieval mechanism for each system.

The challenge for database administrators is to choose the right database management system from a wide variety of available modern systems. Because relational model has existed for such a long time, it has the advantages of better documentation, familiarity, simplicity, reliability and data integrity. They also perform better while computing complex aggregates and queries than other solutions [5]. These databases are also best suited for transactions with total compliance to ACID properties. The downside of relation databases is that they cannot scale out horizontally to multiple servers. Because of their inherent complexity of organization, it is difficult to efficiently expand a database [6].

On the other hand, NoSQL systems have been widely accepted in recent years because of their ease of scalability and availability. NoSQL databases are either columnar, key-value based or document-based. They also support many of the functions of relational databases such as indexing, querying, sorting, and projecting. However, when it comes to complex database queries, they are not as effective as relational databases. Transactions that require atomicity are not supported and updates to the number of redundant copies of the data is an eventual process. The ability to quickly scale out and scale in horizontally gives administrators operational flexibility with a loose coupling with underlying infrastructure [7].

The different types of NoSQL databases that differ so vastly from each other make it very difficult to have uniform standards among all of them. For several decades before NoSQL databases became popular, there was only one query language i.e. SQL for all relational database systems. However, now many query languages are coming up for different types of databases. This paper demonstrates how an SQL style query mechanism can be used to fetch data from both relations and NoSQL database systems. It also supports filters, joins, aggregates and nested queries.

Our previous paper [8] we proposed an SQL-like query mechanism for a combination of relational and NoSQL databases such as MySQL, MongoDB, Cassandra and CSV files. In this paper, we not only dive deeper into the architecture and optimization details but also analyze the response time of the framework and compare it with that of individual systems.

Architecture

Fig. 1 shows the architecture of the framework consisting of the central component called framework layer. The framework layer in turn talks to various databases that are part of the system. The architecture offers flexibility to add or remove nodes for any particular database without impacting other database systems.

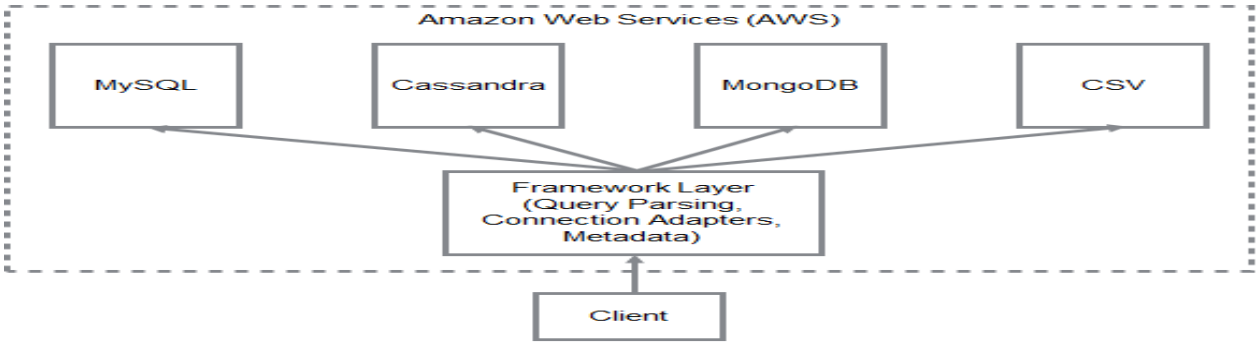

Figure 1: System Architecture

Individual database systems are meant for storing data whereas the framework layer is responsible for maintaining metadata, handling client requests, query processing and returning the response to the client.

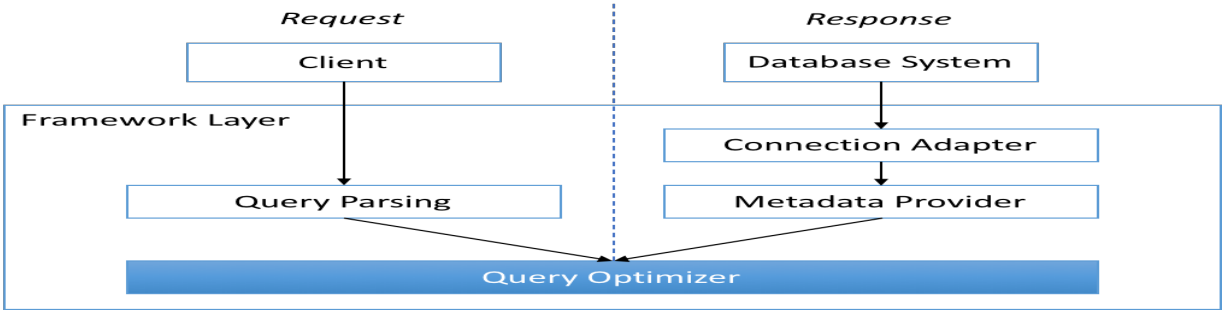

Figure 2: Framework Layer Interactions

Figure 2 outlines the main components of the framework layer. Once the request is received by the framework layer, it goes through the query parser that translates the SQL query into a tree of relational operators. Query optimizer applies query transformation rules on the relational operators to come up with an optimized execution plan. During the optimization process, metadata plays an important part. It guides the optimizer towards the goal of reducing the cost of overall query plan.

Query Parsing and Optimization:

The incoming query is parsed by the framework layer using an open source parser Calcite [9] and converted into relational algebra. The relational algebra is used as the basis for query optimization by applying planner rules.

Consider the following query consisting of a join of between a MySQL table called “weekly_sales” and a Cassandra table called “stores”.

```
SELECT cass."store_type",
        SUM(mysql."weekly_sales")
FROM  "mysql"."weekly_sales" mysql, "cassandra"."stores" cass
WHERE  mysql."store_number" =
        cass."store_number"
        AND mysql."week_date" = '2011-11-25'
GROUP  BY cass."store_type";
```


The initial query plan after parsing this query will be to fetch data from both the tables, join them and then to apply filter on the data resulted from the join as shown in Fig. 3(a). This query plan produces the result as expected however the framework goes one step ahead and tries to optimize the plan further. The optimization returns correct results but at the same time it also makes it efficiently with reduced cost.

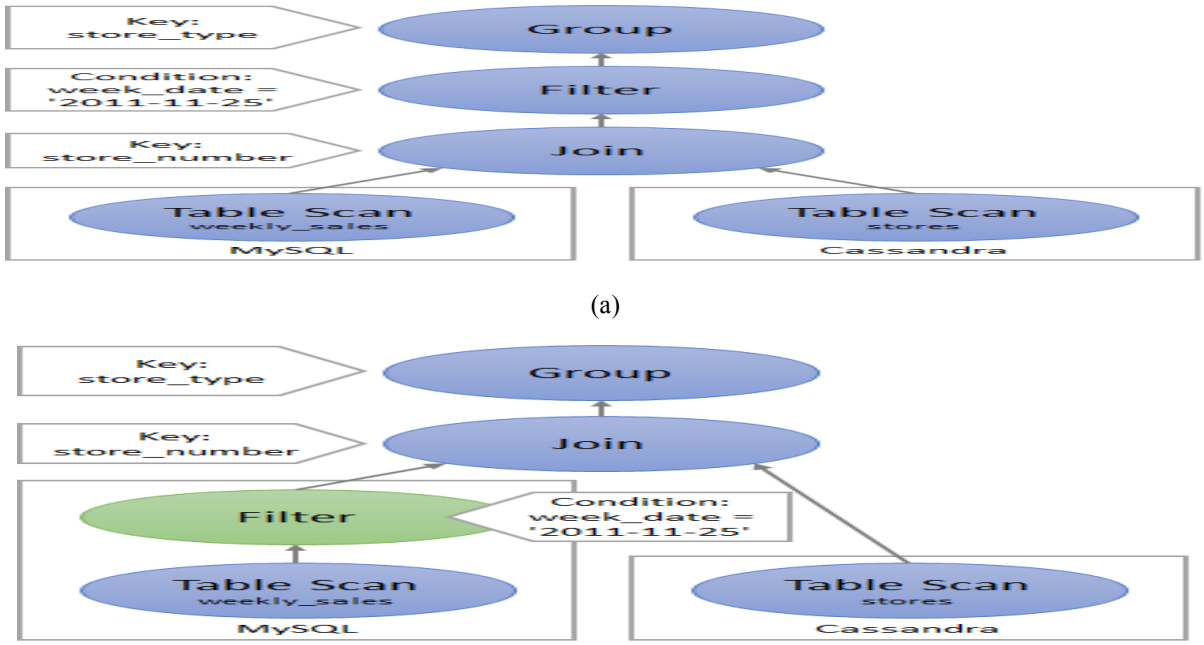

Figure 3: Execution Plan Before and After Optimization

After applying transformation rules for optimization, the plan changes as shown in Fig 3(b). The filter operation is pushed to the source database i.e. MySQL before performing join. This reduces the cost of join operation as the data processed by it is reduced by the filter. The transformation rules result into pushing the filter into the source database if the filter does not require columns from other sources.

Response Time Analysis:

As we have discussed in our previous paper [7], this framework supports filters, aggregates, nested queries and joins not only within same source of data but also across multiple heterogenous data sources. While a detailed performance analysis is out of scope of this paper, we do measure the response time of the framework and compare it with the response time for the same queries executed on the data source directly.

Materials and Methods

The data sources used in this analysis are Cassandra [2], MongoDB [3], MySQL [10] and CSV files. These data sources have been setup on Amazon Web Services (AWS) cloud environment on Elastic Cloud Compute (EC2) instances. The framework layer has been setup on memory optimized EC2 instances on AWS cloud. The storage used is AWS General Purpose SSD (GP2) with a baseline of 100 IOPS and burstable to 3000 IOPS. All the instances are placed within same availability zone to avoid network latency.

The dataset includes both highly structured relational database and also semi-structured data model. We are using historical sales data released by Walmart for 45 of their stores located in different regions [11] for structured data and a dataset containing user comments released by Reddit [12] for semi-structured data.

Results

Following table provides the response time of filter, join, aggregate, sort, text search and inter-database joins. The response time is an average of 100 executions of the query on each of the systems.

Table 1 shows the average response time comparison for a simple two-column SELECT statement with a filter. The

filter action has been performed on each of the data sources directly and also through the framework.
Table 2 shows average response time comparison for aggregate operation with GROUP BY clause. Since aggregates are no supported by MongoDB and Cassandra, the comparison is presented only for MySQL database.

SELECT query with a filter	Average Response Time (milliseconds)
MySQL Direct	70.351
MySQL through framework	92.932
MongoDB Direct	159
MongoDB through framework	188.98
Cassandra Direct	504.037
Cassandra through framework	603.025

Table 1. Average Response Time for a SELECT Query with a Filter

Aggregation with GROUP BY clause	Average Response Time (milliseconds)
MySQL Direct	84.245
MySQL through framework	110.311

Table 2: Average Response Time for Aggregate Operation with a GROUP BY Clause

Table 3 shows average response time for a two-table join within the same data source along with sort with an ORDER BY clause. Since joins are not supported by Cassandra and MongoDB, the comparison is presented for MySQL database.

Join and sort with ORDER BY clause	Average Response Time (milliseconds)
MySQL Direct	93.969
MySQL through framework	133.916

Table 3: Average response time for join within same data source and sort with ORDER BY clause

Table 4 shows average response time for text search through semi-structured data. Since a text search with LIKE condition is not supported in Cassandra, the comparison is presented only for MySQL and MongoDB.

Text search on semi-structured data	Average Response Time (milliseconds)
MySQL Direct	2224.491
MySQL through framework	3803.444
MongoDB Direct	13102.778
MongoDB through framework	16565.333

Table 4: Text Search on Semi-Structured Data

Table 5 shows the average response time for inter-database operations through the framework. The operations performed are simple join, filter, aggregation and nested queries.

Inter-database operations	Average Response Time (milliseconds)
Inter-database join	408.889
Inter-database join with aggregate	432
Inter-database join with filter	492.889
Inter-database join with nested queries	1782

Table 5: Inter-Database Operations

Conclusion

Advance data management systems and associated practices continue to evolve resulting in organizations embracing multiple heterogenous database systems. At the same time, relational data sources, accessed through SQL, remain an essential means to how organizations work with the data. In such a scenario, a uniform framework provided in this paper can play an important role with its support for traditional relational databases as well as NoSQL database systems. It should also be noted that some of the operations such as text search, joins, aggregates which are not natively supported on NoSQL databases can be performed through this framework. For instance, the proposed framework allows filtering on Cassandra tables although Cassandra does not have native support for it. Similarly, the framework allows joins operations on MongoDB collections although MongoDB does not have native support for

such operations.

The results shown above suggest that the time taken by the framework is more than the time taken by the query if executed on the data source itself. This increased time can be attributed to the time taken by the framework for query parsing, validation and optimization. There is scope for further analysis and performance tuning of the framework. On the other hand, the concept of having a uniform query framework for multiple heterogenous data sources and the ability to perform operations such as inter-database joins, aggregates, filters and nested queries is extremely useful and can change how organizations integrate various database systems.

References

[1] Michael Stonebraker and Ugur Çetintemel. 2005. "One size fits all": an idea whose time has come and gone. In 21st International Conference on Data Engineering (ICDE'05). IEEE Computer Society, Washington, DC, USA, 2–11.

[2] Avinash Lakshman and Prashant Malik. 2010. Cassandra: A Decentralized Structured Storage System. SIGOPS Oper. Syst. Rev. 44, 2 (April 2010), 35–40.

[3] Mongo. MongoDB. (Apr. 2018). Retrieved April 24, 2018 from https://www.mongodb.com/

[4] Elastic. Elasticsearch. (Apr. 2018). Retrieved April 24, 2018 from https://www.elastic.co

[5] Rick Cattell. 2010. Scalable SQL and NoSQL Data Stores. SIGMOD Record 39, 4 (December 2010), 16 pages. DOI:10.1145/1978915.1978919

[6] David DeWitt, Avrilia Floratou, Jignesh Patel, Nikhil Teletia, Donghui Zhang. Can the Elephants Handle the NoSQL Onslaught? Proceedings of the VLDB Endowment 5, 12 (August 2012), 12 pages.

[7] Chi-Hung Chi, Guillaume Pierre, Zhou Wei. Scalable Join Queries in Cloud Data Stores. 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (2012), Pages 547-555. DOI:10.1109/CCGrid.2012.28

[8] J.B. Karanjekar and M.B. Chandak. 2017. Uniform Query Framework for Relational and NoSQL Databases. CMES: Computer Modeling in Engineering & Sciences, Vol. 113, No. 2, pp. 177-187, 2017

[9] Calcite. Apache Calcite. (May. 2018). Retrieved May 3, 2018 from https://calcite.apache.org/docs/

[10] MySQL. MySQL. (May. 2018). Retrieved May 3, 2018 from https://www.mysql.com/

[11] Walmart Data. Walmart Sample Data. (May. 2018) Retrieved May 3, 2018 from https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/data

[12] Reddit Comments. Reddit Comments Data May 2015. (May. 2018). Retrieved May 3, 2018 from https://www.kaggle.com/reddit/reddit-comments-may-2015

