Nat. Volatiles & Essent. Oils, 2021; 8(5): 10486-10508

NVEO
Natural Volatiles & Essential Oils

Ingenious Framework For Resilient And Reliable Data Pipeline

KARI VENKATRAM[1] , GEETHA MARY A[2]

School of Computer Science and Engineering, VIT University, Vellore – 632 014, Tamil Nadu, India

Abstract

Data integration is one of the critical requirements for enterprise systems. Data pipelines are used for data integration among distributed databases. Real-time data integration is a very essential requirement in the Internet of things era. The quality of services (QoS) in data pipelines are impacted due to data integration issues while in data sync up. Present data pipeline systems are focused majorly on the real-time data synchronization, the throughput that is essentially scalable solutions in a distributed data pipeline platform. In this context, data reliability problems such as data inconsistency, incompleteness, and conflict in reconciliation, etc., were not prioritized as controlling and monitoring mechanisms not focused. The ingenious framework presented in this article has a comprehensive framework that includes controller, monitoring, and auditing framework to address those issues. Also considered validation, evaluation, and durability aspects in this solution. Message reconciliation and auto retransmission have been proposed as part of the validation and auditing components to measure the effectivenss of the framework. Reliability, partitioning, and fault tolerance, durability, and scalability are few measures used to evaluate the effectiveness. Features such as auditing, reconciliation, and retransmission of error messages for improving QoS. The Auto-retransmission restricts the error percent (<1%) in the data pipeline and also gives high throughput – 180k/ 90K messages for producer/consumer and latency is as low as 5 milliseconds for 1-12 nodes. Compared to the above measures with industry benchmarks and found the framework outperformed as compared to state of art existing frameworks

KEY WORDS: *Data pipeline, message queue, data reconciliation, message controller, data audit*

Introduction :

Nowadays all the applications are distributed in big, heterogeneous environments, the key consideration in this kind of heterogeneous systems is how to connect all these systems and how can we have data flow among them. In other words, multiple applications are integrated to work together and share data among them. To do that we expect some data is transferred from one system to another system, at a very high speed to cater to the real-time need, reliable data transmission, asynchronous transmission for scaling the environment, etc., All these are possible with help of message queues. Nowadays application development is built with the principles of design such as decoupling, asynchronous communication to improve the scalability as it is of utmost importance.

The message queue is a paradigm change and is derived from the publisher/subscriber message pattern[1]. In this context, publishers will send the messages; subscribers will consume or receive the messages. The message queues are aimed to provide an efficient framework to create a data pipeline and provide message orientation, queuing, routing, reliability, security, and resilient transmission of data with help of asynchronous Messaging[2], concurrency of data transmission, persistence, guarantee the transactions of messages, etc.. However, many of the message queues or data pipelines concentrate on a producer, consumer and broker capabilities, one of the major issues with this is not having a controlled flow in the message queues, due to this tracking the data transactions, auditing, re-transmitting the error data or data not transmitted due to network issues can't be addressed. In this section, we give more focus on introducing the controller and data reconciliation.

1.1. Role of a controller in software architecture:

Until the last three decades, the software applications were not so complex. So all the software applications were built in a single component and there is no separation of concerns in the architecture. Until the end of the 1960s, programmable logic controllers (PLC) played an essential role[3]. Since they are simple applications, maintaining was not a concern then. Later the 1990's the software applications are improved and started handling complex scenarios and large data and is being increased day by day. With the increase of application usage and data usage earlier application, which were built with multiple responsibilities in a single component, become a bottleneck. So the older applications had lots of challenges such as scalability as per the expectations, maintainability, modifiability, etc.. Considering the challenges, Trygve Reenskaug has proposed an architectural pattern Model View and Controller, which aimed to the separation of concerns[4].

The controller component, in the context of Information Technology, could be either a hardware device or a software component. It can manage or controls the flow of the data or process among the different components of the system. Chips, cards, some computer peripheral devices are some of the examples for hardware controllers. Software controllers are programs, which will help in directing the data flows. Consider the login page of an application, user prompted for his credentials, and after successful login, user-directed to concern home page as per the user's preferences. The controller makes this easier to manage such scenarios and it is easy to manage among multiple actions of users.

Controller architecture has some of the advantages such as maintenance is easy as a single controller to direct the applications, ease of collaborating, and work together with a clear separation of concerns. It is easy to debug and test the applications. Finally, the controller in our context is the intelligent component that determines optimal paths and responds to outages and new message flow demands for the ecosystem.

1.2. Role of data reconciliation :

Data pipeline used for moving some data from one application to another application by transforming data from one format to another format by a set of processes. One of the goals of data integration systems is to increase[5]: completeness, concision, and accuracy of the data. Data reconciliation (DR) is a process typically used for data verification and is denoted by reference reconciliation and used for entity matching, linking, and duplicate detection scenarios[5] apart from validating the data between two systems. Usually, this happens in the validation phase of data migration from one system to another system. Here the verification of target data is compared against source data to ensure that the migration has transferred the data correctly. During data migration or transformation, it is possible that the expected data in the target system will mismatch may be due to wrong mappings or wrong transformation logic. Apart from this, due to runtime failures such as issues in network/ transaction failures would lead to inconsistency or data in invalid state. To fix the problems there is a need for a good reconciliation method to identify the issues and a model for fixing the reconciliation problems.

1. Literature Review:

Message queuing is implemented by many organizations as a Service, Hardware, or Open Source. The most commonly used messaging queues nowadays are open source and in a distributed environment. As depicted in Table: 1, there are several tools available now in the industry. Vendors such as Solace, Apigee, and Tervela provided hardware/middleware based messaging middleware solution. Cloud-

based software as Service (SaaS) tools currently available such as IronMQ, StormMQ, and Amazon Simple Queue Service (SQS). Finally, open-source tools such as Apache ActiveMQ, Apache Qpid, Apache RocketMQ, RabbitMQ, Apache Kafka are available.

Table 1: Types of Message Queue

Type	Tools
Hardware / Middleware	Solace, Apigee, and Tervela
Software as Service	IronMQ, StormMQ and Amazon Simple Queue Service (SQS), IBM MQ on Cloud, MuleSoft Anypoint Platform, Google Cloud Pub/Sub, AWS MQ, KubeMQ, Azore QueStorage, Alibaba Message Queue,
Open Sourced	Apache ActiveMQ, Apache Qpid, Apache RocketMQ, RabbitMQ, Apache Kafka, JMS

As described in below Table 2: several message queues are available in the market. As discussed in the above table, there are different types of message queues are available. Hardware/ middleware based queues are one is very specific and hard to customize them, important tools in these areas are described in the below table. Software as service-based tools is the tools, which are software products, built for various purposes and specific needs. All of them are build based on a cloud platform as a service. The details of the message queue tools in this category also described in the below table. Open-sourced tools are mostly from the Apache foundation. Those details also described in the below table. Nowadays among all these open-source Apache Kafka is widely used across the industries,

Table 2: Message Queue tools available in the market

Tool	Description
Solance[6]	This Solace tool is enterprise-grade messaging and streaming, which will work on hybrid clouds, IoT, mobile applications, and event-driven micro services[7]
Apigee	Developers creating connected apps, companies updating legacy applications, and others facilitating data transfer between applications and services can use Apigee[8].
Tervela Cloud FastPath	Cloud FastPath is a cloud-based migration solution that automatically loads content and permissions from on-premises file servers and cloud systems[9]. It enables point-to-point, secure data transfers from your sources directly to Dropbox and easily simulate, configure, schedule and run data transfer activities and jobs[10]
IronMQ	MQ provides a reliable way to communicate between services and components. Highly available, persistent by design, with best-effort one-time delivery, MQ is the most industrial strength, cloud-native solution for modern application architecture[11].
StormMQ	StormMQ is a message queuing service, using the standard Advanced Message Queuing Protocol. StormMQ is a hosted, On-Premises or Cloud solution for Machine-to-Machine Message Queuing using AMQP[11]

Amazon Simple Queue Service (SQS	Fully managed message queues for microservices, distributed systems, and serverless applications. Amazon Simple Queue Service (SQS) is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications[11]
IBM MQ	IBM MQ is messaging and queuing middleware, with several modes of operation: point-to-point, publish/ subscribe ; file transfer[12]
Google Cloud Pub/Sub	Pub/Sub is a fully-managed real-time messaging service that allows you to send and receive messages between independent applications. Schedule
Amazon MQ	Amazon MQ is a managed message broker service for Apache ActiveMQ that makes it easy to set up and operate message brokers in the cloud
KubeMQ	Kubernetes Message Queue and Message Broker, Real-time KubeMQ's containers log streaming with powerful options for filtering and views
Azure Queues.	Azure Queue Storage to build flexible applications and separate functions for better durability across large workloads. Simple, cost-effective, durable message queueing for large workloads[11]
AlibabaMQ	Alibaba Message Queue (MQ) is a professional message middleware as a core product in the enterprise-level Internet architecture. It supports reliable message-based asynchronous communication among microservices, distributed systems, and serverless applications[12]
Apache ActiveMQ	Apache ActiveMQ is an open source message broker written in Java together with a full Java Message Service client. It provides "Enterprise Features" which in this case means fostering the communication from more than one client or serve[13]
Apache Qpid	Apache Qpid is an open-source messaging system that implements the Advanced Message Queuing Protocol. It provides transaction management, queuing, distribution, security, management, clustering, federation, and heterogeneous multi-platform support
Apache RocketMQ	RocketMQ is a distributed messaging and streaming platform with low latency, high performance, and reliability[14]
VMware RabbitMQ	VMware RabbitMQ writes "One crucial feature was guaranteed messaging[13]
Apache Kafka	Kafka is an open source distributed commit log that also happens to have a publish/subscribe interface. The Kafka broker is a great fit for long-term storage of immutable logs in your data layer on which you can perform analytics or data transformations (using KStream). In this regard, Kafka is more like a new type of non-relational database for your log/event data at rest. Apache Kafka was built to solve a log ingestion problem[15]

All these message queues are having their pros and cons from their perspective and have their advantage based on the need for the specific domain they have built on. However most of them often having a challenge is controlling the message queues as part of the message queue cluster, due to this

the reliability, suitability, assessment of failed records, and retransmitting the failed records are often having problems.

The controller component in any of the architecture is aimed to have a controlled flow of the application, which is being controlled. It majorly controls by directing the different requests coming from different users/applications to the required process or service and perform certain control routines and take corrective action in the form of output and send the responses to the requested users/ applications. Therefore, the controller acts as a mediator between different users and services/ data providers. It is responsible to control the data transmission among consumers and producers. It maps the user request into model updates. It will not define the business logic just controls them by mitigating bottlenecks inflow and avoid the crash of the system. A controller plays a vital role in preventing overflow or underflow of the messages in the queue.

Most of the message controllers such as RabittMQ are flow controllers to control the bottleneck in the message flow. Distributed data pipelines such as Apache Kafa are part of the broker used to monitor the liveliness of the cluster, elect new leads in case of a failure of the existing leader, and communicate the new leaders to the brokers. It takes Zookeeper's help to do all these.

The role of the controller in the tools is to avoid some of the quality-related problems. This leads to data reconciliation issues such as 1) Missing records 2) Missing values 3) Incorrect values 4) Duplicated values 5) wrongly formatted data and 6) broken links etc.,

Without the data reconciliation stage, these issues can go unnoticed, severely damage the overall accuracy of the data, and lead to inaccurate insights and issues in the data service layer.

The rudimentary approach used in the message reconciliation process was to have relied on the number of messages counts. It is used to observe and tally the expected number of messages that have been transmitted or not. Typical reconciliation was done like this due to low processing capacity hardware and software available in the earlier era. Besides, to perform attribute by attribute comparison used to take a long time and requires lots of processing capacity. With this approach, only how many missing messages are noticed?

Modern message transmission solutions identify where mistakes have happened and any messages are duplicated etc., were identified with the help of the message reconciliation process. Message reconciliation (MR) is defined as a process of verification of messages during transmission. In this process target, messages are compared with source messages to ensure that the transmission framework is properly transmitting the data. Message validation and reconciliation (MVR) shown in Fig.1 means a technology that uses mathematical models to process information. Whatever the measures to be taken for reconciliation process and verification, a model with the measures taken is calculated and reconciled as shown below.

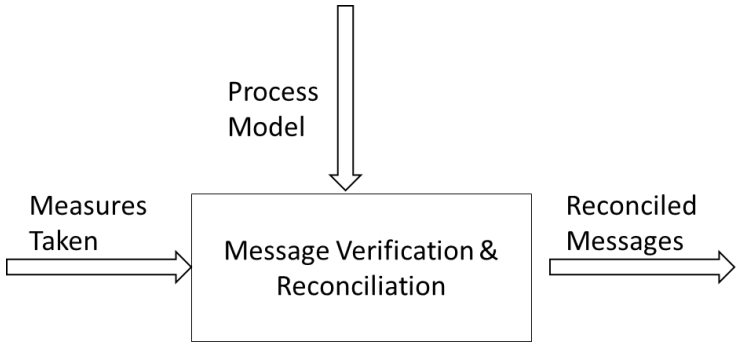

Fig. 1: Message Verification and Reconciliation process.

As part of the message transmission process, it is possible for some mistakes in mapping or transformation logic likely to happen as a human mistake. Some issues such as message failures at the run time also due to network dropouts or broken transactions some messages can get corrupted. These issues need to be identified in the message reconciliation process otherwise this will leads to several quality issues. Message Reconciliation process helps to extract accuracy and reliability in the message transmission. Master Data Reconciliation is a technique of reconciling only the master data messages between the source and target systems. Master data is mostly evolving in nature, and no aggregation operation is done on the dataset. In the case of the master data messages reconciliation process as it is slowly evolving the following measure will be taken to do the reconciliation process. Number of messages, no of the customer, no of items or products, no of active and inactive master messages, etc.,

The reconciliation process is typically has relied on simple record counts to keep track of whether the targeted number of records have migrated or not. However, there are lots of measures that are taken in the current literature to improve the reconciliation process. Tools available in the literature, few important and widely used tools discussed below few reconciliation tools:

Table 3: Reconciliation tools

Tool	Description	URL
Open Refine	Google Refine is transformed as OpenRefine[16]. The tool is useful for database Reconciliation and it is a good framework for cleaning messy data	http://openrefine.org
TIBCO Clarity	This tool is coming as software as a service through web services[16]. It is widely used in ETL framework reconciliation.	https://clarity.cloud.tibco.com
Winpure	It is a data cleansing tool to remove duplicate data, correcting and standardizing them[17]	https://winpure.com/

As shown in the above Table 3, there are few very notified reconciliation tools available in the market OpenRefine (used to call as Google Refine earlier) is a great and powerful utility tool to fix messy data and cleanse the data with it also it enables to transform data from one format into another. It is generally used for DB reconciliation. TIBCO Clarity a cloud-based data preparation tool. It is coming as a software as a service model on-demand service platform. It is generally used in the case of the ETL reconciliation tool. Winpure is a data cleansing, search, and match software tool. This software tool is generally used for cleansing, enhancing, correcting the data. It is used for deduplication purposes.

2. Issues in the data pipelines :

A controller, in a data pipeline context, is a software program that manages or directs the messages flow between a message producer and consumer. A controller in general is a software component, which can interface between two or more components of the ecosystem and manages communications among them. Since there are no proper, controlling components in the existing tools there are several issues related to control mechanisms.

Due to a lack of control mechanisms, the entire data pipeline ecosystem will be clumsy and will be low responsive. Since the controller built within the logic of message transformation, it will not be scalable and agile for any changes. It leads to a less available system, as there is no proper controller in place. Some of the distributed message queues are using a hardware controller for high availability by electing the leader when a node is failed, it will address part of the issue, however, if it is related to the grain of the message then still it can't address the issue. Since there is no proper controller, the mechanism leads to security issues and it is difficult for managing the different users and their access controls. Besides, certain integration-related issues would pop up and lead to data inconsistency. Since there is no controller some standard operating processes also cannot automate, there will be lots of redundant work or manual work due to this. Besides, the controller logic is distributed, it is difficult to analyses the messages sent across multiple consumers. Finally, the operating cost of the pipeline increased.

Hence, there is a need for a better controller component, which can address the above-mentioned issue such as clumsiness, low security, and low integration among the components of the ecosystem, integration issue, difficulty in analyzing the messages, low or difficult in automation and finally cost of operation increases.

Another important and yet not completely addressed in the data pipelines is auditing mechanism in the data pipeline. Data pipeline auditing is the collective measures done to analyze the study and gather data about a data pipeline to ascertain its health following the key performance indicators (KPI) set for the objective of the data pipeline. Data pipeline audit is a way to measure and report on message delivery aspects, and it can provide a comprehensive overview of auditing settings and health the message transmissions, assessments for building a better strategy and precautions to be taken to ensure to improve the trust between the users of the data pipeline. These audits can also focus on data pipeline configuration and change management, data transmissions to help illustrate whether message transmissions were successful, whether out-of-process changes have occurred, whether configuration violations exist, and anything more.

The most critical issue in data pipelines discussed above is the message reconciliation process which was not completely met by the current tools. This approach is to establish the trust between the producer and consumer groups, which is the message producers and message consumers through this data pipeline framework. As shown in the Table:4, due to not having a well-defined reconciliation process, the following issues and challenges are observed. Most of these impact the trust and confidence of the stakeholders and consumers of the data pipeline.

Table 4: Issues due to improper reconciliation

Issue	**Description**
Inconsistency	Messages were not consistently delivered. Not identified on time or notified.
Automation	Inconsistency or no automation process in reconciliation
Timeliness	Timely reconciliation is not done, hence measure to address the discrepancies are not addressed on time
Transparency and Visibility	The reconciliation process is not transparent and not visible to its stakeholders. This leads to distrust in the stakeholders
Confidence	No proper Reconciliation reduces consumer confidence

3. The Proposed Solution :

To address the issues described in the above section concerning controller, audit and reconciliation process, proposed a comprehensive architecture with three major elements such as distributed data pipeline controller and auditing and reconciliation model component

Since data is produced at a faster rate and to avoid the data latency issues [18] and fault tolerance, there is a need for parallel, distributed, and fault-tolerant architecture for data pipeline. All the messages of the dataset will be distributed into multiple nodes and with some redundancy for high fault tolerance and parallel data storage and processing. This architecture will enable storing the dataset into multiple nodes in a clustered environment, hence the data storage and retrieval will be done in collaboration with multiple systems and avoid data latency issues. As described in Fig. 2, the data producer will register the schema and publish the dataset into the distributed data pipeline into a cluster of nodes with some replication factor through a data pipeline controller. The focus of the data pipeline controller is to perform cluster administration, monitoring, controlling, and security maintenance. Consumer queries the schema registry, get the schema details for the message group, and consume the data through the control layer as per the schema definition. Messages are stored in a cluster of nodes; hence, it can perform parallel operations to fetch the messages from the pipeline. This architecture will improve the performance of the end-to-end data pipeline and improve the latency issues in the data pipeline.

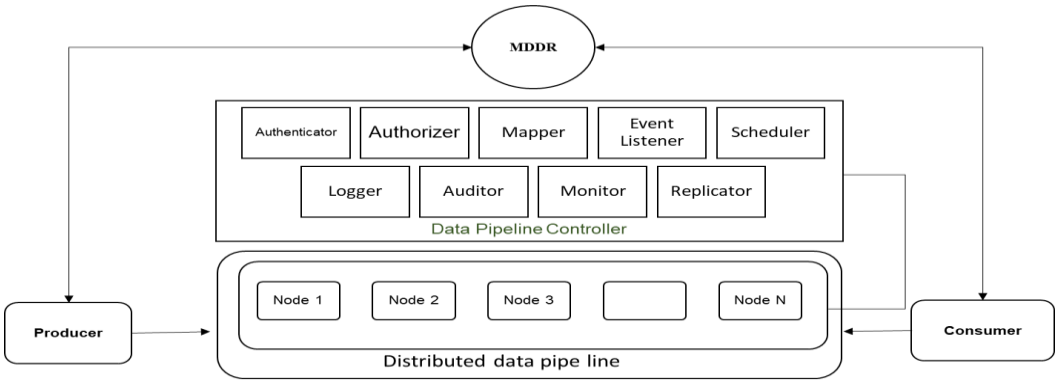

Fig. 2: Data pipeline controller and monitoring system

As shown in Fig. 2, the controller has the following main components that collaborate to flow the data pipeline as expected.

Authenticator: This provides a digital authentication that will verify and enables stronger security to the data pipeline.

Authorizer:

This provides and validates the user roles and responsibilities of the logged-in user in the data

Pipeline.

Mapper:

This provides an integration point to join multiple producers of the same group to come up with a common understanding among the data pipeline users.

Event Listeners:

This is responsible for listening to various requests coming from stakeholders in the data pipeline. It is used executing the re-transmission job in case of any error in the message transmission. In case of event occurs, it will take appropriate action.

Schedulers:

Scheduler is responsible for scheduling job. Especially it is used to run the audit and reconciliation jobs.

Logger:

This component is used for logging the transactions/ events for subscribers and consumer transaction metadata for validation, monitoring, and audit purpose.

Auditor:

This component is the key to message auditing across the data pipeline.

Monitor:

It's a component used for event management in the data pipeline and provides some dashboards as an extension to this.

Replicator:

This component is used to configure the replication. It will repeat the messages in the event of message loss or error in transmission.

We propose JSON based schema, as it is a simple notation and can build a flexible schema. It can allow not only structured data and allows unstructured or hierarchal data. Since it is distributed architecture, it can easily handle the stream of data into the cluster with a higher data transmission rate for publishing and consuming the data.

The controller framework will take care of two major issues in the data pipelines such as message buffering, and congestion avoidance. The message buffering refers to the message that has been received with enough information to process the message in the data pipeline, buffering means that message is till getting and will continue till later point of time. Message congestion control is a process that can be ignored whatever the message prologues, epilogs that are not important for the message are removed to improve the quality and avoid message congestions. The controller is the initial contact point for handling all requests in the data pipeline. The controller would delegate the requests to some of the modules to complete the authentication and authorization of a user.

The centralized controller is a set of controller nodes in a distributed system that handles all the requests coming from the publishers and subscribers in the data pipeline ecosystem. This implementation of centralized control that avoids using multiple controllers is desirable for enforcing organization-wide policies such as user tracking and security.

Controllers can increase security by mandating a reasonable security baseline across an entire data pipeline. It will reject the message consumption or production that does not meet the security authentication, authorization, and any specific requirements of the organization. It also enables the system to enforce the adherence to certain practices such as having good labels, annotations, resource limits, etc.,

The Controller is a set of software components that orchestrates several message functions in the data pipeline. It serves as a message controller or mediator among the message consumer and producer. This controller framework consists of the following functions with the help of different modules mentioned in Fig 2, such as avoid fault, more configurable flow, accountability in the transmission, performance improvement, and security, etc.,

The audit mechanism in the framework is not to just identify what is the issue exist rather it also probes to unravel why is it exists. Mostly issues are deep-rooted and complex to identify the root cause of the issue. As a result, it is a tedious and time-consuming work if there is no automation process involved in it. The monitor module performs message queue auditing from a single point of view. It not only tracks unauthorized and erroneous messages that transmitted in the data pipeline. It enables an automated inventory reporting and policy violation audits to track and report the discrepancies in the message transmissions. This report is a candidate for data reconciliation and retransmission to fix the reconciliation issues. This monitoring and auditing enable addressing the risk of data quality issues, the ability to execute the reconciliation process, and retransmission strategy.

Message Reconciliation is one of the key for data quality. In general, as and when a message reconciliation issue is identified, a notification should be sent to the support team to execute necessary actions to address the issue. The frequency of message reconciliation is also an important aspect to address the issue. One time reconciliation process can perform ad-hoc reconciliations and it is done as and when it is required especially when the new services are adopted prior data is migrated and reconciled. Another process is scheduled reconciliation process, this would be scheduled as per the business requirement hourly, daily or weekly reconciliation as per the schedule. Another key aspect in the reconciliation on other than ad hoc / scheduled reconciliation is how the message reconciliation to be done, which essentially means that how the message reconciliation to be done. There are levels of the reconciliation process such as 1) High aggregate level where the messages are reconciled at the highest level, which is rudimentary just check at the department level or producer level (Only primary attribute) whether the message is transmitted or not. 2) Medium level of aggregation where apart from the department/ producer other additional criteria are also checked (few attributes) and 3) low aggregation level where more no of attributes are reconciled. There are a few key challenges in the message reconciliation process discussed below. This process depends on the complexity of the data pipeline such as many producers and consumers in the system, what are the metrics used to reconcile the messages. Message processing time largely depends on the number of messages that are transmitted in the system. Producers and consumers have to have a meaningful filter selection and based on that cluster should be selected. Other key issues see here is the producer's message quality. If the quality of the message is poor then message reconciliation will not impact it. Some of the key considerations in the reconciliation process are maintainability, supportability, and platform for reconciliation.

To address the reconciliation issues, it is required to retransmit the message again; to do this it is important to persist the message in the data pipeline.

One of the important issues that need to be considered in the data pipeline is the loss of the data introduced by any system or a layer within the cluster of nodes. At times, some data packets may corrupt and message transmission may fail during data replication. This will affect the data replication to the downstream system. To avoid such issues the distributed pipeline must be in a position to hold the data

messages for some time so that in case of data loss, that can be retransmitted or re-replicated by changing the offset to the particular consumer, so as data can be recovered. There will be a purging process for reconciled messages to avoid unnecessary data storage.

The comprehensive architecture of the data pipeline is shown in Fig. 3. In this solution as mentioned in the above sections, a controlled distributed data pipeline along with the schema registry is deployed. In this solution, diversified producers such as Web user interface, mobility data providers, TPDP, Hadoop, enterprise data warehouse, etc., produce the data in different periods. Since it is going through the controller layer that will tag the timestamp with the data and ensure the data consistency with this framework.

As and when any data is generated immediately it will be published into a distributed clustered environment and at the same time, the consumer can consume the data from the cluster. Since it is not depending on any relational database or any data, repository service after publishing, in case of database outage, will not influence the subscribers as the messages are persisted in the log. If any transaction fails, it is possible to retransmit the same message from the queue. There is no need to run multiple threads over the nodes. Consumers will be able to pick the messages as and when it is published by the publisher through configuring the time of refresh at the consumer side. This will resolve all the above-mentioned issues in the legacy architectures and modern architecture having limitations in pub-sub and a hybrid model with a standard service producer can push the data and consumers can pull the data.

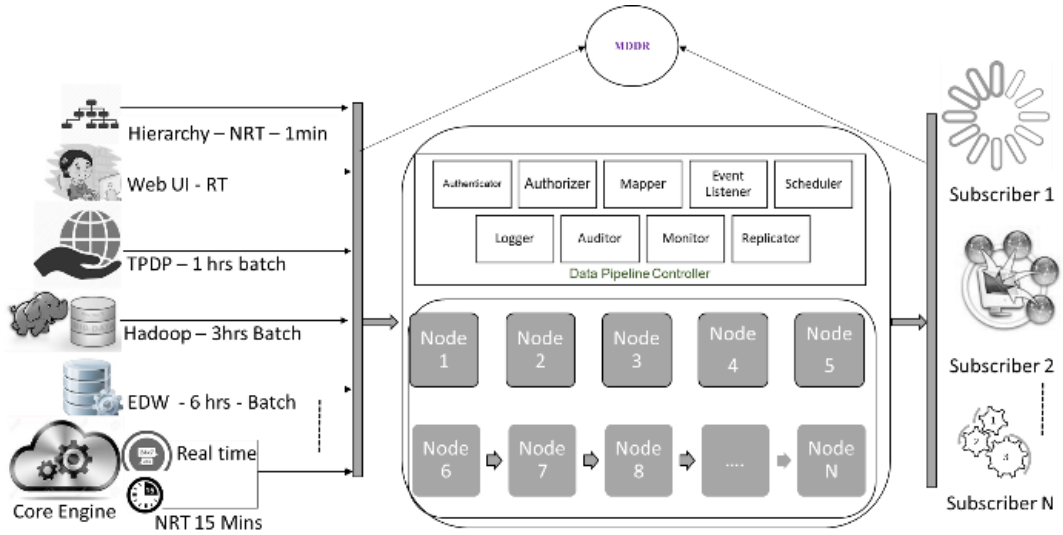

Fig. 3: Comprehensive architecture for data pipeline

In the given comprehensive architecture for data pipeline, Apache Kafka can be used as distributed data pipeline framework. The controller for the data pipeline to be implemented as mentioned in the above section.

As shown in the above Fig. 8, the cluster will interact with two external entities such as Hierarchy – NRT, Web UI, TPDP, etc shown at the right side of the figure called producer and subscribers are mentioned at the right as subscriber 1, subscriber 2, etc.. The middle layer consists of Schema registry and data

pipeline controller to control monitor, administrate the entire cluster along with security enablement. There will be a self-healing mechanism provided to the cluster utilizing the way how zookeeper instance works in distributed node clusters.

4. Results and Discussions:

Quality metrics described here are key performance indicators for your web application or web site. Response metrics show the performance measurement from a user perspective while volume metrics show the traffic generated by the load-testing tool against the target web application.

Three general types of errors can occur in message transmission in the data pipeline. random error, systematic error, and gross errors. These are audit findings in the data pipeline.

Process Measurements: Usually measured process messages inevitably inaccurate form due to the messages are produced from the sources are may not be perfect as per the consumer requirement or the transmission logic and systems we use may not be accurate. It is assumed that any observation is composed of a true value plus having some error in the message transformation as shown in the equation -1.

$$Y = X + E \quad (1)$$

Where Y is the received messages, X is produced messages, E is an error in the transmission.

How do we measure the error (E) in the transmission? This can be a bone random error (RE), Systematic Errors (SE), and Gross Error (GE).

So the above statement can be rewritten as below below equation - 2

$$Y = X + RE + SE + GE \quad (2)$$

Random is also called as an indeterminate error, which can occur due to uncontrollable changes in variables that affect reconciliation. For example. In a stream of messages in a data pipeline while testing if there is some volatility in a field, which means keep changing infrequent intervals would cause random errors and are often reported with measurements because random errors are difficult to eliminate. Random error is caused by one or more factors that randomly affect the error measurement. It follows Gaussian distribution.

Systematic errors also called determinate errors generally happen due to personal mistakes and cause skewed data. Examples of this type of error could be due to wrong logic message transmission in the data pipeline that could go wrong due to wrong transformation logic. Systematic errors can be identified and eliminated after careful inspection of the experimental methods, validate and audit.

Gross error is another measurement of error, this is caused due to hardware failures, etc., Unlikely random errors gross errors are outliers and are so far or below the true value consistently. These messages are usually should be discarded. These kinds of issues are determined using Q-Test and those messages are discarded.

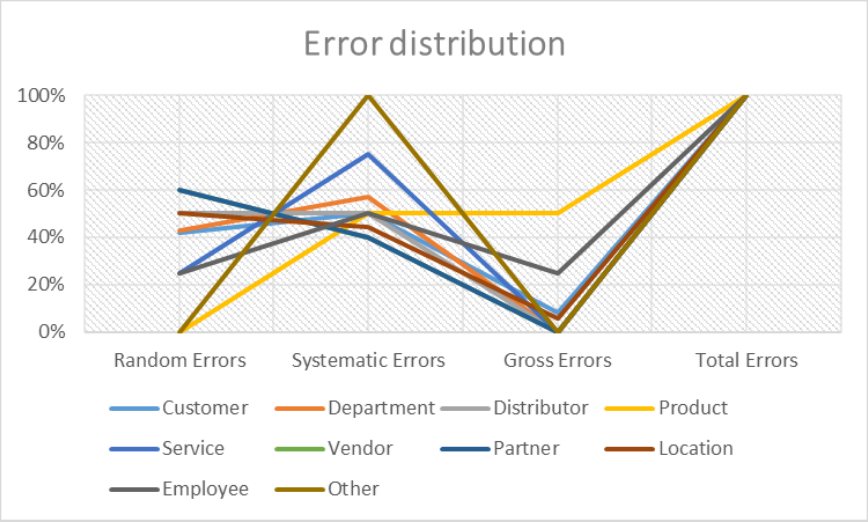

Fig. 4: Error distribution

As depicted in the above Fig. 4, how many errors are transformed during the data message transmission? 10 types of messages are tested with a reasonably good number of messages and tried to analyses the errors. Vendor and partner messages have a maximum of random as high as 60% of total errors. Products and other messages have low or no random errors. Service messages have maximum systematic errors while vendor and partner systematic errors are low. Usually, gross error messages are very small compared to both random and systematic errors. Out of this product, employee and location have some gross errors while others have no significant gross errors.

As depicted in Fig. 5, as observed random errors with error percent of total messages, delivered, vendor messages random errors is more compared to other message groups and products and others are negligible random errors. Again the systematic errors also maximum as shown in the case of vendor messages and employee & product messages have a very low contribution in systematic errors. Overall gross errors are very negligible and customer, product, and the employee has contributed a bit to it. Total errors% is high in the case of vendor and the product is the low percentage of errors.

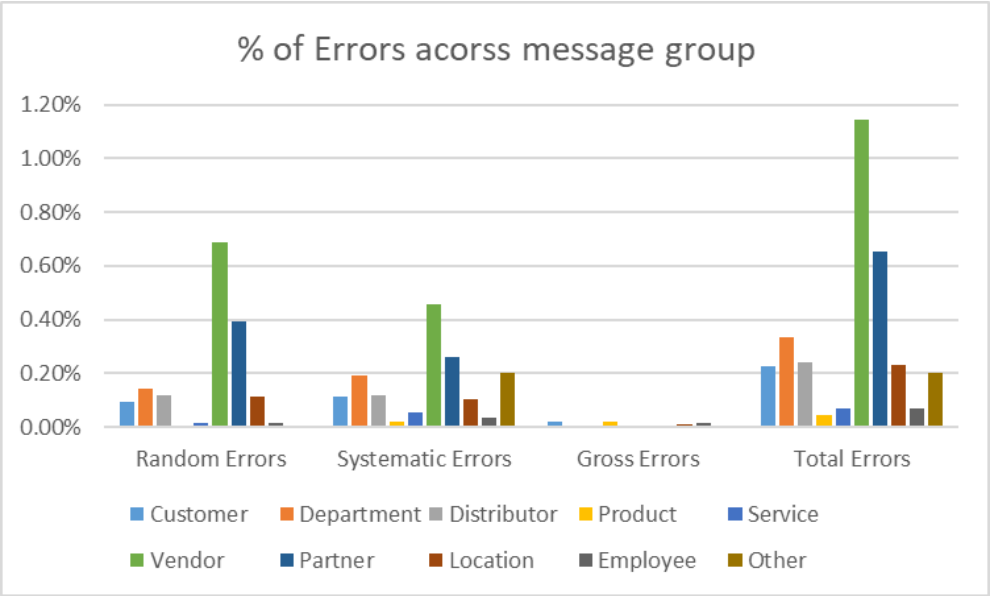

Fig. 5: Error distribution

After error detection, it is important to report reconciliation issues. Reconciliation issues are reported with help of below attributes shown in table 5. No messages transmitted, planned reconciliation, pending for reconciliation, no of messages reconciled, no of issues identified, how many are fixed, and how many are yet to fix and what is the overall status.

Table 5: Attributes for reconciliation

#	Attribute	Description
1	Total	The total number of messages
2	Planned	The number of planned reconciliations
3	Pending	The number of pending reconciliations
4	Completed	The number of completed reconciliations
5	Issues	The number of reconciliation issues
6	Open	The number of open issues to fix
7	Fixed	The number of issues fixed.
8	Actual Completion	The percentage of completed as against the total reconciliations. (#4 /#1)
9	Plan Completion	The percentage of planned reconciliations. (#3/1)
10	Status	The indicator provides the status of saying based on the difference between the plan vs actual.

The reconciliation dashboard is shown in Fig. 6 for all the message groups of an organization at a given point in time. From this, we can easily conclude that the reconciliation process is very good as per the plan and could achieve a high percentage as per the plan. There are few errors yet to be fixed those are reported in the dashboard. This report is very clean and neat to present and take appropriate action. The system administrator will share this dashboard for concerned teams and DevOps engineers and SRE

to ensure the messages are delivered correctly. Once the reconciliation process completes the overall data pipeline message delivery quality will be improved.

Message Group	Total	Planned	Pending	Completed	Issues	Open	Fixed	Actual Completion	Reconciliation Status	Issue Status
Customer	5250	4725	420	4830	11	1	10	102%		
Department	4200	3570	546	3654	12	0	12	102%		
Distributor	2500	2375	150	2350	6	0	6	99%		
Product	4567	3654	822	3745	2	0	2	103%		
Service	5675	4256	1249	4427	3	0	3	104%		
Vendor	875	700	131	744	9	0	9	106%		
Partner	2300	2070	230	2070	14	1	13	100%		
Location	7800	6630	780	7020	16	0	16	106%		
Employee	5600	5040	280	5320	4	2	2	106%		
Other	500	500	0	500	1	0	1	100%		

Fig. 6: Reconciliation Dashboard

Apart from the error and reconciliation process, we also evaluated quality attributes such as throughput, scalability, reliability, etc., of the framework to further the reliability of the framework.

Table 6, shown below describes the experimental results taken from our experiment. This report has min, max, and an average of two key performance measures such as response time, throughput considering 5 producers and 5 consumers in the data pipeline. The reported empirical results exhibited in the table are the average of four runs, with each run lasting for 12 minutes. 12 minutes of the run we considered validating if there are any variability and fluctuation that may arise in the data pipeline. In this experiment we see that response time is also linearly increasing concerning no of messages and throughput is also averaging as the number of messages increased. The response time and throughput are better and considerable[19].

Table 6: details of experimental results

#	Response Time			Throughput (Req/ Sec)		
Req	Avg	Min	Max	Avg	Min	Max
100	12	9	15	98	79	122
200	26	20	28	195	186	215
400	53	45	57	389	372	405
800	107	103	110	770	760	802
1600	222	216	242	1532	1492	1621

3200	446	407	459	3055	2989	3325

In their literature V. John and X. Liu discussed various message brokers in the article “A Survey of Distributed Message Broker Queues”[20] and concluded reliability is a distributed message broker using Kafka is unreliable because the sender does not receive any acknowledgment. Comparing Kafka with Active MQ is reliable. Also mentioned Kafka has higher throughput whereas Active MQ has lower latency and hence it is reliable. Concerning latency the consumer push-model option in Active MQ resulting in low mean latency as compared to Kafka pull-model[20]

So considering these fundamental and tradeoff issues, we featured this framework as performance consistent reliability mechanism and it offers high throughput, moderately low latency message queuing with better reliability. Reliability is achieved in this framework through a selective acknowledgment mechanism. Usually when a consumer receiving messages consumers used to acknowledge when all the messages are processed up to the offset given. If any unforeseen issue happens and the system is restarted then the framework starts consuming or processing the messages from the last offset updated in the system. This is a usual scenario, however for instance a message was wrongly consumed or data packet is lost or something went wrong after consumption of the message, then how to correct the message(s) consumed already?. This is where traditional messaging queues will enable us to go back and redeliver those messages, whereas DDP has a limitation, as there is no selective message acknowledgment is implemented out of the box. Proposed selective message acknowledgment mechanism in this framework, to address this issue. This can be achieved with the help of additional markers as an additional message topic(s). This additional marker topic enables messages to track starting from the start of message processing until it is processed. In this markers topic start and end, markers will be updated for each message being processed. This will be used for tracking the messages; in case of any issues, the messages can be redelivered. Hence, the reliability of the messaging framework is enhanced.

Error preventive mechanisms also conceptually introduced in this framework. This means in case a downstream system is failed or down due to some technical issue and as the reliable mechanism of redelivering messages will continue and it will continue to do it even though the downstream system is not responding, due to message redelivery mechanism. Then system network bandwidth is chocked as the flood of redelivered messages. This has to be identified and stop sending for redeliveries. In such scenarios, the system should stop sending those message redeliveries. This can be achieved by keeping verifying the redeliveries within defined intervals. If the test message verifies and confirms there is no issue, and then the messages again are redelivered, otherwise it will stop redeliveries.

In the current digital world, there is a need for high-throughput processing systems as there is a rapid increase in the amount of data generated and processed to do business analysis[21]. Hence evaluating the throughput of the system is the key function. As described in Fig. 7, the framework performance evaluated with throughput. We tried this framework built with 3 nodes server as a cluster, each node has consisted of multiple producers and consumers jobs running. In Fig. 7, the X-Axis is depicted with 5, 10, 15, 20, 25, and 30, which are no producers and consumers per node. Y-Axis is no of messages (throughput) per second.

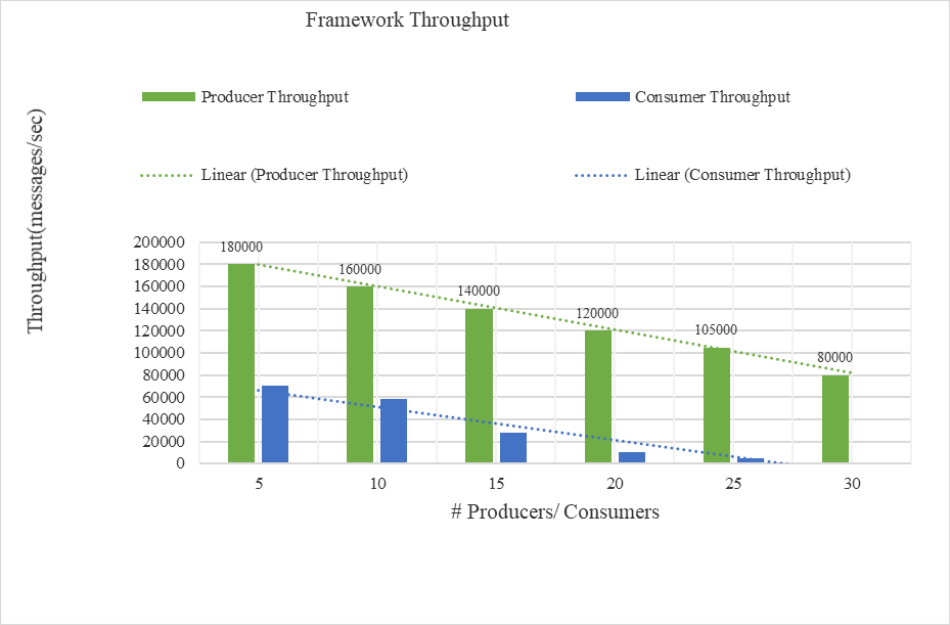

Fig. 7. Producer and consumer wise throughput of the framework- #messages/sec

Vineet John and Xia Liu in their article "A Survey of Distributed Message Broker Queues", they have mentioned, in their experiment with 3 nodes with a single producer they observed around 190K messages/ sec throughput [20]. Compared with that in this experiment considered 5 producers and achieved 180K messages/ sec. Hence, it seems the framework is good at its performance compared to the benchmark mentioned.

Another measure to be considered to evaluate the performance of the framework is the latency of message delivery. Message delivery latency is defined as how long does it take a message to deliver to the consumer. Evaluating this conducted repeated requests between producer and consumer and measured the message delivery time, how long it takes for a producer to send, and then consumed by consumers in the proposed cluster.

In other words, latency is the difference of the time clock by comparing clocks of a producer and a consumer of an event or message. In this framework as depicted in Fig. 8, the average latency or average time is taken for each message to process is around ~11.5 milliseconds with 1 to 12 nodes cluster. Tried to evaluate the latency for between 1 to 5 threads on different nodes. Besides, measured 95% time of each message process is taking around 23 milliseconds with multiple producers and consumers per node at the rate of 22.5K messages/ sec. In the article, Kafka with selective acknowledgments performance & latency benchmark, author A. Warsi, benchmarked for latency between message send and receive is always either 47 or 48 milliseconds[22]. The benchmark of latency with a similar set up is ranging between 35 to 50 ms [21]. Comparing with these two results[23], the framework performance seems to be good and acceptable. We observed with sample test data in this 3-node cluster close to the framework consumed around 22500 messages per second with message size 2-3KB. This is a very good efficiency model[24]...

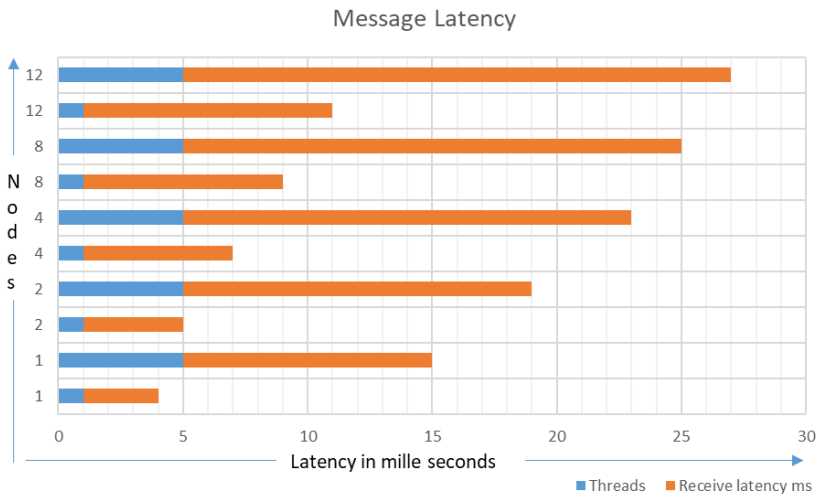

Fig. 8. Average latency of the message

All measurement of analysis of errors has some degree of uncertainity. The process which help in evaluating the uncertainity is called error analysis[25] of uncertain events. We bring this point as we are doing error analysis and provide a way and interpret the error analysis done in this article. So if we measure the reconciliation we estimate and add uncertainity to measure it as shown in the following formula- 3.

$$\mathrm{m} = \mathrm{be} + \mathrm{uc} \quad (3)$$

where m is measurement (total messages to be received), be is best estimate (expected messages to be received) and uc is uncertainty (error messages) in the analysis. For instance you are expecting 3500 messages to be received but you have received only 3475 messages and rest of 25 messages are errors in ths as per the above formula. Uncertaininty here is 25 and which is 0.71%.

Accuracy and precision are two measures will help in measuring the quality of the transmission of errors. Accuracy is closeness of expected messages vs total messages[26]. Precision of degree of reproducibility of the same result[27]. The precision is also called as relative measure[26, 27]. So precision is defined as below formula - 4

$$\text{Precision} = \text{error} / \text{total no of records} \quad (4)$$

As per the above example precision is 0.0002 and which is 0.02% and accuracy is relave error. Which is calculated as below formula 5.

$$Accuracy = \left(\frac{Measured\ Value - Expected\ Value}{Expected\ Value}\right) \quad (5)$$

So the accuracy/ relative error here is -0.0071 and is close to -0.7%.

Can we say error disributions happened as shown in figure 5 are meeing our expectations? In general it is agreed that the measured errors are in the lies within the range of uncertainity based on the scenario. Also in the same way the error values of two measures have standard uncertainity ranges they are said to be consistant and they are to be agreed (range <0.5 %). As per the experiment results below are the most of the measure have relative measure and relative errors are very linear.

So as depicted in the Table :7, most of the message groups relative measure are ranging from 0% to 1.1% however most of them are below 0.5% and relative error also -0.2 to 0.6% . From this we can conclude that the precision and accuracy are good and we should consider the error in a considerable range and this can be accepted.

Table 7: Relative measure vs error

Message Group	Relative measure	Relative error
Customer	0.2%	-0.3%
Department	0.3%	-0.2%
Distributor	0.2%	-0.3%
Product	0.0%	-0.5%
Service	0.1%	-0.4%
Vendor	1.1%	0.6%
Partner	0.7%	0.2%
Location	0.2%	-0.3%
Employee	0.1%	-0.4%
Other	0.2%	-0.3%

Also as shown in the table 5 and figure 9, the throughput is as high as 190K messages/ sec and hive good average response time and average though put. Also the Latency is low and linearly distributed to number of threads.

In this section, we evaluated the architecture with other parameters and conducted the assessment to validate the reliability and reconciliation model of the framework. As data transmission is key in the data pipeline it is important to ensure the delivery of the data to subscribing systems without any transmission errors, transaction failures, and timeliness of the data transmission [28]. In case of any transmission, errors, and failures the system should be in a position to identify the failed records and must re-delivered the failed records. At the same, time the performance and scalable aspects also to be assessed.

It is important to measure the performance of the system from time to time for providing quality services to the application consumers. Based on the need for the system performance and scalable aspect one has to wisely decide the size of the cluster based on the service level agreement.

Data Reconciliation: Data reconciliation is a process aimed to eliminate random errors formulated as a constraint optimization problem [29]. In any data pipeline, one of the important measures to consider is a data reconciliation between producer and consumer. Data produced vs data consumed by different subscribers are compared automatically online or offline infrequent intervals such as hourly, daily, etc. to identify the data replication issues. As discussed above in this section, reliability is achieved in this framework through a selective acknowledgment mechanism. Markers' topic is one additional topic, which used to provide the reliability of the message by tracing the starting of a message until the message processing. Therefore, with this approach, it is easy to reconcile the data produced by the producer against the message consumed by the consumer. In the auto reconciliation model as discussed

above if markers offset is not matching with consumer offset by using selective acknowledgment the messages from the previous offset the messages are redelivered. In offline mode messages, producer count is tallied with corresponding subscribers offset count. In case of any mismatches, identify the offset where the data is in harmony, and then redeliver all the subsequent messages after the offset.

Partitioning enabled high throughput: As this framework is providing good efficiency concerning throughput. This is achieved through topics and their partitions; each topic is divided into many partitions. As each partition enables parallelism, data in the topic will split across multiple brokers and each partition is hosted in a separate node so that multiple consumers can read parallel from a topic. This is one of the main reasons to achieve high throughput.

Fault tolerance: Fault tolerance is achieved in this framework under replication. This is a very necessary functional requirement for any distributed stream platform[30]. This framework protects against failures by replicating the data. Replication is achieved by partitioning and replicating the same data into multiple partitions[31]. One of the partition is will be designated as leader and others are followers of the replica/ partition. The leader will ensure the followers are in sync with it and protects from faults and failures. If any of the nodes fails, other replicas will start serving. To test the fault tolerance of the system, we experimented with the system by killing one of three nodes in the cluster, and still, it is survived to process the messages with minimal additional latency.

Reliable: The framework is reliable as it stores the data in a distributed commit log and can redeliver in case of failures at any point in time. Offset is the one that can help to redeliver from the data log.

Scalable: The scalability of any data pipeline platform is having the ability to integrate and coordinate with new nodes or systems[30]. Since this framework is a distributed framework, it can horizontally scalable. It is a scale-out approach and it is easy to add no nodes as per the business use case and achieve high scalability.

Durability: Since this framework uses a distributed commit log, where the messages are being persisted in the disk at a very faster rate and it enables durability.

Resource consumption: Estimation of resource consumption is hard to talk in the distributed data pipeline[32]. Resource consumption can be measured in detail for CPU and memory with the proposed monitoring tool for administration.

In this experiment, we have mainly evaluated the framework with both producer side and consumer side measure. The following measures were evaluated a) Throughput b) Response Time/Latency and validated quality measures such as auditing of data due to any Timeouts and lag at consumer or producer. We observed better throughput for both at producer and consumer side and low latency at the consumer side even for higher nodes and threads. The major callout of this experiment is that the no of nodes and threads being increased the latency is being flattened accordingly.

5. CONCLUSION :

This article illustrates the data pipeline's importance of controller in improving the quality and assessment process in the data pipeline. Compared to various data pipelines in the literature. Briefly discussed various measures and methods to verification and reconciliation process of data being transmitted in the data pipeline. Highlighted various challenges present in the state of art frameworks

currently available in the industry. The proposed comprehensive, distributed, and controlled data pipeline architecture methods can resolve all the issues and challenges. Data pipeline quality of service improvement plans with the help of the Controller component proposed to have separation of concerns. As every request passes through controller and data pre-processing, filtering and any delegation needed can be performed also security constraints can be imposed in this layer. The proposed auditor, monitor, replicator, and scheduler will ensure the quality of the services in the pipeline. Data persistence enables the re-transmitting only required data if there are any transmission errors or data loss during data transmission. Therefore, with this comprehensive architecture, assessment, and diagnosis framework discussed is ready for addressing the issues discussed with the assessment model built within the framework. In our experiment average response time, average though put is much better and in acceptable limits. Also, audit quality measures such as errors and reconciliation metrics are very acceptable limits. With this, we conclude that the proposed a resilient and reliable framework and which is efficient to control the entire data flow within the ecosystem and enables high data quality transmission over the pipeline. The efficiency and scalability of the framework can be achieved by simple configurations. Auditing and reconciliation is the best part of the framework and can guarantee the data sync ups. Since there are many interesting factors in this framework, our research group is working on further implementation of this framework.

REFERENCES :

[1] E. Dalkıran, T. Önel, O. Topçu and K. A. Demir (2020), Automated integration of real-time and non-real-time defense systems, Defence Technology,http://https://doi.org/10.1016/j.dt.2020.01.005

[2] B. Zheng, E. Holland and S. C. Chapman (2016), A standardized workflow to utilise a grid-computing system through advanced message queuing protocols, Environmental Modelling & Software,84,304-310, http://https://doi.org/10.1016/j.envsoft.2016.07.012

[3] L. C. Tasca, E. Pignaton de Freitas and F. R. Wagner (2020), A study on the performance impact of programmable logic controllers based on enhanced architecture and organization, Microprocessors and Microsystems,76,103082, http://https://doi.org/10.1016/j.micpro.2020.103082

[4] N. Prokofyeva and V. Boltunova (2017), Analysis and Practical Application of PHP Frameworks in Development of Web Information Systems, Procedia Computer Science,104,51-56, http://https://doi.org/10.1016/j.procs.2017.01.059

[5] A. Bakhtouchi (2019), Data reconciliation and fusion methods: A survey, Applied Computing and Informatics,http://https://doi.org/10.1016/j.aci.2019.07.001

[6] Solace PubSub+,

[7] M. S. Writer. (May 20, 2020). Solace's New Event Management Portal Simplifies Event-Driven Architecture ed.). Available: https://martechseries.com/sales-marketing/customer-experience-management/solaces-new-event-management-portal-simplifies-event-driven-architecture/

[8] R. Sturm, C. Pollard and J. Craig, "Chapter 11 - Application Programming Interfaces and Connected Systems," (in Journal, Type of Article vol. no. Issue, p.^pp. 137-150, 2017/01/01/ 2017, Art. no. Article.

[9] C. FastPath. IT'S MORE THAN JUST MOVING BYTES ed.). Available: https://www.cloudfastpath.com/

[10] J. Heinonen, H. Flinck, T. S. Partti and J. T. Hillo, "Method of operating a network entity," (in Journal, Type of Article vol. no. Issue, p.^pp. 2019, Art. no. Article.

[11] R. Cohn (2011), A comparison of AMQP and MQTT, White Paper, StormMQ,

[12]C. Ding, D. Chu, E. Zhao, X. Li, L. Alvisi and R. van Renesse, "Scalog: Seamless Reconfiguration and Total Order in a Scalable Shared Log," in 17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20), 2020, vol., no., p.^pp. 325-338.

[13]V. M. Ionescu, "The analysis of the performance of RabbitMQ and ActiveMQ," in 2015 14th RoEduNet International Conference-Networking in Education and Research (RoEduNet NER), 2015, vol., no., p.^pp. 132-137: IEEE.

[14]M. Yue, Y. Ruiyang, S. Jianwei and Y. Kaifeng, "A MQTT protocol message push server based on RocketMQ," in 2017 10th International Conference on Intelligent Computation Technology and Automation (ICICTA), 2017, vol., no., p.^pp. 295-298: IEEE.

[15]M. Kleppmann and J. Kreps (2015), Kafka, Samza and the Unix philosophy of distributed data,

[16]R. Verborgh and M. De Wilde, Ed.^Eds. Using OpenRefine, ed. (no.). Packt Publishing Ltd, 2013, p.^pp.

[17]A. Karrar and M. Ali (2016), Comparative Analysis of Data Cleaning Tools Using SQL Server and Winpure Tool, International Journal of Computer Applications in Technology,3,371-377,

[18]Z. Milosevic, W. Chen, A. Berry and F. A. Rabhi, "Chapter 2 - Real-Time Analytics A2 - Buyya, Rajkumar," (in Journal, Type of Article vol. no. Issue, p.^pp. 39-61, 2016, Art. no. Article.

[19]K. Salah and T. R. Sheltami, "Performance modeling of cloud apps using message queueing as a service (MaaS)," in 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), 2017, vol., no., p.^pp. 65-71: IEEE.

[20]V. John and X. Liu (2017), A Survey of Distributed Message Broker Queues, arXiv preprint arXiv:1704.00411,

[21]S. Venkataraman, A. Panda, K. Ousterhout, M. Armbrust, A. Ghodsi, M. J. Franklin, B. Recht and I. Stoica, "Drizzle: Fast and Adaptable Stream Processing at Scale," presented at the Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, 2017. Available:

[22]A. Warski (2017), Kafka with selective acknowledgments (kmq) performance & latency benchmark, Softwaremill,

[23]H. Gong, R. Li, Y. Bai, J. An and K. Li (2018), Message response time analysis for automotive cyber–physicalsystems with uncertain delay: An M/PH/1 queue approach, Performance Evaluation,125,21-47, http://https://doi.org/10.1016/j.peva.2018.07.001

[24]J. Kreps, N. Narkhede and J. Rao, "Kafka: A distributed messaging system for log processing," in, 2011, vol., no., p.^pp.

[25]V. M. Kuntsevich and A. V. Kuntsevich (2002), ANALYSIS OF THE PURSUIT-EVASION PROCESS FOR MOVING PLANTS UNDER UNCERTAIN OBSERVATION ERRORS DEPENDENT ON STATES, IFAC Proceedings Volumes,35,1-6, http://https://doi.org/10.3182/20020721-6-ES-1901.00411

[26]C. Chen, J. Twycross and J. M. Garibaldi (2017), A new accuracy measure based on bounded relative error for time series forecasting, PLOS ONE,12,e0174202, http://10.1371/journal.pone.0174202

[27]C. Stanley and D. Lawie (2007), Average Relative Error in Geochemical Determinations: Clarification, Calculation, and a Plea for Consistency, Exploration and Mining Geology - EXPLORATION MINING GEOLOGY,16,267-275, http://10.2113/gsemg.16.3-4.267

[28]J. Li, M. Humphrey, Y. W. Cheah, Y. Ryu, D. Agarwal, K. Jackson and C. v. Ingen, "Fault Tolerance and Scaling in e-Science Cloud Applications: Observations from the Continuing Development of MODISAzure," in 2010 IEEE Sixth International Conference on e-Science, 2010, vol., no., p.^pp. 246-253.

[29]Z. Liao, T. Yang, X. Lu and H. Wang, "An Algorithm for Uncertain Data Reconciliation in Process Industry," in 2009 WRI World Congress on Computer Science and Information Engineering, 2009, vol. 5, no., p.^pp. 225-229.

[30]X. Zhao, S. Garg, C. Queiroz and R. Buyya, "Chapter 11 - A Taxonomy and Survey of Stream Processing Systems," (in Journal, Type of Article vol. no. Issue, p.^pp. 183-206, 2017, Art. no. Article.

[31]F. Carcillo, A. Dal Pozzolo, Y.-A. Le Borgne, O. Caelen, Y. Mazzer and G. Bontempi (2018), SCARFF: A scalable framework for streaming credit card fraud detection with spark, Information Fusion,41,182-194, http://https://doi.org/10.1016/j.inffus.2017.09.005

[32]R. Wiatr, R. Słota and J. Kitowski (2018), Optimising Kafka for stream processing in latency sensitive systems, Procedia Computer Science,136,99-108, http://https://doi.org/10.1016/j.procs.2018.08.242