Real-time market data warehousing platform for data visualization

P.G.T.L. JAYASEKARA
2021

Real-time market data warehousing platform for data visualization

A dissertation submitted for the Degree of Master of Computer Science

P.G.T.L. Jayasekara
University of Colombo School of Computing
2021

DECLARATION

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has also not been submitted for any degree in any university previously.

Student Name: P.G.T.L.Jayasekara

Registration Number: 2018/MCS/040

Index Number: 18440407

_ __(2021/09/05)_

Signature of the Student & Date

This is to certify that this thesis is based on the work of Mr. /Ms. ____P G T L Jayasekara_____ under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by,

Supervisor Name:

____DR.H. A. Caldera_____

Signature of the Supervisor & Date  30-11-2021

I would like to dedicate this thesis to all who supported and encouraged me

ACKNOWLEDGEMENTS

I would like to thank the following people, without then I would not have been able to complete this project, and would not have made it through my Masters degree!

Special thanks for my supervisor: DR. H.A. Caldera for providing guidance and feedback throughput this project. Thanks for Ms. Chathuri Geekiyanage, Architecture of LSEG Technologies for suggesting me this project as my Master degree project. As well as, I would like to thank for my parents and my Husband Thilina, for encouraging me and supporting me to manage all my office works and home works when required to manage time for my Masters studies.

ABSTRACT

Data warehousing is a data management system that supports business intelligence activities like analytics. It collects data from heterogenous data sources and store this data while providing capability to analyze users. The existing warehousing system in the LSEG surveillance system is not supported for market data analysis on real-time basis. It keeps separate component to analyze and commit that data to storage. This is an outdated model as existing warehousing model has limited features when compared to modern warehousing technologies.

Through out critical analysis of utilization, limitations, benefits of modern technologies for warehousing system implementation, technical design has been finalized with two hypotheses.

Experimental based analysis has been performed to satisfy two hypothesis – new system developing with Apache Impala and Kudu can perform faster query response than existing system, and the new system can perform analytical queries for generating data summarization graphs without help of third component for data summarizing. With set of proper experimental query execution for both existing and new system above two hypotheses have been satisfied successfully.

But since the algorithm used for data summarization by existing summarization process is client legacy algorithm, the two summarization graphs don't equal exactly point to point– but it has been able to provide visual summarization output like existing summarization graph. So, implementing existing summarization methodology can be model as UDF which is fallen into future works of the project.

Key words: Data Warehousing, LSEG, real-time, analyze, summarize, storage, Graph, Reports, legacy system, UDF, Exchange system, Surveillance System, Zooming level, query, latency, performance, Hadoop, Apache Kudu. Apache Impala, sql

LIST OF PUBLICATIONS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Data warehousing has become mandatory feature for lots of systems today. Data are collected from different data sources in order to provide meaningful business insight. Data warehouse are used to analyze and generate reports from gathered data via heterogeneous sources. Therefore, this has become the core of analytical systems.

Usually, composition of warehouse system is combination of technologies and components which provide secured connectivity for users. Including Banking, Airlines, Healthcare, Telecommunication, Insurance, Exchange Systems, Hospital industry and lots of systems are basically relay on data warehousing systems. Because it provides consistence information on various cross-functional activities. Users get capability to access critical data from many sources in one place. So, it saves time and reduce stress on production system.

Historical data warehousing and dynamic data warehousing are two different processes. Traditionally, Datawarehouse is refreshed periodically. But with high data ingestion rates, real-time data ingestion and analysis has become essential. Hence developing data warehousing system is not that much of easy task for critical data systems that feeding from exchange data. Because higher data rates analysis and streaming the data for report generation and graph point generation on different zooming levels really time and resource critical processes. So proper architectural design and technology basis implementation is very important for final platform of a warehouse.

Requirement on warehousing system varies from system to system. So, one warehouse solution in another system won't be full filling all needs of your system. Hence understanding each role of the warehousing system is very essential. Unless the final implementation would fail to cover business requirements.

By now lots of warehousing technologies are available. Load Manager is one main component in warehousing system, and it is responsible on loading data onto warehouse. Warehouse Manager ensure consistency of data while creating indexes and views for analyzing. Query Manager handle user queries for the warehouse. Moreover, there are end-user access tools for supporting data reporting, query, application developing etc. So, combining suitable technologies for achieving final goal is very critical.

By now Various technologies are available for each component implementation for a desired warehouse system. Hadoop Hive, Kudu, HDFS, spark, impala Big Query, OLAP, Oracle, Domo, CData Sync, Xplenty, SAP are some of trending warehousing technologies currently available.

So, under this project industrial level warehouse requirement has been addressed. The existing data warehouse system in the organization has been developed 12+ years ago. Meanwhile expecting business requirements are growing day by day. So, the technical feasibility and improvement level for existing native warehousing system has reached its maximum performances level.

Integrating new tools for data query via different open-source user applications and report generation has become limited. Moreover, requirement for graph points generation for different data analysis purposes has fallen into limited scope as this existing system is not supporting for other third-party applications directly. Due to the incompatibilities of existing system data storage format, indexing, data types, it must integrate additional intermediate processes to support such business requirements. Existing data summarization process is such component that reserved for generating graph points for data visualization for user applications.

Data summarization is one of main analytical task that is performing via modern data warehouse technologies. So, under this project, literature level evaluation for suitable warehousing model that can full fill the business and technical requirement has been addressed and experimental performance level evaluation for graph summarization point generation has been carried out while comparing existing graph points generation.

1.1 Motivation

London Stock Exchange Group (LSEG) plc is a British-based stock exchange and financial information company. As well as it is a global financial markets infrastructure business.

LSEG Technology in Sri Lanka develops and operates high-performance technology solutions, including trading, post-trade systems and market surveillance for over 40 organizations and exchanges, including this LSEG's own capital markets.

On a regular day LSEG Exchange System process around 7 million of orders and trades per day for around twenty-four thousands of instruments. Meanwhile another system called LSEG surveillance process these data and checks for market manipulation patterns. So, if it finds any

abnormalities, Surveillance system triggers alerts and indicates market manipulation scenarios for further investigation of the trade.
When we refer more details on Surveillance system, there is a dashboard to visualize these trading data for each instrument. So sometimes during rush hours, it could be several hundreds of orders and trades per second for an instrument.

Hence data summarization plays major role on visualizing these data into the dashboard. Currently this summarization process is done with use of separate process. As per its implementation, it calculates summery data for each zooming level of data visualization considering the time interval displaying in the dashboard and store them in a data warehouse.

So as per present system implementation, it uses separate process for data analysis and summarization. Then these pre summarized values for each summarization level are stored in existing data warehouse system. So, there is another process to handle frontend queries with this data warehouse. So, we understand, that the design of this existing platform outdated with respective to the responsibilities of today warehousing system.

As described in here, this is an industrial level problem which is practically needed a better solution for market data summarization. Summarization is a feature of a warehouse. So, depending on the way data is stored, performances levels for facilitating data query with summarization would differ. As per the problem here, it expects high performance level query output with lower latency for real-time data. This is a challenging task. Google, OLAP, etc. has found and established their specific solutions considering the requirements. Here we are addressing on market data, which are needed to be more accurate and requires more availability for the data accessing while ensure the security of data. Simply the requirement is different and unique.

As described in the introduction, there are lots of available technologies with different core features. So, any of technologies cannot be used without proper evaluation process. Due to this reason the company has needed proper evaluation process and details gathering for better model of data warehousing with required features.

Existing warehousing model has become a paining point for the total system as it adds up lots of unnecessary burden for maintenance. Since it is legacy and initial contributors for the system are now not even in the company, maintaining process has become bit complicated due to less documentations. So, every moment the system needs a simple upgrade or bug fix, it costs lots to the project. Users are developing different business rules and requesting changes in data types, ranges, data source schemas, indexes, and queries. As well as they request to integrate

new applications for the warehouse to generate reports, graphs etc. Hence currently it has identified that scope of data warehousing system is increasing continuously and resource expenditure on training and implementation is also growing in considerably amount.

Due to these reasons, proper warehousing solution implementation with newly emerged technologies has become one of next generation project in the organization.

So, storing historical data and real-time data are the main expectations of the system and high-performance level big data analysis capabilities has become the next main feature as per the business requirement of the warehousing system.

This summarization process needs to be handled by warehouse system itself rather than assigning this task for another process. When graph summarization is handling by another process, it has to stamp summarized data also into warehouse while increasing data amount of the system while producing data redundancy. Concept of warehousing is then violating. Data storges should be managed carefully as they are big data systems. Only 0^{th} level data is stored and on demand, request from user, data should be summarized send back to user for graph visualization.

1.2 Statement of the problem

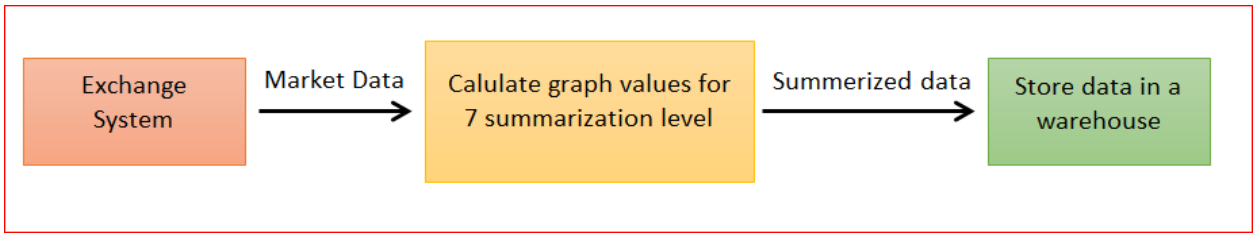


Figure 1: Data Flow

Existing warehousing technology used in this surveillance system has been developed by LSEG Technology Sri Lanka 12+ years ago. So, it has got capability to customize its process to achieve best performance level according to business requirement by then.

But maintaining these processes while keeping code quality standards requires extra effort. Because day by day trade counts are growing, hence scalability has become major requirement for the system. So continues performance level validation and non-functional testing require for

each release as per the standards follow by LSEG Technology Sri Lanka. This is an additional overhead for the project.

The algorithm for this market data summarization and visualization has been implemented concentrating only one type of data graph as per requirements by then. So, if it needs to integrate any other analytical model with different graph points – it has to implement another process to handover the responsibility. Moreover, the existing summarization model is bit complex and as per the internal design of the warehousing model, the designers by then had not anticipated exponential data growth like today. So, day by day, with respective to emerging new requirements expecting from warehouse system, current system is underperforming. Furthermore, as current implementation does not support advanced analytic capabilities, it has found it has to move with new technologies rather than staying on old ones as it got to compete with international business requirements.

So when we look at today's newly emerging technologies, LSEG Technology Surveillance team has found that there is an opportunity to move forward and gain better performance with more secured approaches which are available today. So now the team is looking into new emerging technologies and trying to validate these new technologies with project requirement.

Additional to that, as per new design requirements, organization has wanted to handle these data summarization process on request other than keeping pre calculated data in a warehouse. As the data are real-time and as it needs to maintain low latency, on request real-time data summarization has discovered as an open challenging task in the project.

1.3 Research Aims and Objectives

As per the publication on Data Warehousing and analytics infrastructure at Facebook, it claims that they have authored and contributed numbers of open-source technologies in order to address their requirements. Scribe, Hadoop and Hive have been mentioned as main technologies used for log collection, storage and analyzing the data. Additional to that, they have revealed how these systems have integrated together and enabled to implement a data warehouse that stores more than 15PB of data (2.5PB after compression) and loads more than 60TB of new data (10TB after compression) every day. (Ashish Thusoo, 2010)

As per the paper published by Google Cloud – The Future of Data warehousing – they have introduced Big Query which is an enterprise data warehousing technology, highly scalable, serverless and cost-effective. It mentions that it supports streaming data from IoT sensors, web,

social media, mobile apps and more to support on business delivery quickly and easily. Additional to that, it has mentioned BigQuery supports federated queries for processing data in non-native BigQuery storage systems without moving data and with increased efficiency. (Google, 2020) – More details are discussed under literature review.

So it is clear, LSEG Technology team has more opportunities to grow up their implementation for market data summarization for the dash board.

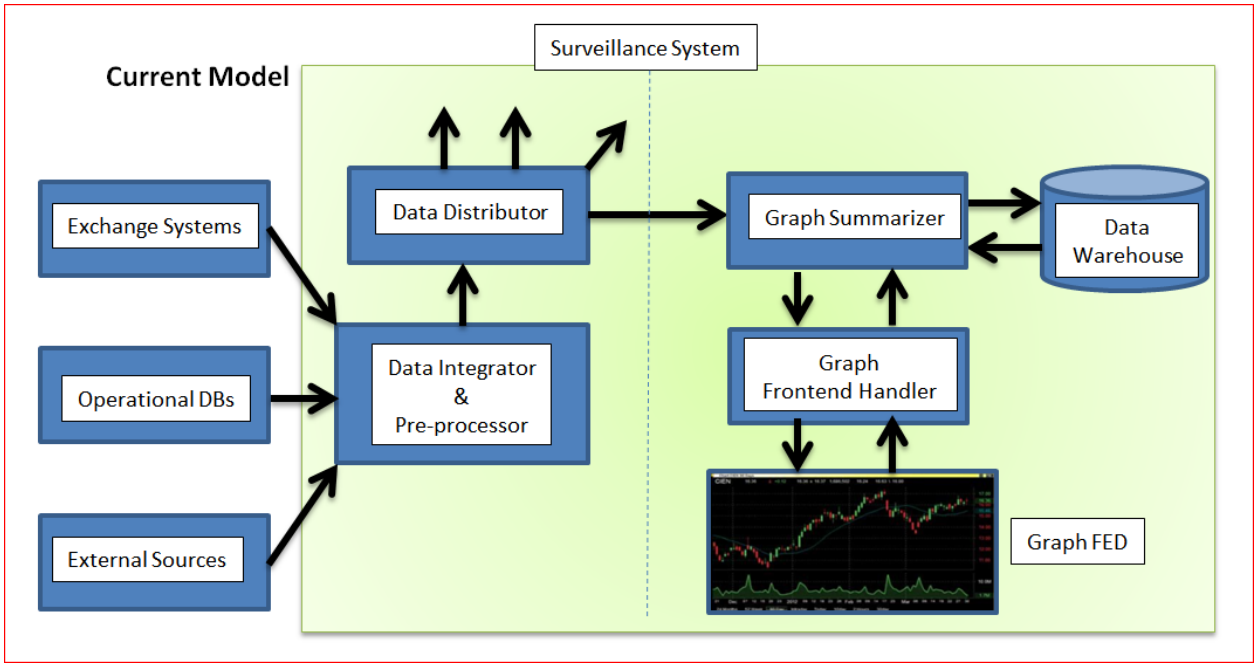


Figure 2: Current Datawarehouse and graph summarization model

- **Exchange Systems, Operational DBs, and External Sources**

Real-time data feeders for surveillance system.

- **Data Integrator & Pre-processor**

Integrate data from all data sources together and sequence them and downstream all these real-time data after some data enrichments.

- **Data Distributor**

Distribute data for different processors for usages.

- **Graph Summarizer**

Summarizes data based on time-based summarization logic and writes data to the Data Warehouse. Feeds real-time graph points to the Graph Frontend Handler.

- **Graph Frontend Handler**

Handles user requests on real time graphs. – postgresql are used as database management system.

- **Graph FED**

Frontend for Graph visualization

1.3.1 Aim

Data warehousing system is not a single component. It is composition of serval components and one single platform functioning together in order to satisfy business requirements.

Then choosing of correct solution for each component of the warehouse system should be done precisely considering the following main technical requirements and business requirements.

Technical requirements for new solution of warehousing system:

1. Lower latency support
2. Real-time data processing capability
3. Data Security on cloud
4. Fault Tolerance
5. High availability
6. Capability handle big data
7. Scalability
8. Maintainable

Business requirements for new solution of warehousing system:

1. Capability to integrate third-party applications for data visualization – Graphs
2. Capability to integrate third-party applications for report generation.
3. Capability to model different graphs for data analysis
4. Analytics on live data and recent data and historical data.
5. Correlation across data domains, even they are not traditionally stored together.
6. Low latency
7. One secure platform with above all features.

Major Expectations with new design implementation:

No separate component for Summarization process. It needs to be handled by data warehouse. So, Data warehouse should store all zero-level data set and on query from frontend handler

through Query Server, it should return summarized data set considering its summery level/zoom level.

So Here the research problem is to implement warehousing model that can support for summarizing real-time market data with usage of newly emerged technologies while achieving business requirements and technical requirements.

Since LSEG Technologies has met their current requirements with separate data summarization process and native data warehousing system – here it is not expecting to develop another process for each data analyzing and application requirements while discouraging capabilities of integrating new technologies for the system.

Hence main aim of this project is to refine the back born for this real-time market data summarization methodology with open-source technologies. So, research problem is based on how to select best technologies to achieve the necessities described here and implementing while validating each technology against full requirement details.

However, as here we have got the Input for the research, which is set of real-time market data and we have got clue about output which describes the final design approach as described above – display summarized set of market data on real-time basis. So here real research problem falls on development of process while merging available technologies for summarizing real-time market data in purpose of visualizing on market graph.

1.3.2 Objectives

- Literature level validation on selecting components for implementing warehouse platform that satisfy both technical and business requirements.
- Remove existing pre data summarization process from the system and assign data summarization responsibility on request for data warehousing system.
- Achieve high query performance with new warehouse platform than existing system – experimental level approach for validation.
- Implement real-time data summarization logic for plotting graphs and compare data summarization output of both existing and new system.

1.4 Scope

1.4.1 CRITICAL STUDY OF PROBLEM DOMAIN.

As described in the introduction, project problem domain is on implementing data warehousing solution for market data summarization. Market data are real-time and they arrive into the warehouse faster. So as soon as data received for the warehouse they should be available for data queries.

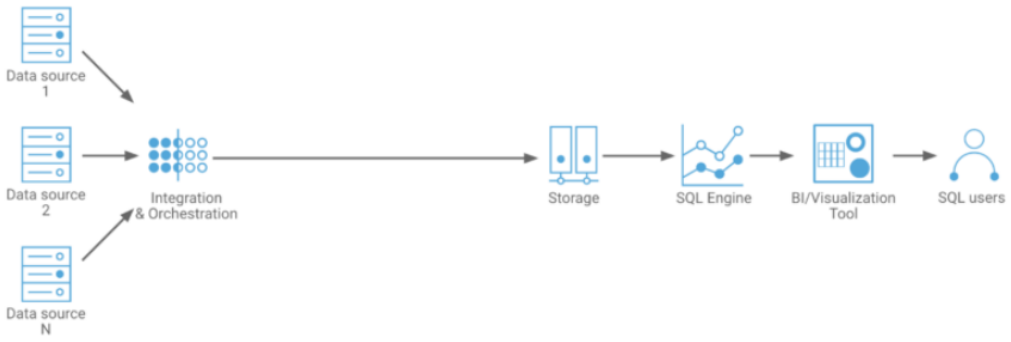


Figure 3: Standard architecture for a Real-Time Data Warehouse. (Hayes, 2020)

The existing warehousing model cannot adopt with varying requirements from business side and technical side. The system cannot support on advances analytical needs with higher performance level. As the data visualization is time critical, existing market data dashboard feature has been implemented with limited set of features. But with standard archtechture for a real-time data warehousing system, we can extend its capabilities satisfying both technical and business requirements.

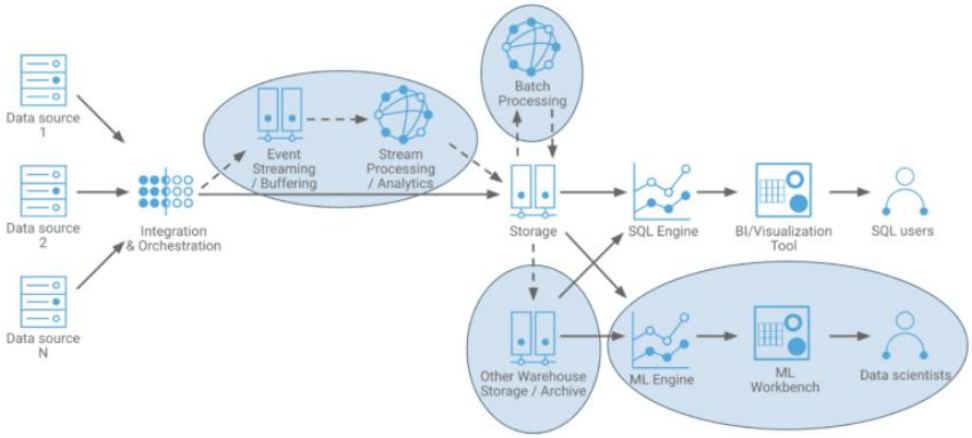


Figure 4: Standard architecture for a Real-Time Data Warehouse with extended capabilities. (Hayes, 2020)

Real-time data warehousing describes a system that reflects the state of the warehouse in real-time. Then if a query is run against the real-time data warehouse to monitor a particular facet about business or entity described by the warehouse, the response gives the state of that entity at the time query was executed. Most warehousing systems have data that are highly latent – or reflects the graph points at a point in the past. (Mukesh, 2009)

Dynamic data summarization is bit challenging task. Although there are lots of static data summarization tools, real-time data summarizing methodologies are limited. Google, Facebook, twitter are some major organizations that face same kind of problem on data storing, analyzing, and accessing. Because as per their record on Facebook, within 6 month of time period their data storage capacity requirement for compressed data has increased 5-10TB range. So, in order to address these challenges on scalability and diversity of data they have built their solution on technologies that support these characteristics at their core. They have used Hive and Hadoop as their core to storage and data processing strategies and Scribe is the core for their log collection strategy. (Dhruba Borthakur, 2010).

But here we cannot use their technology platform as it is. Because data structures, data types, data usages and requirements are much different than their platforms as described in the introduction. It is needed to validate each above-mentioned requirement against core capabilities of each technologies going to be used.

However, now it should be clear about the opportunity to grow up the existing system with open-source technologies to achieve the organization business requirements and the solution is about combination of technology platform.

1.4.2 IDENTIFICATION OF WAREHOUSING TECHNOLOGIES FOR EACH COMPONENTS.

Data warehouse is not a single component, as described it is consists of storage system and query engine. So identifying correct eco system is very important.

1.4.2.1 Comprehensive analysis for technology evaluations for data warehousing against required features for the new system

There are lots of newly emerged technologies that can be used for data warehousing. (Warehouse, n.d.)

1. Amazon Redshift

2. IBM Db2
3. Snowflake (Snowflake, n.d.)
4. Vertica
5. Google BigQuery (BigQuery, n.d.)

So, it must review their strengths and weaknesses against requirements of real-time warehousing system.

Performances on data summarization, query handling and analyzing are the main features expected here. Moreover, lower latency support with capability for optimizing database performances and real-time transaction processing with lower cost of pricing are other requirements.

However, most of above mentioned are not open source. So, selecting best warehousing tool for the project is bit challenging.

1.4.2.2 Comprehensive analysis for technology evaluations for query engine against required features for the new system.

Hive, Impala, HAWQ, IBM Big SQL, Drill, Tajo, Pig, Presto, DryadLINQ, Jaql are some top level big data query engines. (Chinnakali, 2015) But we cannot just combine them on solving our problem. Because each technology got their own strengths, weaknesses and their unique features.

Feature	Oracle	Hive	Impala
Query Language	SQL - full	SQL - subset	SQL - subset
Update individual record	Yes	No	No
Delete individual record	Yes	No	No
Transaction	Yes	No	No
Index support	Extensive	limited	No
Latency	High	low	medium
Data size	TB	PB	PB

Table 1: Comparison on Query Engine (Simplilearn, 2021)

So, each of them has to be validated against expected features from the query engine for this project implementation.

Mainly the query system should be able to support data summarization in an optimal manner with selected technology for data storage while achieving business requirements and technology requirements.

1.4.2.3 Study and select feasible data summarization technologies for real-time data processing

There are lots of tools for static data summarization. But real-time data summarization is bit challenging. Because it has to full fill low latency requirements and data availability requirements also.

So, it needs to justify best summarization methodology which can be iintegrated with query engine.

As per the article on Summarizing data written by WSO2 (WSO2, n.d.) – Enterprise Integrator Documentation, clock-time base summarization involves two steps.

1) Calculating the aggregations for the selected time granularities and storing the results.
2) Retrieving previously calculated aggregations for selected time granularities.

This method is not applicable as per the desired output design as this requires storing some calculated values on retrieving summarizing data.

Therefore under completion of this objective, it needs to discover and validate real-time data summarization technologies/methods which are able to be implemented in selected query engine with use of query language.

As mentioned, real-time data summarization methodologies are not frequent as static data summarization methods. So this objective is more challenging than other objectives.

1.4.2.4 Design evaluation against technologies selected.

As descried in the problem statement, expected design needs to be validated against selected technologies under 1.4.2.1 and 1.4.2.2. Because selected warehouse technology would not be compatible with selected query engine technology although they are the best solutions available as individually. So, this evaluation process is alco comes under the scope of this project.

So, it has to consider integrated system development when achieve the objective of the project. This approach is literature level task. So at the end the project, we have derived conceptual level integrated system which can full fill the requirement of real-time warehousing system with data summarization capabilities.

1.4.3 EVALUATE THE CONCEPTUAL SOLUTION PRACTICALLY

So obviously it needs to validate output of newly designed platform against existing data warehousing methodology and summarizing methodology in the organization.

Both systems need to be fed with same data source and need to check query time of each warehousing technologies. And needs to evaluate data summarization output deviations of new platform against existing platform.

1.5 Structure of the Thesis

Importance of data warehousing has been discussed under introduction for providing initial entrance to the readers. Real-time data warehousing and Traditional warehousing differs from lots of features. Here we are focusing on real-time warehousing systems. Market data handling for a warehousing system is critical task as it requires lots of resources in both hardware level and software level. As data receives, warehouse needs to store them and make available for users to access on real-time basis. Latency, accessibility, availability is major.

Data analytical capabilities are considered as another main feature in modern warehousing systems. Hence choosing appropriate warehousing storage technology and query engine has become decision that taken carefully.

LSEG is a global organization that provide platforms for exchange systems and surveillance systems. It is handling big data on the market. Surveillance system is basically the alarming system for market manipulation detection. So Data Analysis is the core responsibility of the system. For that data warehousing perform tremendous workload. Real-time data storing and make available for analytical purposes is resource consuming. So managing this two tasks is vital.

Current warehousing system consist of warehouse and separate process for data analysis purposes – but it is also not wide purposive. It only handles one algorithmic data summarization model. So, user expectations and technical expectations has limited into specific area.

Here in the thesis, it has discussed reasons for looking on another warehousing model for the organization while describing what is a warehouse, importance of warehouse, difference of real-

time warehousing systems and traditional warehousing systems. Because readers require basics for understanding the theories behind the rationale.

Storage technology used needs to compatible with query engine. So selecting the best combination on order to implement proper solution for this problem solving has been discussed under scope of the project. Main expectations on new warehousing model has been addressed while clarifying weaknesses and strengths of existing system.

As the outcome of thoroughly done literature reviews on data warehousing technologies – it has filtered out 3 of storage systems and 3 of data query systems. Then under approach of the final solution – it has discussed compatibilities of each storage and query systems and their main weaknesses and strengths.

Likewise, potential solution has been identified from Hadoop eco system – Kudu and Impala as storage and query system.

Then the thesis has revisited their strengths and has discussed how it can support to achieve final goal of the warehousing system. Since the suggesting system can't be low level solution, here it has done some practical tests on revealing performance levels in both system – existing on and newly suggesting one. This approach has declared that newly suggesting warehouse platform is capable of performing better.

As the next stage of the thesis, then it has developed online data summarization queries from new warehousing model. Evaluation for this data output has done only for graph points. Since native system graph points are pre calculated and stored – this algorithm is bit complex to implement via a single query. Therefore, it has developed several models for data summarization and validated their deviations against current summarized graph points.

CHAPTER 2
LITERATURE REVIEW

2.1 A Literature Review

2.1.1 Case Study - Data warehousing and analytics infrastructure at Facebook. *(Ashish Thusoo, 2010)*

We already know that many of Facebook's website features are based on large dataset analytics. So the analysts analyze data across the company and creating business intelligence dashboards in order to pattern mining. Then simple reporting apps like Facebook get capabilities to do more advanced type advertising such as friend recommendations. Their data requirement it is growing exponentially. But it has successfully managed their requirement inside the platform. As per the records, they have inspired a number of open-source technologies to meet these needs on Facebook. These include Scrub, Hadoop and Hive. Scrub has composed the Facebook's log collection, and Hive and Hadoop is the core technology for storage and analytics infrastructure.

As per reference paper (Ashish Thusoo, 2010) they have presented how these systems have come together and enabled them to implement a data warehouse that stores more than 15PB of data (2.5PB after compression) and loads more than 60TB of new data (10TB after compression) every day.

Additional to that they have discussed the motivations behind their design choices, the capabilities of this solution, the challenges that they face in day-to-day operations and future capabilities and improvements that they are working on.

2.1.2 Case Study - Google cloud – Introducing big Query *(Google, 2020)*

At Google, they have developed their serverless, highly scalable, and cost-effective cloud data. Warehouse BigQuery to address all these questions. It has brought agility, advanced analytics, data sharing and collaboration, security and governance, and data diversity.

As per the studies BigQuery is easy to set up and manage and does not require a database administrator. They further mention that any organization can quickly get up and running in seconds and start querying gigabytes to petabytes of data with standard SQL. It has powerful

security, governance, and reliability controls with a 99.9% uptime SLA and built-in protections like encryption by default, and fine-grained identity and access controls.

2.1.3 Customer case study for Big Query (Devoteam, 2021)

StubHub: Partnered with Google to transform their data infrastructure and empower teams to unlock the power of data.

Strategy

- Replaced legacy data warehousing infrastructure with BigQuery.
- Used BigQuery to establish a new data lake and set up an ETL framework.
- Migrated 60% of their data to BigQuery in less than a year.

Results

- Gained the ability to run complex queries in minutes, including ones that always used to time out.
- Created a single source of truth for all their data.
- Laid the foundation for using BigQuery ML to automate anomaly detection.

2.1.4 Case Study – OLAP *(WAREHOUSE, 2021)*

Customer: British American Tobacco Trading Company, a member of British American Tobacco Group, the world leader in the tobacco industry. Their target was to develop a consolidated data warehouse for storage and analysis of data uploaded from four different systems. The solution also needed to generate ad-hoc reports.

As the system had to provide online data processing and analysis, they have chosen Microsoft SQL Server 2008 Analysis Services (SSAS) OLAP technologies as the basis for the solution. All input data was moved to OLAP-cubes, and the multidimensional data warehouse was the only place where it could be stored and processed as per their requirement.

This allowed the system managers to analyze and measures in coming data from different systems (more than 150 measures) in different analytical slices (more than 40 dimensions and around 500 attributions).

2.1.5 Case Study - Google Mesa *(Ashish Gupta, 2014)*

Google runs an extensive advertising platform across multiple channels that serves billions of advertisements (or ads). So Google has present end-to-end design and implementation of a geo-

replicated, near real-time, scalable data warehousing system called Mesa. Mesa supports online queries and updates while providing strong consistency and transactional correctness guarantees. To capture and render content at high speeds, with low latency, they needed to do something that hadn't been done before.

Mesa has got capability to handle petabytes of data, processes millions of row updates per second, and serves billions of queries that fetch trillions of rows per day. Mesa is geo-replicated across multiple datacenters and supplies steady and repeatable query answers at low latency, even when an entire data center fails.

Technical Background study

So, let's study on some technologies mentioned above case studies and other related technologies trending by now.

2.1.6 OLAP *(queries, 2021)*

Online analytics Processing is a technique that can be used to analyze multidimensional data through queries. Additionally, it can be used as a business intelligence tool for data analysis, reporting, forecasting, and planning.

It performs analyzing by collecting data from multiple resources and storing in a Datawarehouse. Then it sorts the data and organize the data in different cubes and categorize data by dimensions.

There are several types of OLAP Types. ROLAP, MOLAP, HOLAP are some of them.

OLAP system advantages:

- Fast and efficient analysis operation in real time
- Forecasting with "What if" action
- Flexible self-service
- Single platform for all data operations
- Multidimensional data representation

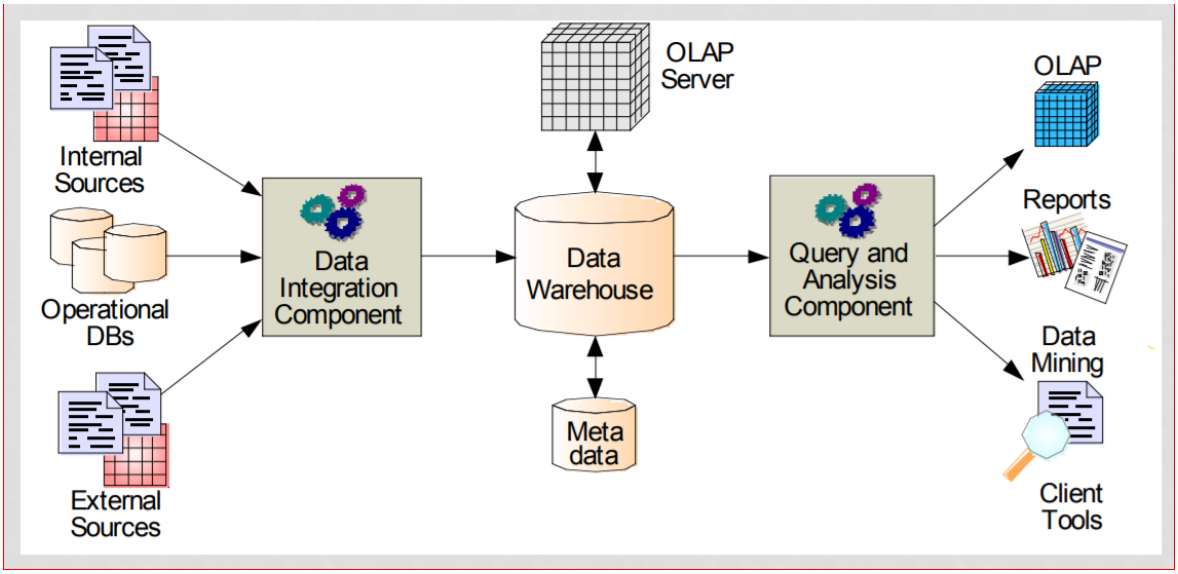


Figure 5: OLAP and Data Warehouse

Typically, OLAP queries are executed over a separate copy of the working data.

Usual OLAP queries need considerable time to be processed. Therefore, improve the execution time of each single query is vital. ParGRES is an open-source database cluster middleware which can be used for high performance OLAP query execution. By using intra-query parallelism on PC clusters, ParGRES has shown outstanding performance speed up using the TPC-H benchmark. So, it has recognized ParGRES as a very cost-effective solution for OLAP query execution in real-time data processing. (Mattoso, n.d.)

But due to following limitations of OLAP systems, still this also not a solution anymore.

- The traditional OLAP tools do not allow for the immediate analysis without pre-modeling.
- Although business personnel are the intended user of OLAP, they will still have to work with the IT pros because the traditional OLAP tools requires a complex modeling procedure and its users have to write a great number of codes/scripts/SQL.
- Poor computation capability - the traditional OLAP tools are of insufficient computational capabilities and few computational methods such as drilling, slicing, rotation, and simple column computation. This is because their architectures are old, lacking the innovation. (Collective, n.d.)

So, as per this literature level analysis – OLAP is not a potential solution for further investigation.

2.1.7 Hadoop Eco system *(Hadoop, n.d.)*

Apache Hadoop is a pool of open-source software tools that enables to handle network of computers to resolve problems relating huge amounts of data and calculations. It provides a software platform for distributed storage and handling of big data which using the MapReduce programming model.

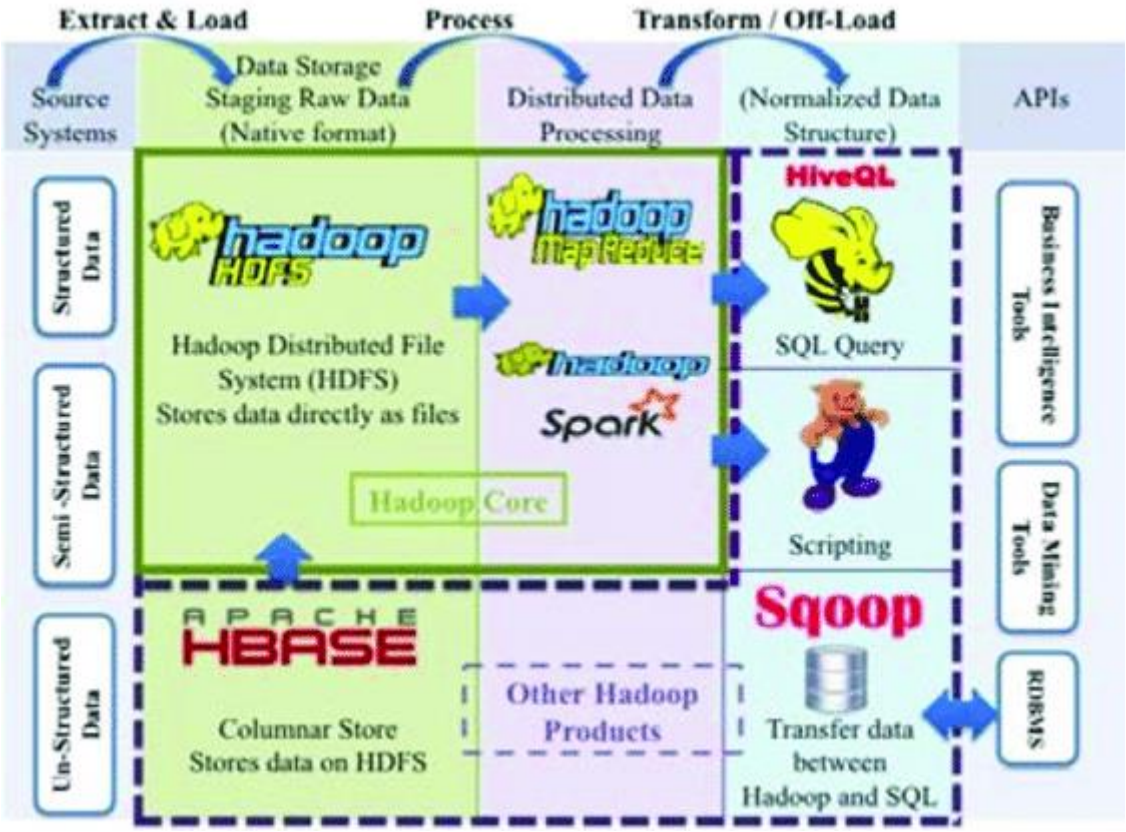


Figure 6: Apache Hadoop Ecosystem (Hsiao-Kang Lina, 2016)

Hadoop was initially implemented for computer clusters. Since then, it also found that is has been used on clusters of higher-end hardware.

The core of Apache Hadoop consists of:

- Storage - Allows organizations to store and analyze unlimited amounts and types of data
- Data processing, analyzing and serve - Quickly integrate with existing systems or applications to move data into and out of Hadoop through bulk load processing or streaming.

With Hadoop, analysts and data scientists have the flexibility to develop and iterate on advanced statistical models using a mix of partner technologies as well as open-source frameworks like Apache Spark

Hadoop breaks files into large blocks and distributes them across nodes in a cluster. Then it transfers codes into nodes to process the data in parallel. This approach got advantage of data locality, which nodes operate the data that they can access. This lets the dataset to be processed quicker and more efficiently than a more conventional supercomputer architecture. Additionally, followings can be listed as benefits of Hadoop platform.

1. Low-cost implementation
2. Scalable
3. Open-source software for reliable, scalable, distributed computing.
4. Running application on clusters
5. Data Management Provision
6. Data analysis
7. Process, big messy data sets for insights and answers.

So moreover, we can check available Hadoop storage systems and analytical tools for further analysis.

Top Hadoop Analytics Tools for 2021: (Analytics, 2021)

Apache Spark, MapReduce, Apache **Impala**, Apache **Hive**, Apache Mahout, Pig, HBase, Apache **Storm**, Tableau, R, Talend, Sqoop

Top Hadoop Storage systems (UpGrad, 2021)

HDFS, Mahout, GIS tools, Spark, **MapReduce**

2.1.8 HDFS

Hadoop Distributed File System, which is commonly known as HDFS is designed to store a large amount of data, hence is quite a lot more efficient. HDFS is used to carter large chunks of data quickly to applications. Yahoo has been using Hadoop Distributed File System to manage over 40 petabytes of data. (UpGrad, 2021) It provides a reliable means for managing pools of big data and supporting related big data analytics applications.

Features of HDFS can be listed like this: Data replication, Fault tolerance and reliability, High availability, Scalability, High throughput, Cost effectiveness, Large data set storage, Streaming data access.

2.1.9 Impala *(Overview, 2021)*

Impala is open-source software and it is a Massive Parallel Processing SQL query engine which can process huge volumes of data which is stored in Hadoop cluster. I is capable of providing high performance and low latency more than other SQL engines for Hadoop. Additional to that, Impala carries scalable parallel database technology for Hadoop, while giving capability for users to issue low-latency SQL queries to data stored in HDFS and Apache HBase without demanding data movement or transformation.

In other words, Impala is the main performing SQL engine (giving RDBMS-like experience) which provides the fastest way to access data that is stored in Hadoop Distributed File System.

2.1.10 MapReduce *(MapReduce, n.d.)*

Since it was presented by Google in 2004, MapReduce (MR) has been emerged as a popular framework for Big Data processing model in cluster environment and cloud computing. It has become a key of success for processing, analyzing, and managing large data sets with some number of implementations including the open-source Hadoop framework.

MR has many interesting qualities which are highly noticed in its design and simplicity in writing programs. It got only two functions, known as Map and Reduce, written by developer to process key-value data pairs.

Even though, MR is very simple to understand, it is hard to develop, optimize, and maintain its functionality especially in large-scale projects.

Moreover, MR identifies several limitations coming from its batch nature to handle real-time data.

Additional to that, as per records, it is inappropriate for many operations with multiple feedbacks like Join operation. So, the effort related to low-level programming of MR provides rise to high-level query languages (HLQL) based on MR.

Therefore, MapReduce is not potential candidate for the warehousing system implementation here as per our technical requirements.

2.1.11 Apache Storm *(Storm, 2021)*

Apache Storm is a distributed real-time computation system which is free and open source. Apache Storm helps to process limitless streams of data for real-time like Hadoop did for batch processing. Apache Storm can be used with any programming language and it is simple to use.

Apache Storm can be used for real-time analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. And it is really fast and it can process over a millions of tuples per second by a node. As well as, It is scalable, fault-tolerant, guarantees that data will be processed, and it is easy to set up and work with. Among many, Yahoo, Alibaba, Groupon, Twitter, Spotify uses Apache Storm.

Here the main limitation for our usage is that Storm performs Task-Parallel computations rather than data-parallel computation. Although Storm can provide better latency than spark streaming – development cost is much higher than streaming. (Hari Kumar, 2015)

Therefore, we have to keep out Apache Storm also from the final list of technology platforms.

2.1.12. Apache HIVE

Apache Hive is a java-based data warehousing tool designed by Facebook for analyzing and processing large data. And HIVE is data warehousing system built on to of Apache Hadoop. It is data warehousing solution for big data. Hive provides SQL-like interface and abstract query language (HiveQL) to query data stored in database and fill systems that are integrated with Hadoop. Hive Query Language (HIVEQL) supports similar SQL operations including joins, sub queries, Order By, Sort by etc. Hive tables consist of data and schema and they are divided for maximum flexibility. The CPU analysis depends on Hadoop configuration nodes and System configuration. (Sai Prasad Potharaju, 2014)

Hive support analysis of large datasets stored in the HDFS, with compatibility for file systems such as Amazon S3 and Alluxio. Hadoop use tool MapReduce and generate output for data graphs and reports. (Analytics, 2021)

Most of applications in the organization have been written in C++ and the market data storing are not file type – but relational. So, approach with HIVE is not possible for market data processing.

Figure 7:Apache HIVE usage

These each technology include their own core level features. So, filtering out each technology as per requirement belongs to the literature part. And integrating them together and evaluating belong to the implementation part of the project. But since there are lots of available technologies, each one must be evaluated carefully considering the whole requirement of the project. Because as mentioned, this is a industrial level requirement and here we are not targeting just to develop and implement a solution. It is required to justify the implementation model against their core capabilities considering use cases and evaluate them with production level dataset for the performance study.

2.1.13 Apache Spark

This is also an open-source unified analytical engine that can be used for big data and machine learning. And this has been developed speeding up Hadoop big data processing. Apache Spark enables batch, real-time, and advanced analytics over the Hadoop platform. Spark provides in-memory data processing for the developers and the data scientists. Companies, including Netflix, Yahoo, eBay, and many more, have deployed Spark at a massive scale.

Features of Apache Spark: Speed, Easy to use, Generality, Run everywhere.

2.2 Presentation of Scientific Material

As mentioned here I have used two approaches for validating the final platform of warehouse.

This is not a single technology platform – it needs to be the best combination of several technologies for solving the problem. Selecting best storage for the warehouse is not enough. Combining the most preferable query execution tool, metadata hander is a challenge. So, this initial validation is literature level approach with publicly available experimental outputs.

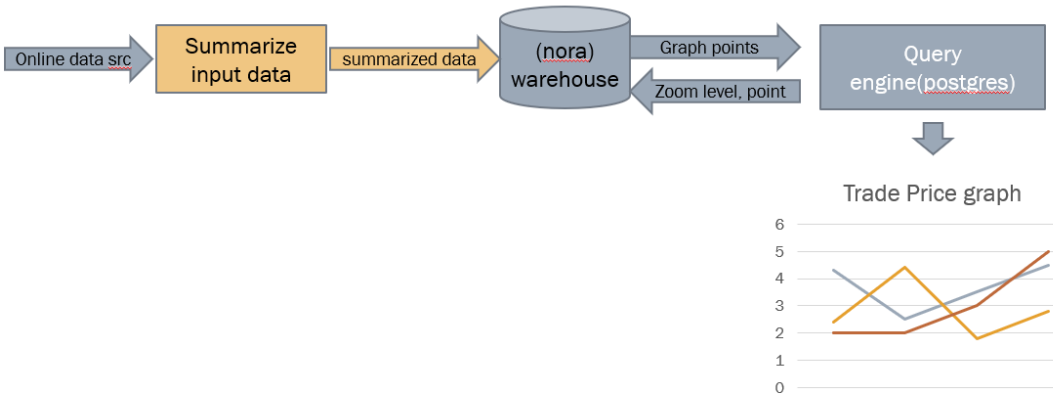


Figure 8: Current warehousing model

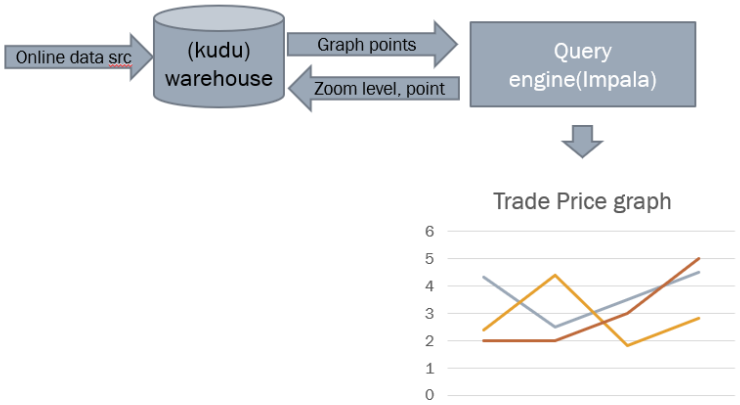


Figure 9: New System suggest

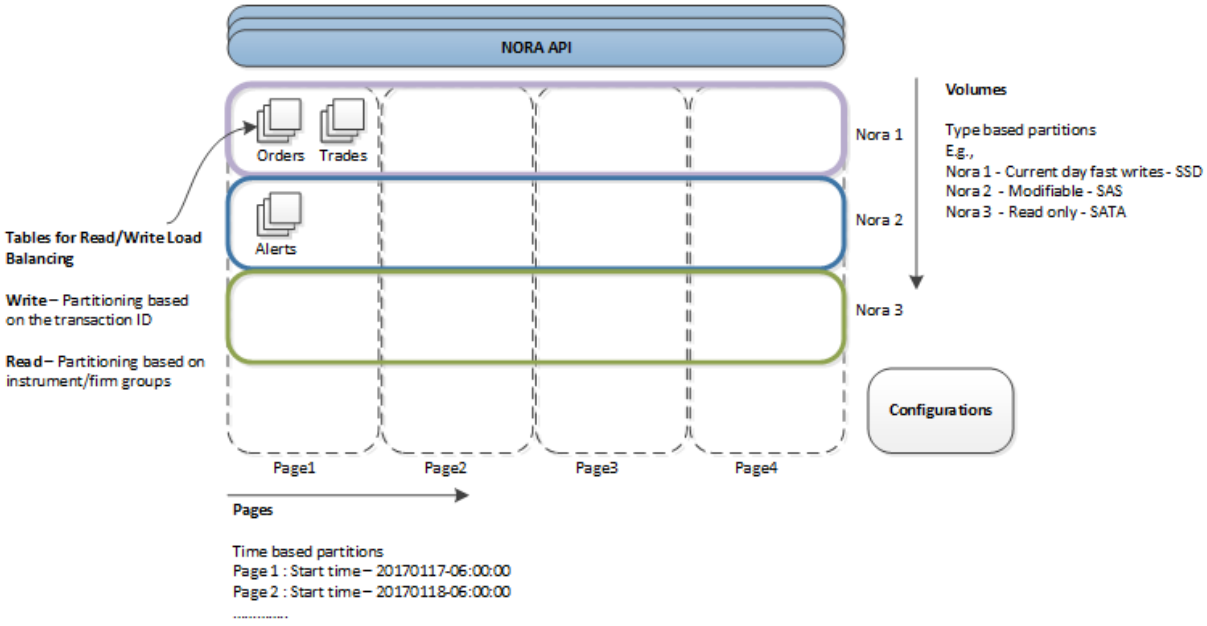


Figure 10: Existing Native warehouse platform

Key Features	Limitations
• Customizable in-house API - SQL support via NORA API • Flexibility of having customized data retrieval methods. • Performance - writing • Scalability control - i.e., data partitioning and distribution support (pages, volumes, and files - range and hash) • Tiered storage support • Backward compatibility support e.g., data transformations • Data replication support • Schemeless behavior - ability to store multiple data types in a single data store • Archive/backup/restore support	• Query return time slowness • Lack of analytical query support • Cluster dependency • Lack of security features such as column encryption • Insert rate limitation due to lack of real time compression and encoding • Cannot support updates • Weak data integrity checks • No transaction concepts • No commit/rollback support • Native message type dependency on meta data management • No concurrency control/atomicity – no table level locks • Cannot scale horizontally due to cluster dependency • Lots of random writes in index writing and hence index writing requires faster disks e.g., RAM

Table 2: Strengths and weaknesses of existing warehousing model

- Nora is a row oriented flat file storage with indexing support.
- Row oriented: data stored row by row.
- Index support: single layer and multi-layer indexing for column values of the rows

So now let's go through the evaluation of selected solutions for implementing data warehousing system.

Hadoop Eco system

One platform, multiple components.

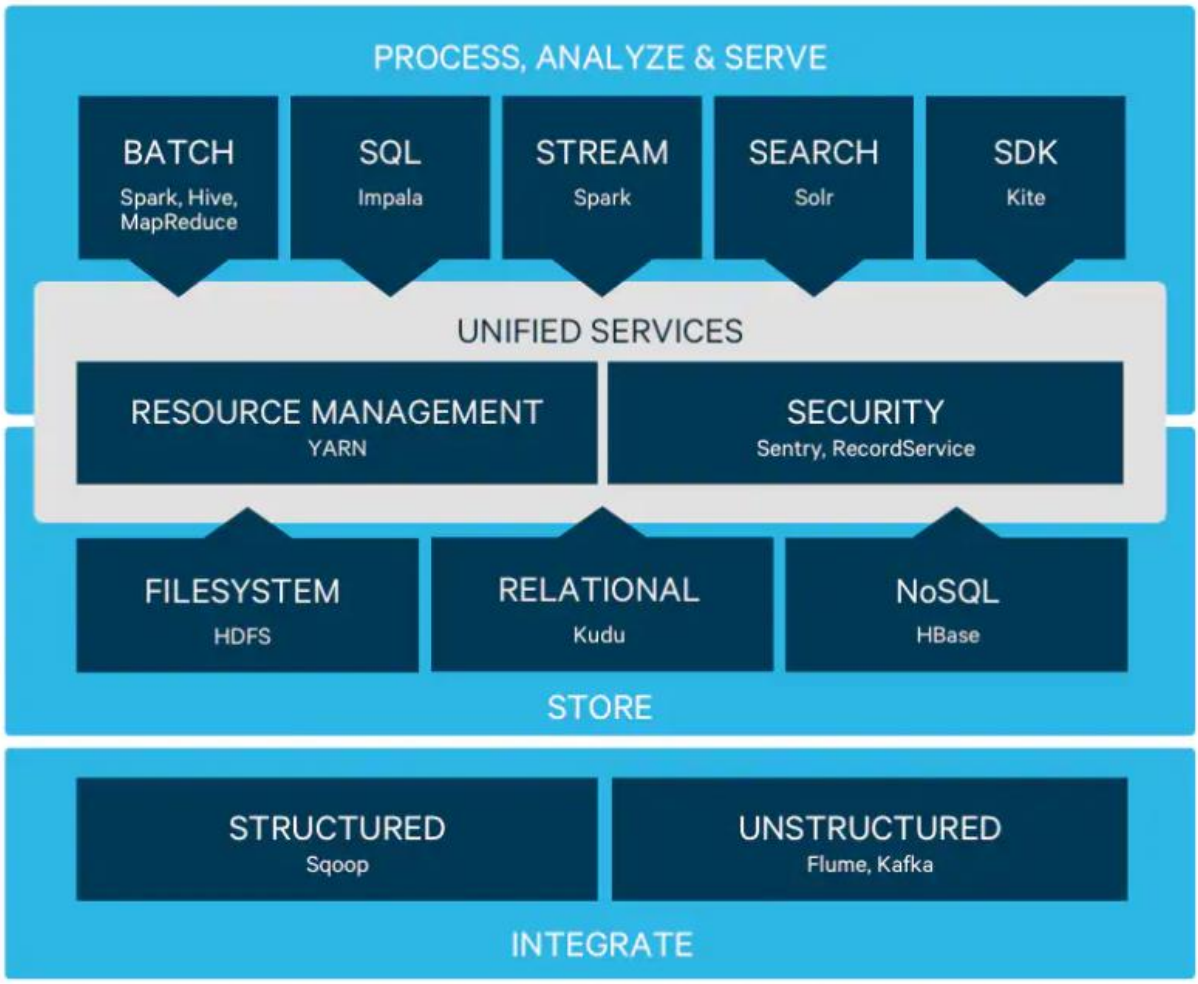


Figure 11: Apache Hadoop Eco System

1. **Data Store options:**

1.1.HDFS – File system

1.2.KUDU – Relational

1.3.Hbase – NoSQL

2. **Data Processing, Analyzing and Serving options:**

2.1.Spark, Hive, Pig, MapReduce – Batch

2.2.Spark – Stream

2.3.Apache Impala – SQL

2.4.Apache Solr – Search

2.5.Kite – Other

Targeting warehousing system needs to process Random access queries, support data streaming, support multiuser environments.

And the Analytical tool should have features like – scalable, faster access support, in memory data processing and security

Selecting right storage for relational data from Apache HDFS, Apache Hbase, Apache Kudu.

Requirement – Analytics in Hadoop

The warehousing system needs to support analytical capabilities

1.1.HDFS:

- Flexibility to store any type of data in any format.
- Infinite scalability for cost-effective active archival
- Highest throughput and storage density for analytics on static data set.

Cons:

- Limited/no ability for updates, deletes or streaming inserts.
- Good for traditional batch applications but not good for data streaming as data arrives.

1.2.Hbase:

- Flexibility to store any type of data with semi-structured schema. (But difficult to query with SQL)
- Real-Time Data Ingest and Serving
    - Built to handle fast changing data.
    - Scalable
    - So can serve big data requirements, but

Cons:

- Poor performance for analytic queries. (than HDFS)

1.3.Kudu

- Simple architecture for building real-time analytic application.
- Fast Analytics on Fast Data
    - Reporting with update support through Kudu and Impala

- Real-Time and streaming applications with Kudu+Spark

Kudu is not fast as HDFS storage efficiencies, and not good as Hbase random access.

But with Kudu we can build happy intermediate solution easier.

2.1.Spark

- Analytics engine for large scale data processing
- Can achieve high performance for both batch and streaming data – high speed.
- Offers over 80 high-level operators that make it easy to build parallel apps. – easy to use.
- Supports stack of libraries including data analytics.
- Can run on everywhere – on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources.

Apache spark is great, but not the perfect solution for here.

Cons:

- Apache Spark does not fit for a multi-user environment. It is not capable of handling more users concurrency.

2.2.Solr

- Advanced full-text search capabilities
- Optimized for high volume traffic
- Easy monitoring
- High scalable and fault tolerance
- Near real-time indexing
- Flexible and powerful query language allows to build complex queries
- High-speed query response

But there are some major limitations of using this for cloud-based systems.

Cons:

- Does not support authentication and authorization – so can be placed inside private network

2.3.Impala

- High performance SQL in Hadoop.
- Scalable
- Cost Effective
- Flexible
- Resilient to failure
- Support standard Hadoop components.

Cons:

- Impala does not support both serialization and deserialization.
- Impala can only read a text file. It does not support to read binary file user defined.
- Impala should update table whenever new record/file are added in the data directory of HDFS.

As per the organization technical and business requirements – above limitations of impala are not related as its main objective is to store market data and process.

We see that Impala can be integrated with storage managers – Kudu, HDFS or Hbase.

Therefore, Impala data analysis tool can be used on implementing the warehousing solution with above any of storage managers as we evaluated here.

Final Approach

Let's have detail study on shortlisted components – Data storage: Apache Kudu and analysis tool: Apache Impala.

1. Data Storge: Apache kudu

Rationale – Following main features of Apache Kudu supports for the storage.

- Scalable
    - Millions of read/write operations per second across cluster.
    - Multiple GB/second read throughput per node.
- Fast
    - Currently tested up to 300 nodes
    - Designed to scale 1000s of nodes and tens of PBs.
- Tabular

    - Represent data in structured tables like relational database– no clob/blob/files.
    - Individual record level access to 100+ billion row tables.
- Kudu tables has SQL-like schema and finite number of columns (not like Hbase/Cassendra)
- Support different data types – Bool, INT8, INT16, INT32, INT64, FLOAT, DOUBLE, STRING, BINARY, TIMESTAMP
- With subset of columns – can create composite primary key.
- Fast ALTER TABLE.
- Natively Kudu has different APIs– Java, Python, C++ and NoSQL-Style APIs
- Insert(), Update(), Delete(), Scan() functioning with low-milliseconds latencies.
- SQL via integrations with Impala and Spark.
- So great for online applications. But if expects to access with complex queries like join, group, aggregate we have to integrate with Impala or Spark.
- Support for many columns encoding and compression schemes.
    - Encoding – Delta, dictionary, bitshuffle.
    - Compression – LZ4, gzip, bzip2
- Kudu support flexible set of partitioning schemes
    - Partition by time range, hash or both
- Parallelizable scans.
- Scale-out storage system.

Partitioning schemes - By time range + series hash

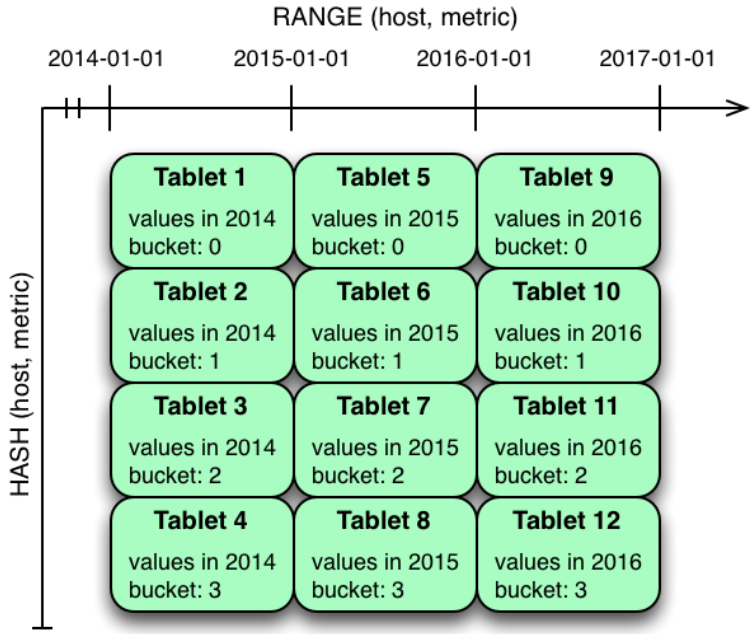


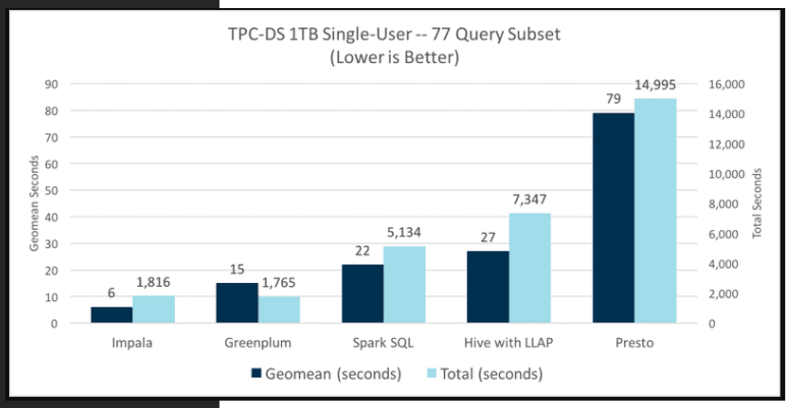
Figure 12: Apache Kudu Schema Design (Design, n.d.)

2. Data analysis tool: Apache Impala

Rationale: Impala is software from Cloudera, which is leading software for Massively Parallel Processing of SQL Query Engine, which runs natively on Apache Hadoop.

- Impala is modern sequel engine for Hadoop.
- Has implemented with idea of MPP – Massive Parallel Processing.
- Has designed for very good performances. – Has written from scratch c++
- Meta data stored in HIVE MetaStore
- Impala is open source and open standard
- Impala can access data using a query like SQL.
- Impala supports in-memory data processing. It is possible to analyze or access data stored on nodes in Hadoop without data transaction.
- can connect via ODBC/JDBC, Authenticate via Kerberos/LDAP, Authorization with GRANT/REVOKE.
- Impala can be integrated with Business Intelligence (BI) tools such as Tableau, Pentaho, Micro strategy, and Zoom data.
- Impala supports various file formats such as LZO, Sequence File, Avro, RCFile, and Parquet.

Query performance for single user



https://blog.cloudera.com/apache-impala-leads-traditional-analytic-database/

Figure 13: Impala Query performance for single user

Query performance for multiple user

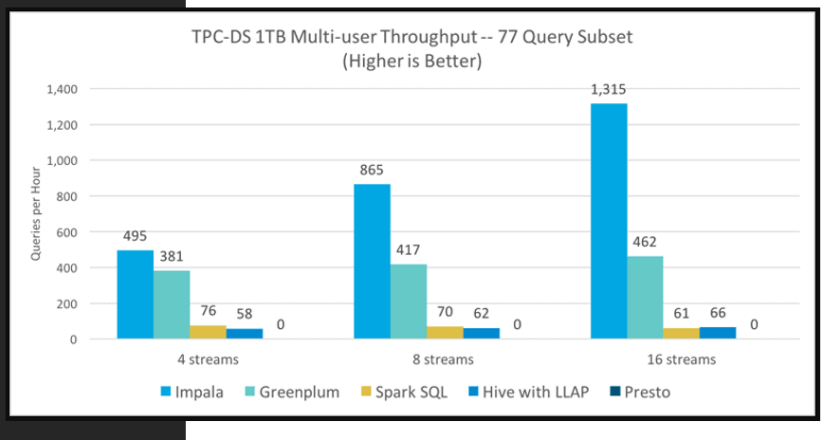


Figure 14: Impala Query performance for multiple users

Kudu and Impala together

Traditional database

Ex: Postgress

Figure 15: Kudu and Impala together

So as per our literature review – we have evaluated that Apache Impala and Apache Kudu as candidates for solving out warehousing model for real-time data processing. Now let's check how we can validate our hypothesis under methodology section.

Hypothesis1:
Together with Apache Kudu and Apache Impala can perform better than existing warehousing model on data query.

Hypothesis 2:

Together with Apache Kudu and Apache Impala can perform data summarization queries without no other external component.

CHAPTER 3

METHODOLOGY

As described in above section – Apache Kudu has been able to select as best solution for storage and Impala has been able to select as best option for query execution. Then Hive meta store becomes the metadata handler. So, this selection criteria are purely based on literature reviews and experiments has been done by others.

But validating our hypothesis is experimental approach.

Methodology of validating query performances of new warehousing model.

Initially, here it has been identified some production like queries as per the current system implementation. Then these each query has been validated against existing warehouse system and newly suggesting warehousing system. As these set of queries are already used by used for query data for Frontends, the performance outcome can be directly used for validating and compare two systems.

Special notes:

- Please note that here native system (Nora) and new system (Kudu) was set up with same set of data in the same instance.
- Although there are around 35 table spaces in original system – in order to perform testing only 3 tables have been migrated to Kudu storage (Execution report table, trade report table and instrument table)
- Queries have been performed individually and measured query return time.
    - For Nora :
        - time ~/postgres/bin/psql -q -t -c "select count(*) from instrument"
    - For impala
        - Query time returns with the query as follows in default.

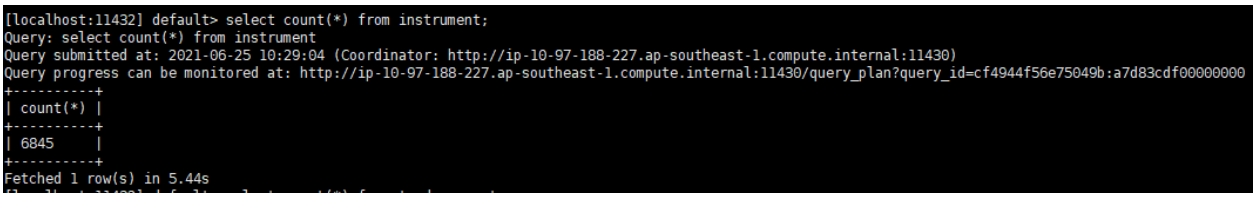


Figure 16: Impala shell - query time

- Queries ran on each warehouse systems as follows and due to data privacy reasons, their results won't be displaying.
- But as for proof of experiment following analitical queries output will be displayed under experiment 02.

Experiment 01:

Run same set of queries in two systems and check query return time.

Here this experiment has been performed on high data volume from production data – so only query time details only reported.

And please note that the queries have been selected considering existing system queries for the system.

Query No	PSQL query for Native warehouse	Impala Query for KUDU warehouse
1 – Trade query	SELECT instrument_id,buy_broker_id,sell_broker_id,buy_order_id,sell_order_id,exec_type, execution_type,executed_qty,executed_value,transact_time,time_sequence,trade_report_id, security_description,buy_participant_name,sell_participant_name,routing_seq, trade_report_link_id,aggressor_side,bargain_conditions,buy_account_type,buy_clearing_alpha, buy_client_order_id,buy_submittedtrader,buy_trader_id,cross_id,cross_type, dealing_capacity_buy, dealing_capacity_sell,delay_mode,exchange_transaction_id,execution_venue,firm_trade_id, intended_publish_time,isin,late_trade_indicator,match_status,novated_indicator, old_qty,old_value,only_for_market_data,order_book_id,original_exe_price, product_type,publish_indicator,reporting_participant,reporting_submitted_trader,segment_id, self_execution_flag,sell_account_type,sell_clearing_alpha,sell_client_order_id, sell_submittedtrader,sell_trader_id,settlement_currency,settlement_date,symbol, third_party_trade,trade_event_id,trade_report_parent_id,trade_report_ref_id, trade_reporting_model,trade_reported_time,trade_status,trade_sub_type,trad	SELECT instrument_id,buy_broker_id,sell_broker_id,buy_order_id,sell_order_id,exec_type, execution_type,executed_qty,executed_value,transact_time,time_sequence,trade_report_id, security_description,buy_participant_name,sell_participant_name,routing_seq, trade_report_link_id,aggressor_side,bargain_conditions,buy_account_type,buy_clearing_alpha, buy_client_order_id,buy_submittedtrader, buy_trader_id,cross_id,cross_type,dealing_capacity_buy, dealing_capacity_sell,delay_mode,exchange_transaction_id,execution_venue,firm_trade_id, intended_publish_time,isin,late_trade_indicator,match_status,novated_indicator, old_qty,old_value,only_for_market_data,order_book_id,original_exe_price, product_type,publish_indicator,reporting_participant,reporting_submitted_trader, segment_id, self_execution_flag,sell_account_type,sell_clearing_alpha,sell_client_order_id, sell_submittedtrader,sell_trader_id,settlement_currency,settlement_date,ImpReserv_symbol, third_party_trade,trade_event_id,trade_report_parent_id,trade_report_ref_id, trade_reporting_model,trade_reported_time,trade_status,trade_sub_type,trade_type, transaction_id,unique_id,exchange_routing_seq,is_carried_forward,agreed_time, waiver_indicator,pt_is_actx,pt_is_size,pt_is_ilqd,pt_is_lrgs,pt_is_rpri,pt_is_du

	e_type, transaction_id,unique_id,exchange_routing_seq,is_carried_forward,agreed_time, waiver_indicator,pt_is_actx,pt_is_size,pt_is_ilqd,pt_is_lrgs,pt_is_rpri,pt_is_dupl, pt_is_tpac,pt_is_xfph,pt_is_benc,pt_is_tncp,pt_is_sdiv,pt_is_pric,pt_is_orgn, price_notation,unit_quantity,notional_amount,notional_currency,buy_client_id, sell_client_id,buy_investor_code,sell_investor_code,buy_executing_trader_code, sell_executing_trader_code,buy_dea_flag,sell_dea_flag,buy_lp_flag,sell_lp_flag, buy_algo_flag,sell_algo_flag,buy_client_type,sell_client_type,buy_investor_type, sell_investor_type,buy_executing_trader_type,sell_executing_trader_type from trade_report where (instrument_id = 'RDSA') AND ((transact_time>='20190707-23:00:00.000') AND (transact_time<='20191006-22:59:59.999')) AND (_rownum_ < 100000);	pl, pt_is_tpac,pt_is_xfph,pt_is_benc,pt_is_tncp,pt_is_sdiv,pt_is_pric,pt_is_orgn, price_notation,unit_quantity,notional_amount,notional_currency,buy_client_id, sell_client_id,buy_investor_code,sell_investor_code,buy_executing_trader_code, sell_executing_trader_code,buy_dea_flag, sell_dea_flag,buy_lp_flag,sell_lp_flag, buy_algo_flag,sell_algo_flag,buy_client_type,sell_client_type,buy_investor_type, sell_investor_type,buy_executing_trader_type,sell_executing_trader_type from trade_report where (instrument_id = 'RDSA') AND ((transact_time>='20190707-23:00:00.000') AND (transact_time<='20191006-22:59:59.999')) order by routing_seq limit 100000;
2 – Trade History query	SELECT instrument_id,buy_broker_id,sell_broker_id,buy_order_id,sell_order_id,exec_type,execution_type, executed_qty,executed_value,transact_time,time_sequence,trade_report_id,security_description, buy_participant_name,sell_participant_name,routing_seq,aggressor_side,agreed_time, bargain_conditions,buy_account_type,buy_clearing_alpha,buy_client_order_id,buy_submittedtrader, buy_trader_id,cross_id,cross_type,dealing_capacity_buy,dealing_capacity_sell,delay_mode, exchange_transaction_id,execution_venue,firm_trade_id,intended_publish_time,isin, late_trade_indicator,match_status,novated_indicator,old_qty,old_value,only_for_market_data, order_book_id,original_exe_price,product_type,publish_indicator,reporting_participant, reporting_submitted_trader,segment_id,self_execution_flag,sell_account_type, sell_clearing_alpha,sell_client_order_id,sell_submittedtrader,sell_trader_id, settlement_currency,settlement_date,symbol,third_party_trade,trade_event_id, trade_report_link_id,trade_report_parent_id,trade_report_ref_id,trade_reporting_model, trade_reported_time,trade_status,trade_sub_type,trade_type,transaction_id,	SELECT instrument_id,buy_broker_id,sell_broker_id,buy_order_id,sell_order_id,exec_type, execution_type,executed_qty,executed_value, transact_time,time_sequence,trade_report_id,security_description,buy_participant_name,sell_participant_name,routing_seq, aggressor_side,agreed_time,bargain_conditions,buy_account_type,buy_clearing_alpha,buy_client_order_id,buy_submittedtrader,buy_trader_id, cross_id,cross_type,dealing_capacity_buy,dealing_capacity_sell,delay_mode,exchange_transaction_id,execution_venue,firm_trade_id, intended_publish_time,isin,late_trade_indicator,match_status,novated_indicator,old_qty,old_value,only_for_market_data,order_book_id, original_exe_price,product_type,publish_indicator,reporting_participant,reporting_submitted_trader,segment_id,self_execution_flag, sell_account_type,sell_clearing_alpha,sell_client_order_id,sell_submittedtrader, sell_trader_id,settlement_currency, settlement_date,ImpReserv_symbol,third_party_trade,trade_event_id,trade_report_link_id,trade_report_parent_id, trade_report_ref_id,trade_reporting_model,trade_reported_time,trade_status,trade_sub_type,trade_type,transaction_id, unique_id,exchange_routing_seq,waiver_indicator,is_carried_forward,pt_is_actx,pt_is_size,pt_is_ilqd,pt_is_lrgs,pt_is_rpri, pt_is_dupl,pt_is_tpac,pt_is_xfph,pt_is_benc,pt_is_tncp,pt_is_sdiv,pt_is_pric,pt_

	unique_id, exchange_routing_seq,waiver_indicator ,is_carried_forward,pt_is_actx,pt_is_ size, pt_is_ilqd,pt_is_lrgs,pt_is_rpri,pt_i s_dupl,pt_is_tpac,pt_is_xfph,pt_is_be nc, pt_is_tncp, pt_is_sdiv,pt_is_pric,pt_is_orgn,pric e_notation,unit_quantity, notional_amount,notional_currency,buy _client_id,sell_client_id,buy_investo r_code, sell_investor_code,buy_executing_trad er_code,sell_executing_trader_code,bu y_dea_flag, sell_dea_flag,buy_lp_flag,sell_lp_fla g, buy_algo_flag,sell_algo_flag,buy_clie nt_type, sell_client_type,buy_investor_type,se ll_investor_type, buy_executing_trader_type, sell_executing_trader_type from trade_report where trade_report_link_id='1XOQQLSGXF' AND transact_time>='20190731- 00:00:00.000000' AND transact_time<='20191031- 05:52:56.646208' AND instrument_id = 'RTO' AND (_rownum_ < 100000);	is_orgn,price_notation,unit_quantity, notional_amount,notional_currency,buy_cl ient_id,sell_client_id,buy_investor_code ,sell_investor_code,buy_executing_trader _code, sell_executing_trader_code,buy_dea_flag, sell_dea_flag,buy_lp_flag,sell_lp_flag,b uy_algo_flag,sell_algo_flag,buy_client_t ype, sell_client_type,buy_investor_type,sell_ investor_type, buy_executing_trader_type,sell_executing _trader_type from trade_report where trade_report_link_id='1XOQQLSGXF' AND transact_time>='20190731- 00:00:00.000000' AND transact_time<='20191031- 05:52:56.646208'AND instrument_id = 'RTO' order by routing_seq limit 100000;
3 – Order query	SELECT instrument_id,broker_id,trader_id,ord er_status,order_sub_type,transact_tim e,order_qty,value,order_id, exec_type,execution_type,side,securit y_description,participant_name,tif,en try_time,symbol, account_type,active_status,p_or_a_ind icator,capacity,clearing_alpha,client _id,client_order_id, container,contingent_condition,cross_ id,cross_type,cumulative_executed_siz e,date_of_expiry, exchange_transaction_id,execution_min _size,executed_qty,executed_value,hid den_size, inactive_time,isin,old_qty,old_value, only_for_market_data,order_book_id,or der_book_priority, order_consideration,order_reject_code ,order_seq,order_type,original_client _order_id, original_cross_id,original_visible_si ze,parent_order_id,passiveonlyorder,p ricedifferential, public_order_id,reason,time_sequence, routing_seq,segment_id,stop_price,sub mittedtrader, time_of_expiary,total_qty,trade_repor t_id,trade_report_link_id,trade_reque st_type, transaction_id,unique_id,visible_size ,exchange_routing_seq,waiver_indicato r,client_code, investment_decision_maker,executing_t rader_code,dea_flag,lp_flag,algo_flag ,party_client_type, investor_type,executing_trader_type,b bbo_setting,priority_time_stamp from execution_report where (instrument_id	SELECT instrument_id,broker_id,trader_id,order_ status,order_sub_type,transact_time, order_qty,ImpReserv_value,order_id,exec_ type,execution_type,side,security_descri ption, participant_name,tif,entry_time,ImpReser v_symbol,account_type,active_status,p_or _a_indicator, capacity,clearing_alpha,client_id,client _order_id,container,contingent_condition ,cross_id, cross_type,cumulative_executed_size,date _of_expiry,exchange_transaction_id, execution_min_size,executed_qty,executed _value,hidden_size,inactive_time, isin,old_qty,old_value,only_for_market_d ata,order_book_id,order_book_priority, order_consideration,order_reject_code,or der_seq,order_type,original_client_order _id, original_cross_id,original_visible_size, parent_order_id,passiveonlyorder,pricedi fferential, public_order_id,reason,time_sequence,rou ting_seq,segment_id,stop_price,submitted trader, time_of_expiary,total_qty,trade_report_i d,trade_report_link_id,trade_request_typ e,transaction_id, unique_id,visible_size,exchange_routing_ seq,waiver_indicator,client_code,investm ent_decision_maker, executing_trader_code,dea_flag,lp_flag,a lgo_flag,party_client_type,investor_type ,executing_trader_type,bbbo_setting,prio rity_time_stamp from execution_report where (instrument_id = 'RDSA') AND ((transact_time>='20190707- 23:00:00.000')AND

	= 'RDSA') AND ((transact_time>='20190707-23:00:00.000') AND (transact_time<='20191006-22:59:59.999')) AND (only_for_market_data != 1) AND (_rownum_ < 100000);	(transact_time<='20191006-22:59:59.999')) AND (only_for_market_data != 1) order by routing_seq limit 100000;
4 – Order History query	SELECT instrument_id,broker_id,trader_id,order_status,order_sub_type,transact_time,order_qty,value,order_id, exec_type,execution_type,side,security_description,participant_name,symbol,account_type,active_status, p_or_a_indicator,capacity,clearing_alpha,client_id,client_order_id,container,contingent_condition, cross_id,cross_type,cumulative_executed_size,date_of_expiry,entry_time,exchange_transaction_id, execution_min_size,executed_qty,executed_value,isin,old_qty,old_value,only_for_market_data, order_book_id,order_book_priority,order_consideration,order_reject_code,order_seq,order_type, original_client_order_id,original_cross_id,original_visible_size,parent_order_id,passiveonlyorder, pricedifferential,public_order_id,reason,time_sequence,routing_seq,segment_id,stop_price, submittedtrader,tif,time_of_expiary,total_qty,trade_report_id,trade_report_link_id, trade_request_type,transaction_id,unique_id,visible_size,exchange_routing_seq, waiver_indicator,client_code,investment_decision_maker,executing_trader_code, dea_flag,lp_flag,algo_flag,party_client_type,investor_type,executing_trader_type,bbbo_setting, priority_time_stamp from execution_report where order_id='00eujjkVjrZk' AND transact_time>='20190731-00:00:00.000000' AND transact_time<='20191031-06:29:28.118834' AND instrument_id = 'RDSA' AND (only_for_market_data != 1) AND (_rownum_ < 100000);	SELECT instrument_id,broker_id,trader_id,order_status,order_sub_type, transact_time,order_qty, ImpReserv_value,order_id,exec_type, execution_type,side,security_description,participant_name, ImpReserv_symbol,account_type,active_status,p_or_a_indicator, capacity,clearing_alpha,client_id,client_order_id,container, contingent_condition,cross_id,cross_type,cumulative_executed_size, date_of_expiry,entry_time,exchange_transaction_id,execution_min_size, executed_qty,executed_value,isin,old_qty,old_value,only_for_market_data, order_book_id,order_book_priority,order_consideration,order_reject_code,order_seq, order_type,original_client_order_id,original_cross_id,original_visible_size,parent_order_id, passiveonlyorder,pricedifferential,public_order_id,reason,time_sequence,routing_seq,segment_id, stop_price,submittedtrader,tif,time_of_expiary,total_qty,trade_report_id,trade_report_link_id, trade_request_type,transaction_id,unique_id,visible_size,exchange_routing_seq, waiver_indicator, client_code,investment_decision_maker,executing_trader_code,dea_flag,lp_flag, algo_flag,party_client_type,investor_type,executing_trader_type,bbbo_setting,priority_time_stamp from execution_report where order_id='00fFcCKBVkXl' AND transact_time>='20190731-00:00:00.000000' AND transact_time<='20191031-06:29:28.118834' AND instrument_id = 'RDSA' AND (only_for_market_data != 1) order by routing_seq limit 100000;

Table 3: Experiment1 - Queries

Query	Time – Existing system	Time – proposing model for standalone mode.
1	1	0.842
2	0.84	0.690
3	6.198	2.13
4	1.3	0.65

Table 4: Experiment1 - Query Performance Time

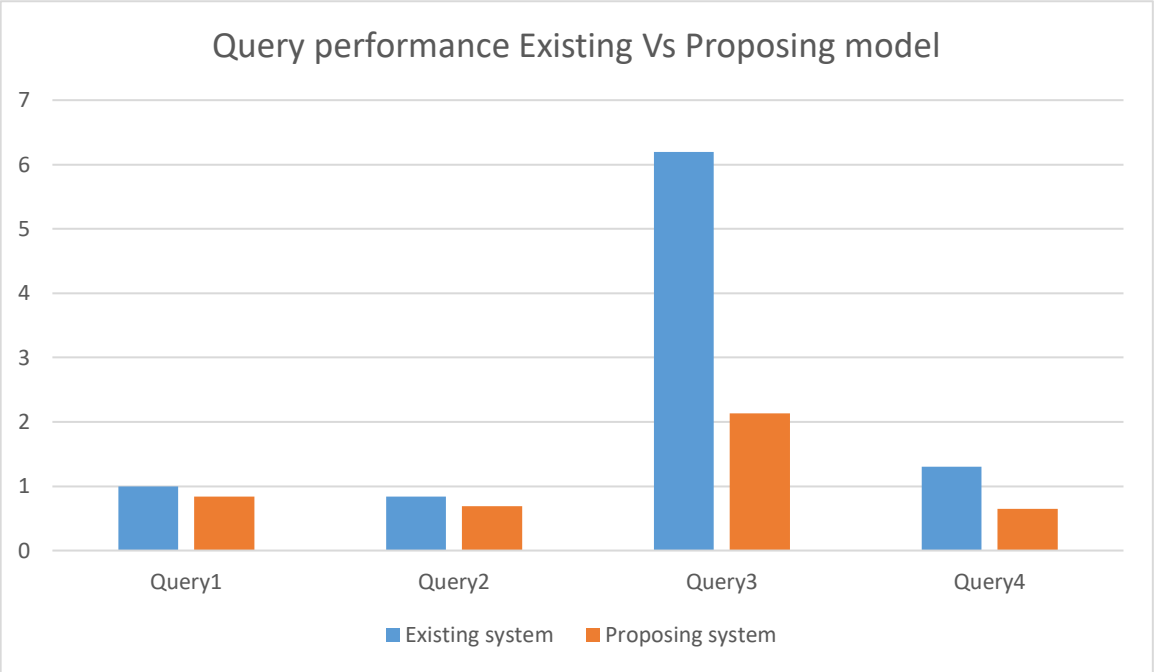


Figure 17: Experiment 1 - Query Performance time Graph

Results: Proposing system show better query performance than existing system.

Experiment 02:

Perform some analytical queries on both systems. And compare their query execution time.

Query No	PSQL query for Native warehouse	Impala query for KUDU warehouse
01 - Get average executed quantity of orders per instrument	`select avg(executed_qty) as avg_ex from execution_report group by instrument_id order by avg_ex desc;`	
02 - Sum of executed quantity per broker and per instrument	`select instrument_id, broker_id, sum(executed_qty) as sum_trade_vol from execution_report group by instrument_id, broker_id order by sum_trade_vol desc limit 10;`	
03 - Joining execution report and trade report based on transaction_id	`select e.instrument_id, order_id, e.trade_report_link_id, trade_reported_time, currency_conv_indicator, trade_consideration, t.waiver_indicator from execution_report e inner join trade_report t on e.transaction_id = t.transaction_id limit 100;`	

Table 5: Experiment 2 - Analytical Queries

Query	Time – Existing system (s)	Time – proposing model for standalone mode. (s)
1	5.634	0.27
2	0.655	0.23
3	7.28	0.41

Table 6: Experiment 2 - Analytical query time

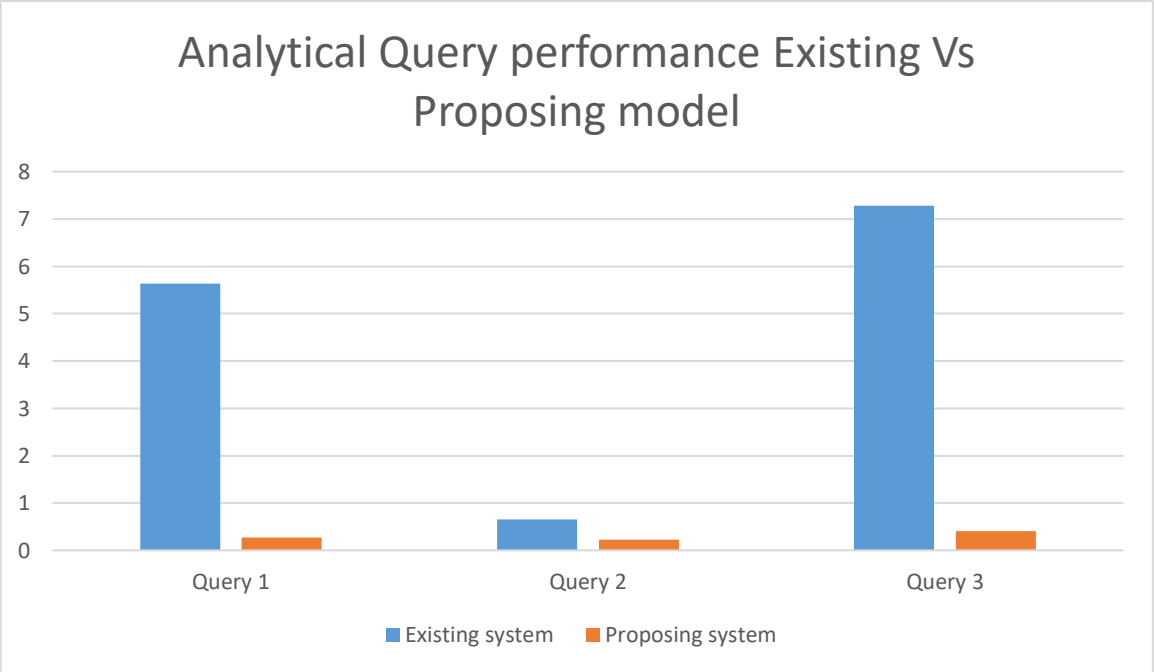


Figure 18: Experiment 2- Analytical Query time graph

Query 01

Kudu sample query output

```
[localhost:11432] default> select avg(executed_qty) as avg_ex from execution_report group by instrument_id order by avg_ex desc;
Query: select avg(executed_qty) as avg_ex from execution_report group by instrument_id order by avg_ex desc
Query submitted at: 2021-06-25 12:22:00 (Coordinator: http://ip-10-97-188-227.ap-southeast-1.compute.internal:11430)
Query progress can be monitored at: http://ip-10-97-188-227.ap-southeast-1.compute.internal:11430/query_plan?query_id=33469db4b7091dc3:c52b052500000000
+------------------+
| avg_ex           |
+------------------+
| 625              |
| 183.7837837837838 |
| 133.3333333333333 |
| 117.6470588235294 |
| 0                |
| 0                |
| 0                |
| 0                |
+------------------+
Fetched 8 row(s) in 0.27s
[localhost:11432] default>
```

Figure 19: Experiment 2 - Query 1- Impala query Output

Native system query output:

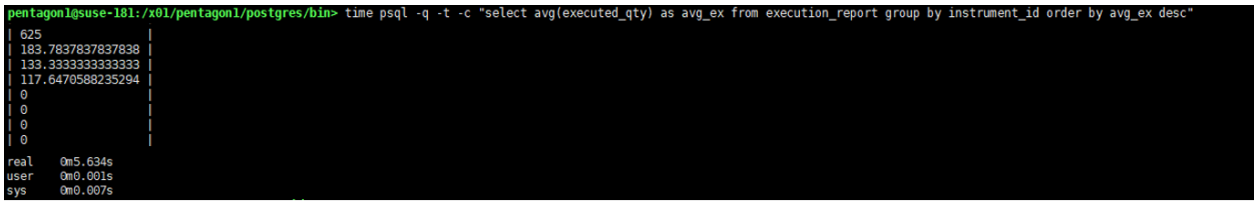


Figure 20: Experiment 2 - Query 1- Postgres query Output

Query 02

Kudu sample query output:

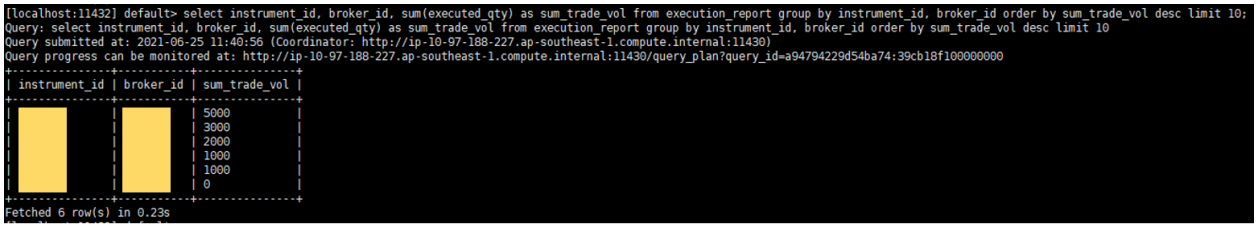


Figure 21: Experiment 2 - Query 2- Impala query Output

Native system query output:

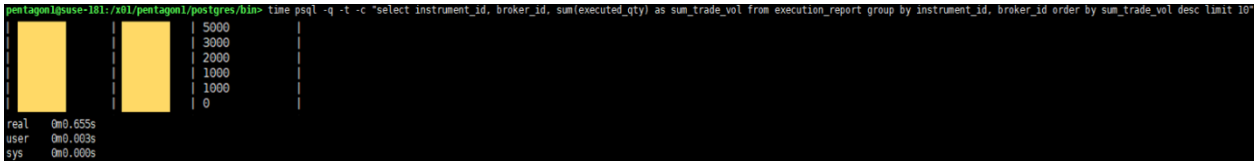


Figure 22: Experiment 2 - Query 2- Postgres query Output

Query 03

Kudu sample query output:

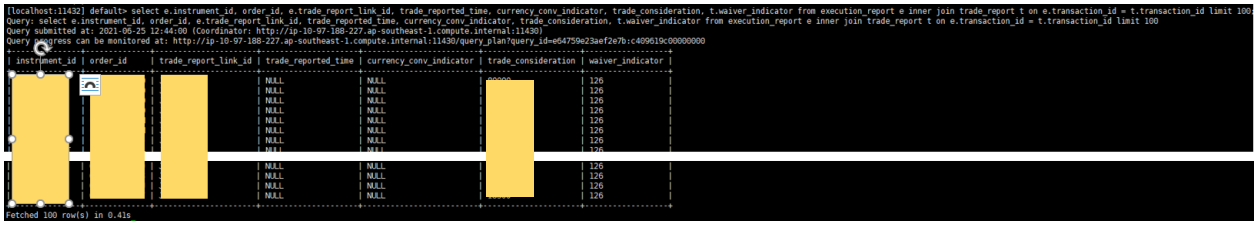


Figure 23: Experiment 2 - Query 3- Impala query Output

Native system query output:

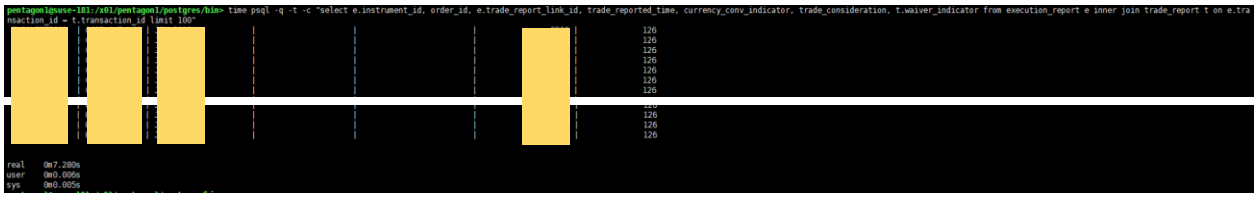


Figure 24: Experiment 2 - Query 3- Postgres query Output

Experiment 03

Now need to build data summarization query which support maximum performance level and low error rate compared to existing data summarized values.

In existing system there is a table for storing pre calculated summarization data points. So Postgres queries just got to collect data from table. But new proposing system needs to perform online data summarizations also.

Here at this point we have concluded the data query performance level is better in new proposing system. So, under this experiment it is expected to validate query output for data summarization.

Level zero original data points

```
SELECT time_sequence,y_pos from graph_points WHERE (time_sequence >= 1603947279832890000 AND time_sequence <= 1603956063602808000 ) AND filterfieldvalue='IRR04' AND level = 0 AND data_partition_id = 551;
```

time_sequence	y_pos
1603947279832890000	120
1603947281149609000	123
1603947281909799000	126
1603947283073676000	129
1603947283961332000	132
1603947285059472000	135
1603947286034132000	138
1603947287060625000	141
1603947288102194000	144
1603947289188403000	147
1603947290511048000	150

1603947291246374000	153
1603947292276991000	156
1603947293311413000	159
1603947294438627000	162
1603947295419112000	165
1603947296482587000	168
1603947297844352000	171
1603947298596609000	174
1603955991782818000	174
1603956059470845000	220
1603956060498245000	223
1603956061536288000	226
1603956062566090000	229
1603956063602808000	232
+--------------------+-----------+

For zoom level 5000 real output from Native system,

```
SELECT time_sequence,y_pos from graph_points WHERE (time_sequence >= 1603947279832890000 AND time_sequence <= 1603956063602808000 ) AND filterfieldvalue='IRR04' AND level = 5000 AND data_partition_id = 551;
```

time_sequence	y_pos
1603947279832890000	120
1603947286034132000	138
1603947290511048000	150
1603947295419112000	165
1603947298596609000	174
1603955991782818000	174
1603956060498245000	223

Suggested query 01:

```
WITH temptable AS (SELECT time_sequence, FLOOR ((time_sequence - MIN(time_sequence) OVER (PARTITION BY level))/5000000000) AS g, y_pos FROM graph_points WHERE level=0 AND filterfieldvalue='IRR04' AND data_partition_id=551 AND level=0 AND (time_sequence >= 1603947279832890000 AND time_sequence <= 1603956063602808000)),

temptable2 AS (SELECT MAX(y_pos) OVER (PARTITION BY g) AS y_pos_selected, LAST_VALUE(time_sequence) OVER (PARTITION BY y_pos,g) AS time_sequence_selected,y_pos FROM temptable)
```

```
SELECT  y_pos_selected,time_sequence_selected  FROM  temptable2
WHERE y_pos = y_pos_selected;
```

y_pos_selected	time_sequence_selected
132	1603947283961332000
147	1603947289188403000
162	1603947294438627000
174	1603947298596609000
174	1603955991782818000
220	1603956059470845000
232	1603956063602808000

MAX(y_pos) aggregate function should be replaced by a suitable aggregation function for summarization.

Original level 0 graph:

Figure 25: Level zero original data points - Graph

Existing summarized graph

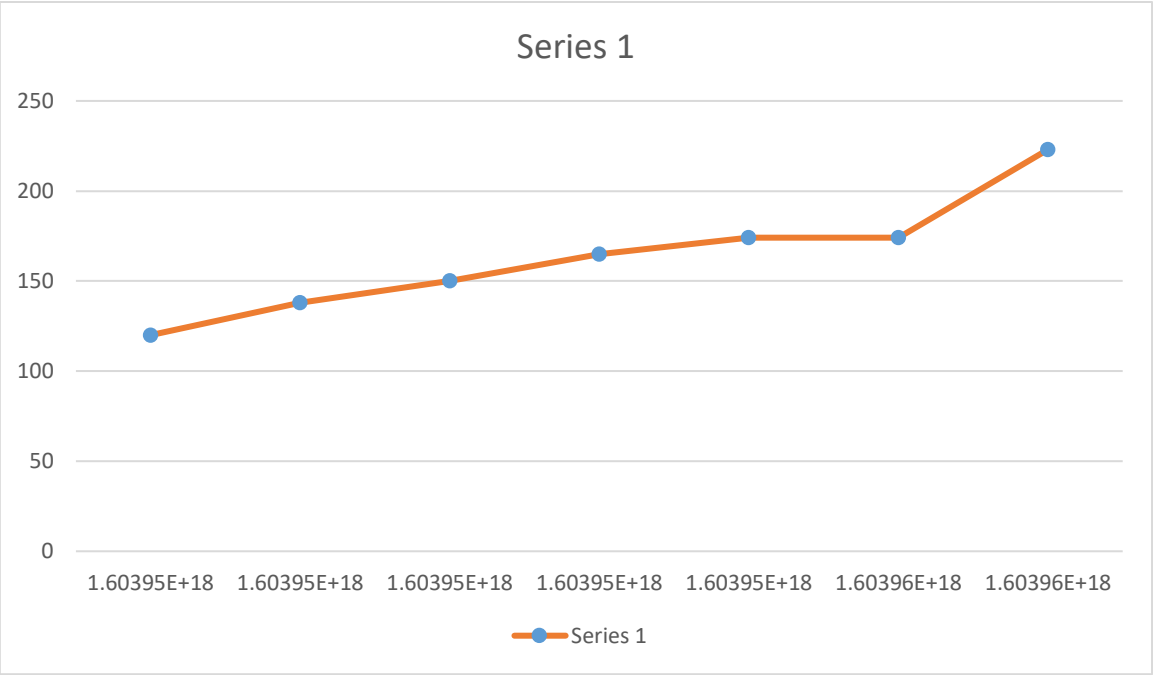


Figure 26 : Summarization Graph for zoom level 5000 real output from Native system

Newly suggesting summarization graph

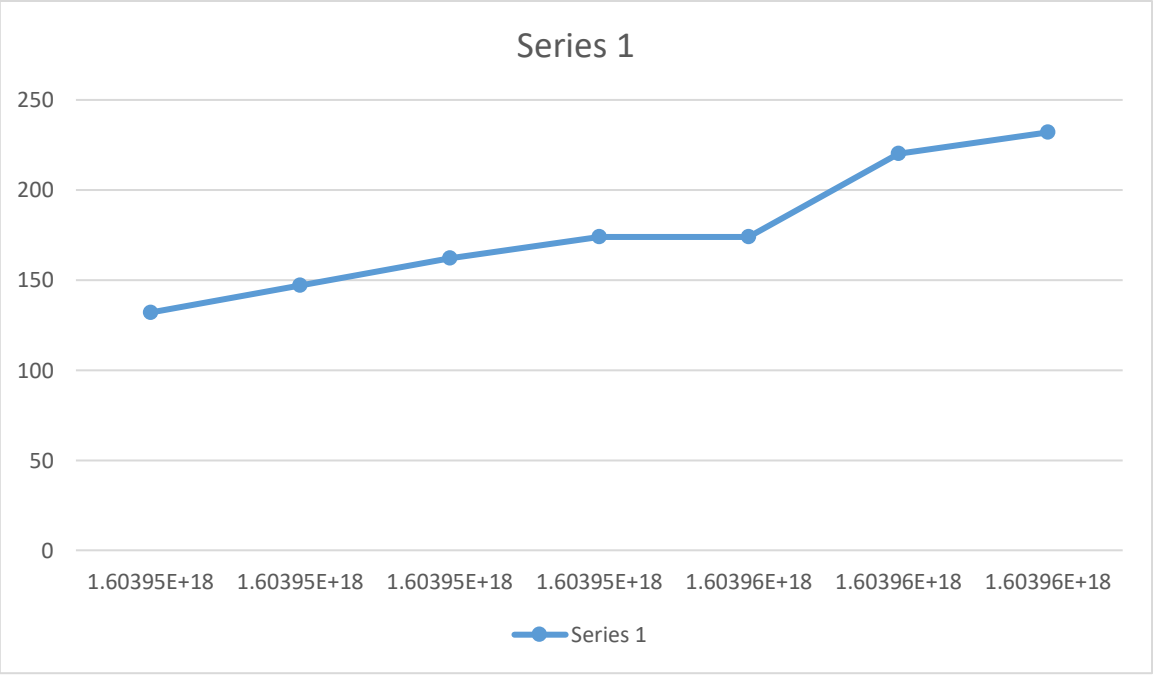


Figure 27: Summarization Graph for zoom level 5000 output from New system

CHAPTER 4

EVALUATION AND RESULTS

Real-time data warehousing system suggested has been evaluated via set of actions as follows.

1. Identify weaknesses of existing warehousing model and discuss requirements for new warehousing model for the surveillance system. – Chapter1
2. Identify expected technical and business requirement for new warehousing model – Chapter1
3. List down components for warehousing system implementation – storage system and query system. – Chapter2
4. List down appropriate technologies for each component considering both business requirements and technical requirements– Chapter2
5. Shortlist compatible technologies for buildup total warehousing system considering their features and compatibilities for final system. – Impala and Kudu technologies were finalized for hypothesis development. – Chapter2
    a. Hypothesis1:

       Together with Apache Kudu and Apache Impala can perform better than existing warehousing model on data query.
    b. Hypothesis2:

       Together with Apache Kudu and Apache Impala can perform data summarization queries without no other external component.
6. Experiment on validating each Hypothesis – Methodology
    a. Hypothesis1:
        i. Experiment 1 has concluded that Query performance is better in proposing system around 37.32% for direct queries.

Query	Time(s) – Existing system	Time(s) – proposing model for standalone mode.	Query performance enhancement Px
1	1	0.842	$P1 = \frac{1 - 0.842}{1} \times 100\% = 15.8\%$
2	0.84	0.690	$P2 = \frac{0.84 - 0.690}{0.84} \times 100\% = 17.86\%$
3	6.198	2.13	$P3 = \frac{6.198 - 2.13}{6.198} \times 100\% = 65.63\%$
4	1.3	0.65	$P4 = \frac{1.3 - 0.65}{1.3} \times 100\% = 50\%$
Average enhancement of query return time			$\frac{P1 + P2 + P3 + P4}{4} = 37.32\%$

Table 7: Experiment 1 – Query enhancement analysis

ii. Experiment 2 has concluded that Query performance is better in proposing system around 84.82% for analytical queries.

Query	Time – Existing system (s)	Time – proposing model for standalone mode. (s)	Query performance enhancement Px
1	5.634	0.27	$P1 = \frac{5.634 - 0.27}{5.634} \times 100\% = 95.20\%$
2	0.655	0.23	$P2 = \frac{0.655 - 0.23}{0.655} \times 100\% = 64.88\%$

3	7.28	0.41	$P3 = \frac{7.28 - 0.41}{7.28} \times 100\% = 94.37\%$
Average enhancement of query return time			$\frac{P1 + P2 + P3}{4} = 84.82\%$

Table 8:Experiment 1 – Analysis Query enhancement analysis

As per above experimental results – it clearly concludes that new system perform better. When it comes to analytical queries, new proposing system has been able to record 84.82% of query enhancement. This is a real big number that can make positive trigger point to look at new system as an analytical system rather than integrating or reserving another process for that.

b. Hypothesis 2:

i. Experiment 3 has concluded that Query output for summarized data output has deviation of SS% when compared to native system data summarization.

From 26 graph points:

After summarization as per existing model – 7 points has been extracted for visualization.

After summarization as per suggested query – 7 points has been extracted for visualization.

But only 2 points has overlapped. – means expected similarity between two graph/summarization model is around 28%. This value is not as expected due to following reasons:

- Data summarization algorithm in existing system is a complex model.
- Demonstrated summarization model is a simple analytical query selecting highest value for each granular time interval – mans two systems has demonstrated outputs from two different algorithms.

But visually they have created 28% similar outputs as summarization outputs. We have more potential for enhancing the summarization logic for reach out existing summarization output. Since the objective is to conclude that with new model, we can analyze data and generate reports and graphs has been satisfied.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Moving into new warehousing model is not a simple task for any kind of live system. Because it includes stream of proper validation procedures and migration tasks. But as described, existing warehousing system in the organization has reached out its maximum performance level and requirement level. Facebook, Google, and many more systems maintain their own warehousing models and has introduced fascinating warehousing technologies. But Market data warehousing system behaviors and expectations are much different than them. So, adopting to such platforms is not feasible here.

Apache Hadoop is echo system consisting of various solutions for big data storing and analyzing. As a result of critical literature review, Apache Kudu and Apache Impala have been identified as candidates for new warehouse implementation. Analyzing on their strengths and weaknesses helped to reach out this conclusion and hypothesis development. But justification has been conducted in experimental level. Both hypotheses have been concluded successfully. Therefore, new warehousing model technologies have been proofed as better solution than existing warehousing model.

In hypothesis2: although we have concluded that new system is capable of generating analytical queries and generate summarization output for given set of data – its output has not exactly matched for pre summarized data set in existing model. It is obvious both systems have used two different algorithms hence two different outputs. So as future work: it can enhance/implement more realistic summarization model matching to existing summarization algorithm. As per appendices – In order to reach out exact summery points for given data set – it needs to develop UDF for impala queries with existing algorithm implementation.

APPENDICES

Other summarization techniques:

1. Average all the values in a specific interval and get the floor(avg) as y_pos and maximum time_sequence value as the output time_sequence.

```
WITH temptable AS (SELECT time_sequence, FLOOR ((time_sequence - MIN(time_sequence) OVER (PARTITION BY level))/5000000000) AS g, y_pos FROM graph_points WHERE level=0 AND filterfieldvalue='IRR04' AND data_partition_id=551 AND level=0 AND (time_sequence >= 1603947279832890000 AND time_sequence <= 1603956063602808000)) SELECT FLOOR(AVG(y_pos)) AS y_pos_selected, MAX(time_sequence) AS time_sequence_selected FROM temptable GROUP BY g ORDER BY time_sequence_selected;
```

```
+----------------+------------------------+
| y_pos_selected | time_sequence_selected |
+----------------+------------------------+
| 126            | 1603947283961332000    |
| 141            | 1603947289188403000    |
| 156            | 1603947294438627000    |
| 169            | 1603947298596609000    |
| 174            | 1603955991782818000    |
| 220            | 1603956059470845000    |
| 227            | 1603956063602808000    |
+----------------+------------------------+
```

As per this only one graph point has overlapped with existing model.

2. Moving average technique

As explained in (Introduction, 2021), rolling average can be used to summarize the trade points. Per minute a point is plotted over a graph, where that point is based on rolling average of the previous 5 minutes.

This can be altered to implement summarization technique for different levels. Since we can identify the trade points related to each possible interval, we can calculate the rolling average of trade points for 2 intervals (previous interval and current interval). The time sequence selected would be the final time sequence of that interval.

Example:

time_sequence	g	y_pos	
1603947279832890000	0	120	
1603947281149609000	0	123	126
1603947281909799000	0	126	
1603947283073676000	0	129	
1603947283961332000	0	132	133
1603947285059472000	1	135	
1603947286034132000	1	138	
1603947287060625000	1	141	
1603947288102194000	1	144	
1603947289188403000	1	147	148
1603947290511048000	2	150	
1603947291246374000	2	153	
1603947292276991000	2	156	
1603947293311413000	2	159	
1603947294438627000	2	162	
1603947295419112000	3	165	

| 1603947296482587000 | 3 | 168 |

| 1603947297844352000 | 3 | 171 |

| 1603947298596609000 | 3 | 174 |

| 1603955991782818000 | 1742 | 174 |

| 1603956059470845000 | 1755 | 220 |

| 1603956060498245000 | 1756 | 223 |

| 1603956061536288000 | 1756 | 226 |

| 1603956062566090000 | 1756 | 229 |

| 1603956063602808000 | 1756 | 232 |

+---------------------+------+-----+

Query –

```
WITH temptable AS (SELECT time_sequence, FLOOR ((time_sequence - MIN(time_sequence) OVER (PARTITION BY level))/5000000000) AS g, y_pos FROM graph_points WHERE level=0 AND filterfieldvalue='IRR04' AND data_partition_id=551 AND level=0 AND (time_sequence >= 1603947279832890000 AND time_sequence <= 1603956063602808000)),

temptable2 AS (SELECT MAX(time_sequence) AS time_sequence_selected,SUM(y_pos) AS sum_y_pos,COUNT(y_pos) AS n FROM temptable GROUP BY g)

SELECT time_sequence_selected, FLOOR(SUM(sum_y_pos) OVER (ORDER BY time_sequence_selected ROWS BETWEEN 1 PRECEDING AND CURRENT ROW)/SUM(n) OVER (ORDER BY time_sequence_selected ROWS BETWEEN 1 PRECEDING AND CURRENT ROW)) as y_pos_selected FROM temptable2
```

time_sequence_selected	y_pos_selected
1603947283961332000	126
1603947289188403000	133

```
| 1603947294438627000    | 148            |
| 1603947298596609000    | 162            |
| 1603955991782818000    | 170            |
| 1603956059470845000    | 197            |
| 1603956063602808000    | 226            |
+------------------------+----------------+
```

3. Cumulative moving average.

Cumulative moving average from the starting time_sequence also can be used.

Example –

```
+---------------------+------+-------+
| time_sequence       | g    | y_pos |
+---------------------+------+-------+
| 1603947279832890000 | 0    | 120 |
| 1603947281149609000 | 0    | 123 |
| 1603947281909799000 | 0    | 126 | 126
| 1603947283073676000 | 0    | 129 |
| 1603947283961332000 | 0    | 132 | 133
| 1603947285059472000 | 1    | 135 |
| 1603947286034132000 | 1    | 138 |
| 1603947287060625000 | 1    | 141 | 141
| 1603947288102194000 | 1    | 144 |
| 1603947289188403000 | 1    | 147 |
| 1603947290511048000 | 2    | 150 |
| 1603947291246374000 | 2    | 153 |
| 1603947292276991000 | 2    | 156 |
| 1603947293311413000 | 2    | 159 |
| 1603947294438627000 | 2    | 162 |
```

| 1603947295419112000 | 3 | 165 |

| 1603947296482587000 | 3 | 168 |

| 1603947297844352000 | 3 | 171 |

| 1603947298596609000 | 3 | 174 |

| 1603955991782818000 | 1742 | 174 |

| 1603956059470845000 | 1755 | 220 |

| 1603956060498245000 | 1756 | 223 |

| 1603956061536288000 | 1756 | 226 |

| 1603956062566090000 | 1756 | 229 |

| 1603956063602808000 | 1756 | 232 |

+---------------------+------+-------+

Query-

```
WITH temptable AS (SELECT time_sequence, FLOOR ((time_sequence - MIN(time_sequence) OVER (PARTITION BY level))/5000000000) AS g, y_pos FROM graph_points WHERE level=0 AND filterfieldvalue='IRR04' AND data_partition_id=551 AND level=0 AND (time_sequence >= 1603947279832890000 AND time_sequence <= 1603956063602808000)),

temptable2 AS (SELECT MAX(time_sequence) AS time_sequence_selected,SUM(y_pos) AS sum_y_pos,COUNT(y_pos) AS n FROM temptable GROUP BY g)

SELECT time_sequence_selected, FLOOR(SUM(sum_y_pos) OVER (ORDER BY time_sequence_selected ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)/SUM(n) OVER (ORDER BY time_sequence_selected ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)) as y_pos_selected FROM temptable2
```

+----------------------+----------------+

| time_sequence_selected | y_pos_selected |

+----------------------+----------------+

```
| 1603947283961332000    | 126 |
| 1603947289188403000    | 133 |
| 1603947294438627000    | 141 |
| 1603947298596609000    | 147 |
| 1603955991782818000    | 148 |
| 1603956059470845000    | 151 |
| 1603956063602808000    | 163 |
+------------------------+-----------------+
```

ISSUE

Consider the points available for level 300000, the suggested query would result the following,

```
WITH temptable AS (SELECT time_sequence, FLOOR ((time_sequence - MIN(time_sequence) OVER (PARTITION BY level))/300000000000) AS g, y_pos FROM graph_points WHERE level=0 AND filterfieldvalue='IRR04' AND data_partition_id=551 AND level=0 AND (time_sequence >= 1603947279832890000 AND time_sequence <= 1603956063602808000)),

temptable2 AS (SELECT MAX(y_pos) OVER (PARTITION BY g) AS y_pos_selected, LAST_VALUE(time_sequence) OVER (PARTITION BY y_pos,g) AS time_sequence_selected,y_pos FROM temptable)

SELECT y_pos_selected,time_sequence_selected FROM temptable2 WHERE y_pos = y_pos_selected
```

```
+----------------+------------------------+
| y_pos_selected | time_sequence_selected |
+----------------+------------------------+
| 174            | 1603947298596609000    |
| 232            | 1603956063602808000    |
+----------------+------------------------+
```

Actual available points would be,

```
SELECT time_sequence,y_pos from graph_points WHERE (time_sequence >= 1603947279832890000 AND time_sequence <= 1603956063602808000 ) AND filterfieldvalue='IRR04' AND level = 300000 AND data_partition_id = 551;

+---------------------+-------+
| time_sequence       | y_pos |
+---------------------+-------+
| 1603947298596609000 | 174   |
+---------------------+-------+
```

From the starting time_sequence of 1603947279832890000 until 1603947579832890000 would covers first time interval (300000ms from the starting time_sequence). Only first 19 entries belong to that interval. Rest of the final 6 entries belongs to another interval.

```
+---------------------+---+-------+
| time_sequence       | g | y_pos |
+---------------------+---+-------+
| 1603947279832890000 | 0 | 120 |
| 1603947281149609000 | 0 | 123 |
| 1603947281909799000 | 0 | 126 |
| 1603947283073676000 | 0 | 129 |
| 1603947283961332000 | 0 | 132 |
| 1603947285059472000 | 0 | 135 |
| 1603947286034132000 | 0 | 138 |
| 1603947287060625000 | 0 | 141 |
| 1603947288102194000 | 0 | 144 |
| 1603947289188403000 | 0 | 147 |
| 1603947290511048000 | 0 | 150 |
| 1603947291246374000 | 0 | 153 |
```

```
| 1603947292276991000 | 0 | 156 |
| 1603947293311413000 | 0 | 159 |
| 1603947294438627000 | 0 | 162 |
| 1603947295419112000 | 0 | 165 |
| 1603947296482587000 | 0 | 168 |
| 1603947297844352000 | 0 | 171 |
| 1603947298596609000 | 0 | 174 |
| 1603955991782818000 | 29 | 174 |
| 1603956059470845000 | 29 | 220 |
| 1603956060498245000 | 29 | 223 |
| 1603956061536288000 | 29 | 226 |
| 1603956062566090000 | 29 | 229 |
| 1603956063602808000 | 29 | 232 |
+---------------------+----+-------+
```

But there is only 1 entry after summarization for level 300000. Hence there is clear requirement for implementing summarization function with the help of UDF as future works.

REFERENCES

[1]Analytics, T. 1. H. A. T. f. 2. –. T. a. D. i., 2021. *Top 15 Hadoop Analytics Tools for 2021 – Take a Dive into Analytics.* [Online]
Available at: https://data-flair.training/blogs/hadoop-analytics-tools/

[2]Ashish Gupta, F. Y. J. G. A. K. K. C. L. S. W. S. G. D. A. R. K. A. A. B. M. H. J. C. M. S. D. J. J. S. A. G. S., 2014. *Mesa: Geo-Replicated, Near Real-Time, Scalable Data.* [Online]
Available at:
https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42851.pdf

[3]Ashish Thusoo, D. B. R. M. Z. S. N. J. H. L. S. A. J. S. S., 2010. *Data warehousing and analytics infrastructure at Facebook..* [Online]
Available at: https://research.fb.com/publications/data-warehousing-and-analytics-infrastructure-at-facebook/

[4]BigQuery, G., n.d. *Google BigQuery.* [Online]
Available at: https://www.g2.com/products/google-bigquery/reviews

[5]Chinnakali, K., 2015. *The 10 Distributed SQL QueryEngine for Big Data.* [Online]
Available at: https://www.linkedin.com/pulse/10-distributed-sql-queryengine-big-data-kumar-chinnakali/

[6]Collective, S., n.d. *Seven Drawbacks of Traditional OLAP.* [Online]
Available at: https://www.smartdatacollective.com/seven-drawbacks-traditional-olap/

[7]Design, A. K. S., n.d. *Apache Kudu Schema Design.* [Online]
Available at: https://kudu.apache.org/releases/1.3.0/docs/schema_design.html

[8]Devoteam, 2021. *Building a BigQuery Data Warehouse for a disruptive employment agency.* [Online]
Available at: https://nl.devoteam.com/success-stories/building-a-bigquery-data-warehouse-for-a-disruptive-employment-agency/

[9]Dhruba Borthakur, N. J. S. S. A. T. Z. S. S. A. R. M. ,. H. L., 2010. *Data Warehousing and Analytics Infrastructure at.* [Online]
Available at: https://research.fb.com/wp-content/uploads/2010/06/data-warehousing-and-analytics-infrastructure-at-facebook.pdf

[10]Google, 2020. *The Future of data warehousing.* [Online]
Available at:
https://services.google.com/fh/files/misc/the_future_of_data_warehousing_2020_ebook_google_cloud.pdf

[11]Hadoop, A., n.d. *Apache Hadoop.* [Online]
Available at: https://en.wikipedia.org/wiki/Apache_Hadoop

[12]Hari Kumar, M. P. K., 2015. *Apache Storm vs Spark Streaming.* [Online]
Available at: https://www.ericsson.com/en/blog/2015/7/apache-storm-vs-spark-streaming

[13]Hayes, J., 2020. *An Overview of Real Time Data Warehousing on Cloudera.* [Online]
Available at: https://blog.cloudera.com/an-overview-of-real-time-data-warehousing-on-cloudera/

[14]Hsiao-Kang Lina, J. A. H. C.-I. C., 2016. *A Hyperconnected Manufacturing Collaboration System Using the Semantic Web and Hadoop Ecosystem System.* [Online]
Available at: https://www.researchgate.net/publication/307619823_A_Hyperconnected_Manufacturing_Collaboration_System_Using_the_Semantic_Web_and_Hadoop_Ecosystem_System

[15]MapReduce, n.d. *MapReduce.* [Online]
Available at: https://en.wikipedia.org/wiki/MapReduce

[16]Mattoso, M. P. A. B. L. V., n.d. *High-Performance Query Processing of a Real-World OLAP Database with ParGRES.* [Online]
Available at: https://link.springer.com/chapter/10.1007%2F978-3-540-92859-1_18

[17]Mukesh, M. N. S. V., 2009. [Online]
Available at: https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-39940-9_8

[18]Overview, I. -., 2021. *Impala - Overview.* [Online]
Available at: https://www.tutorialspoint.com/impala/impala_overview.htm

[19]queries, O. a. q. l. h. t. w. O., 2021. *OLAP and query language: how to write OLAP queries.* [Online]
Available at: https://galaktika-soft.com/blog/olap-essence-query-language.html

[20]Sai Prasad Potharaju, S. S. A. R. K. T. S. C., 2014. *CASE STUDY OF HIVE USING HADOOP.* [Online]
Available at: https://www.researchgate.net/profile/Amiripalli_Srinivas/publication/314724458_CASE_STUDY_OF_HIVE_USING_HADOOP/links/58c5114192851c0ccbf7fb57/CASE-STUDY-OF-HIVE-USING-HADOOP.pdf

[21]Simplilearn, 2021. *What is Hive?: Introduction To Hive in Hadoop.* [Online]
Available at: https://www.simplilearn.com/basics-of-hive-and-impala-tutorial

[22]Snowflake, n.d. [Online]
Available at: https://www.g2.com/products/snowflake/pricing

[23]Storm, A., 2021. *Apache Storm.* [Online]
Available at: https://storm.apache.org/

[24]UpGrad, 2021. *Top 10 Hadoop Tools to Make Your Big Data Journey Easy [2021].* [Online]
Available at: https://www.upgrad.com/blog/top-hadoop-tools/

[25]WAREHOUSE, C. D., 2021. *CONSOLIDATED DATA WAREHOUSE.* [Online]
Available at: https://galaktika-soft.com/project/consolidated-data-warehouse

[26]Warehouse, T. D., n.d. *Top Data Warehouse Software.* [Online]
Available at: https://www.g2.com/categories/data-warehouse

[27]WSO2, n.d. *Summarizing Data*¶. [Online]
Available at: https://ei.docs.wso2.com/en/7.2.0/streaming-integrator/guides/summarizing-data/