ISSN: 0974-5823 Vol. 7 No. 1 January, 2022

International Journal of Mechanical Engineering

The Disruptive Approach in Message Discovery and Repository for Solving the Communication and Responsibility Issues

Kari Venkatram
School of Computer Science and Engineering, VIT University, Vellore, Tamil Nadu, India.

Geetha Mary A
School of Computer Science and Engineering, VIT University, Vellore, Tamil Nadu, India.

Abstract - Issues such as data syncup, improper schema, data replication etcs are concerns in data pipleline. Due to demand for messages exchange through data pipelines, it has became a very prominent in the enterprise ecosystem. Although several frameworks evolved, they often ignore the certain quality attributes such as reliability, consistancey, accuracy, precision etc., Due to the lack of good framework to address communication, and responsibility issues in data pipeline, serious problems such as inconsistency and incompleteness have been arising. Therefore, the proposed work presents an ingenious framework that can address the above-mentioned issues with the help of a conceptual framework called Message Description, Discovery, and Registry (MDDR). The information retrieval techniques used to evaluate the results and results show that the proposed model is better and will help in resolving communication and ownership issues in data pipelines. This proposal can significantly improve the discovery process with any kind of matching algorithm such as the Fusion semantic registry to improve the end-user service experience when discovering large-scale messages in competition with any state-of-the-art solutions of matching message algorithms.

Index Terms - MDDR, Data pipeline, message queue, data sync up, schema registry

INTRODUCTION

In this current era of big data, an enormous amount of data has been produced by varieties of devices in different forms (variety) at a high pace (velocity) in very huge quantities (volume)(Venkatram & Geetha, 2017). The advantage of big data has not only brought the opportunities but also a lot of challenges. The distributed computing system amplified the data growth with the high velocity of data from both producing and consuming side. Real-time processing, which means analyzing the data as it arrives, is the key to success as it provides a real-time decision. The delay in transmission or inaccurate data would lead to lots of data inconsistencies and produces adverse results. To address these data sync up issues, there are many data pipeline frameworks evolved however, there are several critical issues unaddressed.

THE EVOLUTION OF DATA PIPELINES

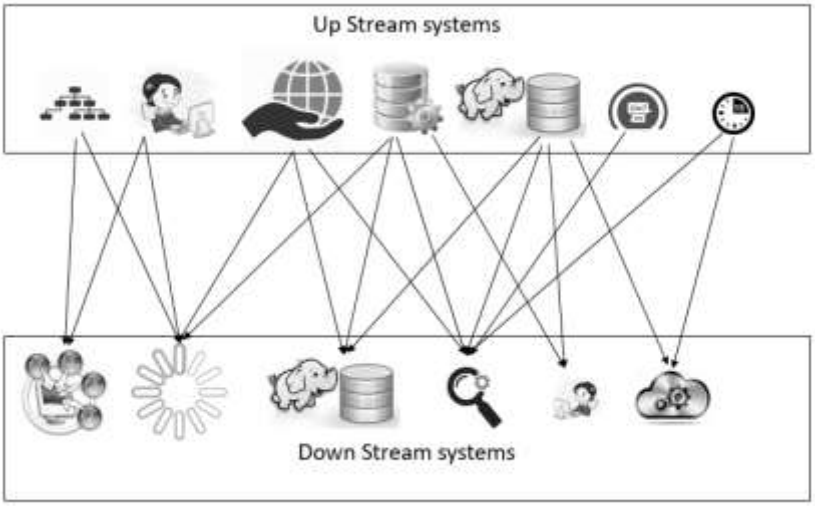
Over the period, collecting and analyzing data has been significantly changed. Data was used to store in local files via log files and then evolved to store in the servers and now with the Internet of Things(IoT) uses cloud computing data access storage(M. Wang & Zhang, 2020). Now the modern systems are capable of tracking and learning the data insights with the help of machine learning and artificial intelligence almost in real-time(Ait Hammou, Ait Lahcen, & Mouline, 2020). The more recent approaches for building data pipelines enables real-time analytics(Indrakumari, Poongodi, Suresh, & Balamurugan, 2020) on the data being processed.

The data pipeline is a process of replicating data from one system to other systems more precisely geographically and functionally distributed data stores (Warhade, Dahiwale, & Raghuwanshi, 2016). Data pipeline plays a vital role in any organization(Santiago-Duran et al., 2020) because they maintain their business data using different applications and store them in different database servers across the global locations. To meet the business demands, organizations need to have a consolidated view of their data collected from different data sources. This consolidation and data sync up amongst the systems can happen through the data pipeline.

Formerly most of the organizations used to follow linear data pipelines for replicating the data from one system to other systems. The following Fig.1 depicts how organizations build their data pipelines with a separate connection among any two or more systems where there is a need for data sync up. For instance in the below diagram hierarchal data, user interface data, application data, etc. have created numerous independent data connections to their downstream systems(Sumbaly, Kreps, & Shah, 2013).

From the below diagram, one can easily understand that the architecture used for the data pipeline seems to be very messy and tedious to organize, maintain, and monitor. Since there are separate connections among the systems, it is difficult to extend or enhance the functionality of the data set. For instance, if we have added a few attributes to the data set, it is required to change in multiple places to get this flow into the complete ecosystem. Besides, if we need to add a new source or destination systems we need to create separate connections and corresponding transformation logic, which is not a scalable and viable solution.

FIG. 1. TRADITIONAL DATA PIPELINES WITH SEPARATE CONNECTIONS AMONG THE SYSTEMS



Hence, it is very tedious to add any additional sources to the existing architecture. Besides, there is no way to reconcile the data and there is no proper control on the data flow. In this architecture, the payloads are in standard formats, where both upstream of the message and downstream of the message will produce and consume the same format of the data. The messages are synchronous hence if there is a very high data load it will affect the entire ecosystem and cannot respond.

Hence this model has no standard way of producing or consuming the data, this might lead to data quality issues(Tiwari, Sharma, & Swaroop, 2011) such as data inconsistency, latency, dirty, in-completeness, unavailability due to transmission errors. Since there are many challenges associated with this traditional model, there is a need for a better solution for the data pipeline. Subsequently, a variety of data pipeline architectures and frameworks have evolved some of them are discussed in the next session.

2.1 Evolution of Centralized Messaging System/ Data pipelines

To overcome the problems in the traditional architecture, experts started logging their data messages into a central messaging queue. In this approach, multiple consumers can consume the data from this queue. This idea is to avoid creating multiple connections and thereby to improve in the scalability. This is the stepping-stone for the evolution of centralized messaging systems. The responsibility of the messaging system (message queue) is liable for transferring data from one system to another system (Zhang, Chen, Kim, & Lei, 2011). With the implementation of message queues, online transaction processing (OLTP) can mainly focus on data generation and processing rather than data transmission issues.

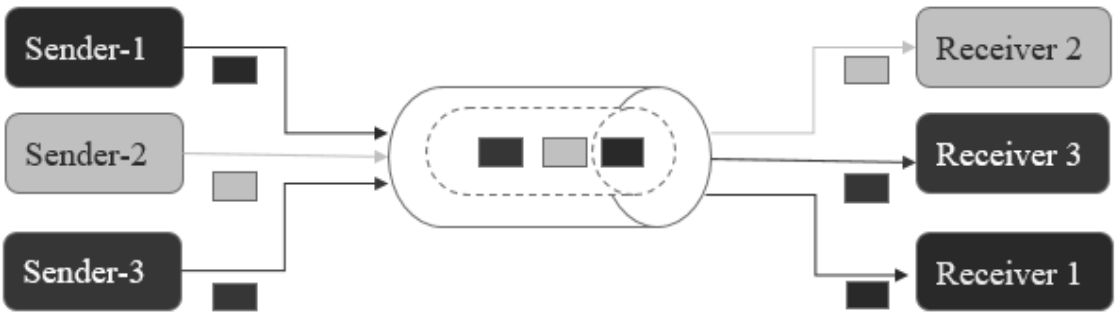
Messaging systems broadly classified into two patterns or types that are 1) Point to point messaging pattern 2) Pub – sub-messaging pattern

2.2 Point-to-Point messaging pattern (P2P pattern)

As shown in Fig. 2, in P2P messaging pattern, all messages are persisted in a queue and one or more consumers will consume this data through the queues. In this approach, a particular message is delivered or consumed by only a specific consumer (Yang, Ye, Zhang, & Xing, 2014). After a consumer consumes the particular message, the particular message deleted from the message queue.

Some of the typical examples for the P2P messaging pattern (Monson-Haefel & Chappell, 2000) is order processing queue. In this queue, each order will be generated by message sender very specific to message receiver even though the same message is required for multiple receivers there will be separate messages generated for individual receivers, however, all the orders will be in the queue and work together well.

FIG. 2. POINT TO POINT MESSAGE QUEUE



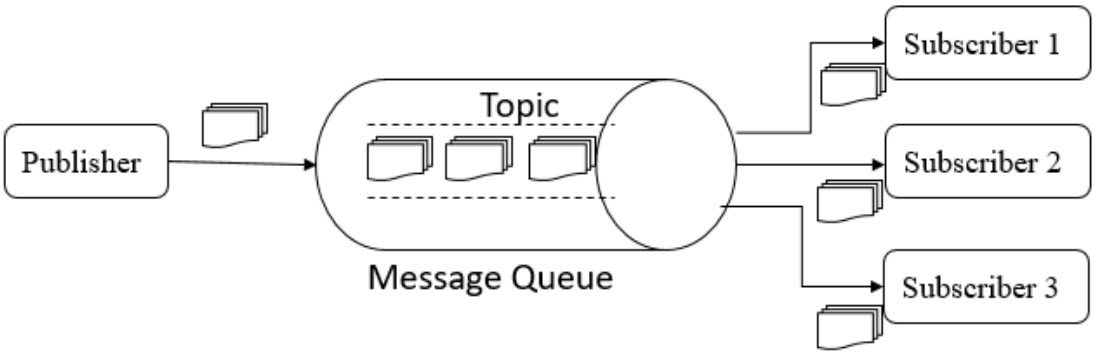
This kind of pattern is not suitable for the issue we discussed above, as there are a set of consumers who are required to consume the same message of one application (Xie, Zhang, Li, & Shi, 2012), such as pushing change events to multiple downstream from the source system. In this architecture, the payloads are very specific to the sender vs receiver. Also, the sender and receiver of the message use common schema for payloads and the message is asynchronous. `Even though the same message to be consumed by multiple receivers it requires duplicating the messages in the queue for each consumer. Hence, it will influence the overall performance.

2.3 Publish and Subscribe pattern (Pub-Sub pattern)

As depicted in Fig.3, publish-subscribe data messaging system, or pattern managed by persisting all the messages in a queue called a topic. In contrast with the P2P messaging system, consumers or subscribers can consume messages from multiple topics by subscribing to one or more topics (Monson-Haefel & Chappell, 2000). Besides, the consumer can consume all the messages on the topic. Distributed message brokers enable them to communicate their messages asynchronously within their ecosystem using a paradigm called the publish-subscribe model(John & Liu, 2017). In this pattern three important components such as 1) publisher, 2) subscriber, and 3) brokers (Xie et al., 2012) presented. Data producers are called publishers, data consumers are called subscribers and the brokers are the actual nodes where the data is stored (Ivan & Dadarlat, 2011) while it is in transit.

A typical and real-life example for the pub-sub pattern is direct to home (DTH) TV where the publisher produces or publish the messages as a form of channels such as sports channels, movie channels or music channels, etc. and a subscriber of DTH can subscribe to these messages or channels of their interest.

FIG. 3. PUBLISHER – SUBSCRIBER (PUB-SUB) MESSAGE QUEUE



In this architecture, the payloads are very generic as a producer can produce multiple messages and a subscriber can consume them as per their need. These entire messages are asynchronous. If there is any huge data load in the system, still it will perform better compared to prior architecture, as it is not required to duplicate the messages as compared to P2P architecture. Hence, this is a better approach to deal with the scenario illustrated in the earlier sections.

2.4 Data pipeline - Hybrid model

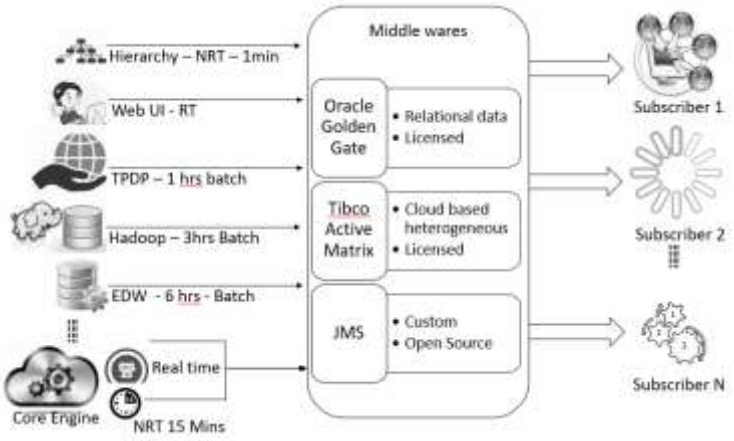
Although the pub-sub data pipeline model is one of the efficient models, it alone is not sufficient to handle all the required scenarios for an organization without any customization. Therefore, there are some hybrid approaches generally used in the industry. Organizations use multiple commercially available tools and frameworks to build their data pipelines instead of building their own data pipelines in-house as mentioned in the above section. There are several commercial tools and frameworks available for asynchronous data movement. Below are the few most popular tools or frameworks that are used in the industry. Oracle Golden Gate: A very common tool used for data integration and virtualization (Milosevic, Chen, Berry, & Rabhi, 2016). This tool enables real-time data integration and replication among the relational database. TIBCO ActiveMatrix: A service grid and comprehensive platform for data pipelines for heterogeneous systems (Milosevic et al., 2016), which is a cloud-based servicing model.

IBM WebSphere MQ: A distributed messaging queue to move the data across the systems. Gives atomic transaction support and it will be a processing overhead (Sadooghi et al., 2015). Apart from these frameworks, many open-source frameworks like JMS, Apache Kafka, and Active MQ, etc., requires a good amount of customization as per the need.

This section deals with how the existing architecture used in the industry enables the data pipeline for their message communications. This architecture based on a centralized messaging system to counter the issues of infeasible messy or decentralized solutions. This hybrid approach has a middleware of multiple message queues each used for the unique requirement. The data changes are pipelined through a centralized service bus - message-oriented middleware (Aizstrauts, Ginters, Baltruks, & Gusev, 2015) (MOM) and the same data changes are pushed to the down streams. The disadvantage of having multiple connections across the systems was avoided in this design. This can be scaled up to some extent than the previous design. Middleware consists of Oracle GoldenGate, Tibco services, etc. Oracle GoldenGate is used for data sync up or replication among the relational database; it uses table-to-table transformation among source and destinations. Many times messages are not structured data messages alone, in the current service-oriented architecture, microservice architecture, the data consumed through some services. In such scenarios, GoldenGate is not a suitable solution, and services such as Tibco services are used to address these scenarios. Tibco services are used to perform some pre-processing and filtering in the services and support multiple service

input/ output formats such as XML and JSON. Besides, most of the organizations use their custom implemented solutions such as JMS based message queues, etc. along with other middleware buses.
Below an example explains how a master item in Master Data Management (Customers, Locations, Persons, Employees, Organizations, Suppliers, Items, and Services, etc. in the master data management (Vilminko-Heikkinen & Pekkola, 2013)) called customer data is flown in the ecosystem. The customer hierarchy changes happen in a near real-time manner, which runs every minute will be flown to downstream subscribers through Oracle GoldenGate as these are relational database changes. The subscribers through a web service call will be updating a change in the WebUI (real-time). Some of the TPDP (third party data providers), data warehouse data, etc., would require some custom transformations, and messaging would go through custom message queues.

FIG. 4. CENTRALIZED DATA PIPELINE WITH MIDDLEWARE TOOLS - A HYBRID MODEL



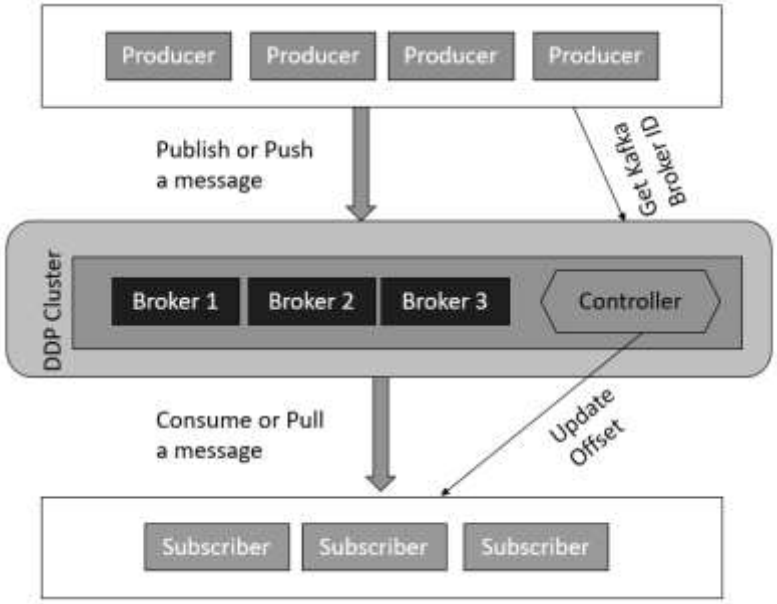
Though this architecture addressing the challenges discussed in the last section, there are few more challenges in both the publisher and consumer side. There are three major concerns in this architecture, they are 1) message standardization/ schema definition 2) monitoring, auditing the message 3) reconciliation, and re-transmission is not present in the architecture. Another aspect is the licensing cost used in this hybrid approach. The article focuses majorly on message standardization issues and how messages can be described discovered, stored securely.

2.5 Distributed data pipeline frameworks and comparison

Distributed data pipelines (DDP) are built with the concept of reliable message queuing(Z. Wang et al., 2015). Messages will be queued asynchronously between client applications and messaging systems. DDP has four components such as producer, consumer, topic and broker (Apache)
As depicted in Fig. 5, Let consider Apache Kafka as distributed data pipeline framework(Apache). Each producer will create a topic for their data in the JSON form of messages. The velocity of data messages may differ, as there are real-time, the stream of data, near real-time and batch data generators in the system. These JSON messages are pushed into a specific topic of its own within a partition in the cluster. There are four major components in distributed data pipelines such as topic, producer, consumer, and a broker.

Topic: Distributed data pipeline such holds it fed of messages in the form of Topics
Producer: A system, which processes and publishes the messages to a Distributed data pipeline topic called producer (Vuppalapati, Ilapakurti, & Kedari, 2016). The producer will generate messages (push the message) to be consumed by consumers or subscribers
Consumer: A system that processes and subscribes to one or more topics (pull the messages) which were published by producers, is called consumer (Vuppalapati et al., 2016). A consumer can consume one or more messages from one or more topics(Computing et al., 2017).
Broker: A cluster of nodes/ systems in a distributed environment is called a broker (Venkataraman et al., 2017). It receives messages from producers and persists them later dispatches these messages as and when required by consumers as a form of the pull request.
As shown in Fig. 5, The cluster (Kafka) will interact with two external entities called producer and subscriber. As illustrated four different producers are publishing the messages to four different topics and those will be consumed by three consumers in the cluster. There are three nodes called brokers, which will contain the topics. There will be one or more node controllers called zookeeper instances to monitor the cluster, which will give the broker id to the producer to publish a message in a broker. After publishing the message offset will be updated with the current offset in the broker, the same will be shared with the consumer.

FIG. 5. COMPONENTS OF THE DISTRIBUTED DATA PIPELINE



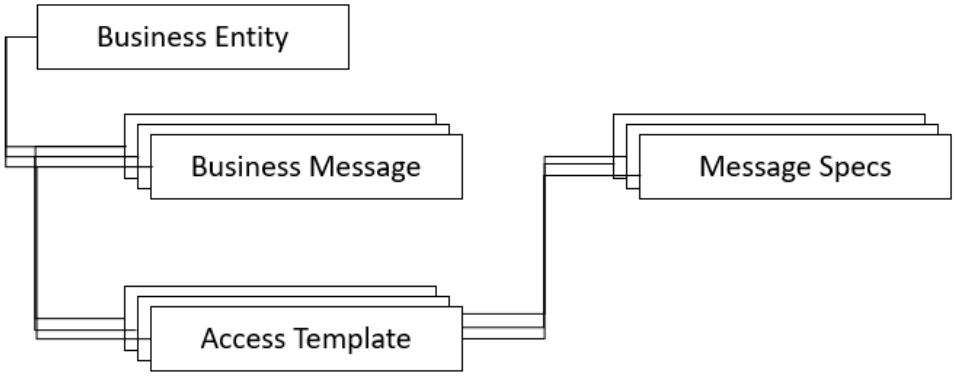
Jay Kreps(Kreps, Narkhede, & Rao, 2011) et al. compared Kafka with other famous messaging systems such as ActiveMQ and RabbitMQ from both producer and consumer standpoint. In both cases, Kafka seems to be given superior throughput compared to both of them. Hence, Kafka is one of the best-distributed data pipeline framework among the currently available frameworks. It is distributed, partitioned, and having replicated to commit log services(John & Liu, 2017). It is a Producer-Consumer patterned tool, where producers will publish messages to a Kafka topic, and consumers will subscribe to topics and consume the messages(Barba-González, Nebro, Benítez-Hidalgo, García-Nieto, & Aldana-Montes, 2020). However, only the Kafka framework cannot resolve all the issues mentioned above and is detailed in the below sections. Recently proposed a gearbox encapsulated pipeline model for processing large volumes of data into virtual containers(Santiago-Duran et al., 2020). The Message Queue Telemetry Transport (MQTT)(Elemam, Bahaa-Eldin, Shaker, & Sobh, 2020) is another platform for distributed message, especially for IoT systems. MQTT protocol is aim is to offer a light communication model based on a publication/subscription pattern(Garcia, Montalvo-Lopez, & Garcia, 2020). Advanced Message Queuing Protocol (AMQP)(Dalkıran, Önel, Topçu, & Demir, 2020) is an open standard protocol ensuring interoperability among data pipelines.

One of the key challenges in business data pipeline today IoT business models is the ability to dynamically discover the messages that the business needed, and consume them to carry out their business processes. If there is no proper description, discovery, and registry process in the data pipeline, the consumer may expect different schema, from what producers have defined. As two schemas are different and are not compatible with each other. This leads to confusion among the consumer group and messages will be ruined. Apache foundation has come up with the Schema registry under the name of Avro(https://avro.apache.org/docs/current/, 02/12/2020 20:59:42) a few years ago. The Avro enables the RESTful interface for managing their schemas(D. TEAM, 2020) also permits to store the schemas in the registry. Avro has addressed the major issue in the previous data pipelines of schema definitions. There are major chances of having different schemas between consumers and producers. Now, this can bring the schema confirmed to the requirement and avoid the communication challenges(c. i. team, 2019). While the AVRO schema only contains syntactic information, it can be used to convey information(Holom, Rafetseder, Kritzinger, & Sehrschön, 2020). There is no specific schema validation found in the available tools in the industry. Schema registries in the message queue literature broadly depend on a distributed registry environment but as far as literature found in this domain the idea of using a functional semantic description model in their schema registries are missing. In addition, discovery process of messages and secured message consumption were not available in the currently available data pipelines. Hence, there is a need for a better solution for description, discovery process and hold them in a registry of the metadata. Also, security to be provided for the messaging system.

THE PROPOSED SOLUTION

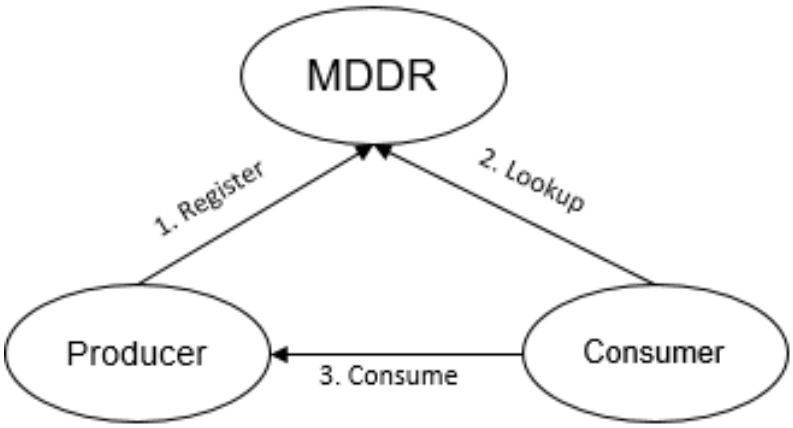
To address these communication and responsibility issues in message communication described in the above sections, proposed a disruptive model for message description, discovery, and repository called MDDR in the similarity of UDDI defines a standardized model for service registry(Juric, Sasa, Brumen, & Rozman, 2009). MDDS defines the information model, message repository, and message provider's API for the data pipeline message consumers. Besides, it will have API for consuming, publishing, editing, and deleting messages. Apart from the capabilities of UDDI, it has an additional feature, which will have a flexible schema definition framework. Also added semantic search and quality-aware matching in this proposal.

FIG. 6. MESSAGE DESCRIPTION, DISCOVERY AND REPOSITORY DATA STRUCTURE



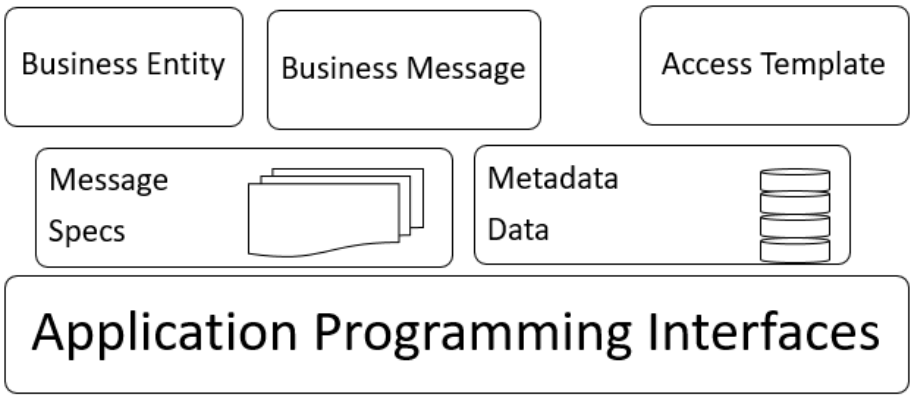
The MDDR data structure defined in the above Fig 6, the message schema defined through a JSON based schema. It contains mainly four core components of the data structure. They are 1) Business Entity - Message producer for a business entity. This business entity producer can be from multiple producer groups for the same business entity. 2) Business Message - Message being produced by different producers of the business entity, this is a consolidated message attributes among all the business entity producer., 3) Access Template gives access information about the message such as how the business message can be accessed and other credentials, etc., and 4) Message Specs are specifications of API for searching and publishing messages in MDDR.

FIG. 7. MESSAGE DESCRIPTION, DISCOVERY, AND REPOSITORY- SERVICING MODEL.



The MDDR message service model described in the above Fig 7. Producers will register their message schema into MDDR. The consumers will look up in MDDR for the required message and download the message schema from MDDR after that with that schema definition message consumed from the producer by the consumer. We propose MDDR must have a minimum one logical node to handle both its metadata and data. A complete set of nodes in an MDDR should jointly define a well-defined schema for its consumers. The publishers and consumers of MDDR use the registration process with the help of predefined APIs to access the MDDR. Message publishers, the organizations/ departments who wish to list business messages on the MDDR will use it to publish and describe their message. Message consumers, clients who are looking for the messages to consume, will discover the message and download schema from the MDDR and use the information to locate and work with the messages.

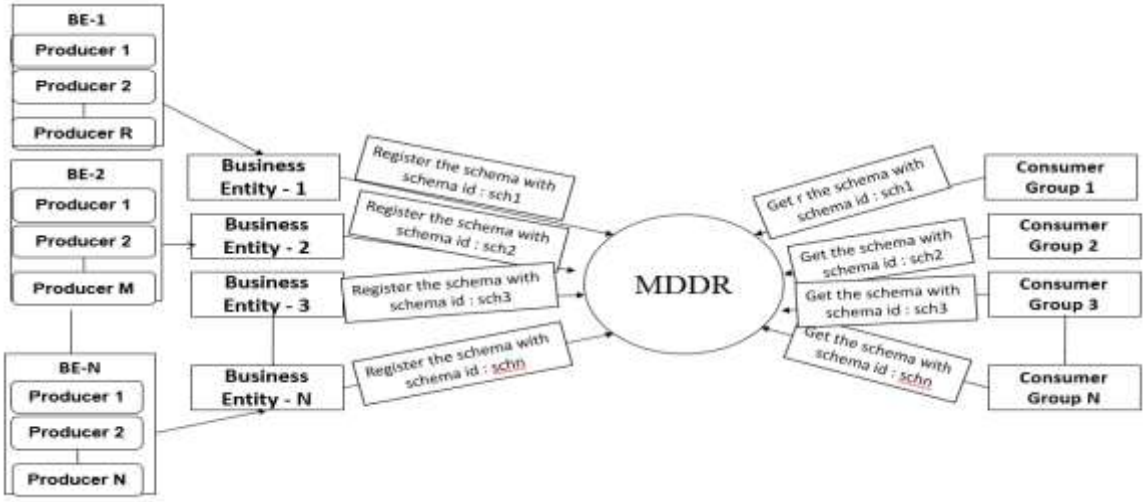
FIG. 8. MESSAGE DESCRIPTION, DISCOVERY AND REPOSITORY STACK



MDDR stack contains majorly four components from the technical standpoint:
1) The MDD Registry. 2) The MDDR Specifications. 3) The APIs (Application Programming Interfaces) to publish and consume the message. Message discovery services. 4) The metadata (Schema definitions), messages that will be registered in MDDR

As depicted in the above Fig 8, MDD Registry is a schema registry stores tuples consisting of local name and version of schema and schema format, typically JSON that describes the binary form of the data. Let's business entities to describe themselves and their business messages, provide instructions consuming their messages. Search for messages and get the schema to consume the messages. Access Template enables consumer access to privileged users and provides access permission to consumers. The MDDR's data, metadata, and other details to describe how to locate and invoke them. The specification provides detailed instructions for operating on MDDR. The APIs provide client-side tools for publishing, deleting, managing, and querying MDDR.
How to deploy and publish messages using MDDR: Before publishing a business message using MDDR, the message should be ready to publish. First, the message has to be registered by checking if there is any business entity available in the MDDR if so get the schema and if the schema is having all the attributes the message requires it's fine otherwise it has to be added to the schema and check the metadata in and register the message metadata. Then the message service published for any consumer to consume it. Please make sure that the additional attributes are added in the message should be defined as a default value so that it will not affect the consumers not consuming this additional attribute.
How to discover and consume a message using MDDR: Before consuming the message, the consumer has to discover the message in MDDR and get the schema of the message. Check the message access template for the credentials if any. Based on the access availability message-consuming request will be submitted to the producer and consumed accordingly.
Flexible Schema definition: MDDR Flexi schema definition suggests enables to add a new column to message schema without affecting existing application consumers, this is possible with a default to a schema for that particular new column. At the same time, it is possible to remove a column that had a default value from the message. In addition to this, it is possible to order the column without any impact on consumers. Moreover, it should allow us to modify the default value of the column and permit to add default value if it has not defaulted.
Matching Algorithms in the discovery process in MDDR: There are two types of discovery processes that are present in the MDDR discovery module. Behavior was aware of semantic matching: When producers are registering their messages, the semantic descriptors are be submitted to MDDR. The discovery algorithm for matching any parts included in a message functional description model shall compute the semantic similarity of two targets. For the sake of behavior-aware, we focus on concept similarity and predicate similarity.
Adding semantics to MDDR framework: Augmenting MDDR with semantic matchmaking capabilities that facilitate accurately and message discovery process will constitute a significant advancement as compared to the existing publication and discovery facilities offered in conventional registries in message or data pipeline systems.
To support the concept-driven discovery of message services descriptions, the FUSION Semantic Registry enhances the purely syntactic search facilities that an MDDR can offer. This is achieved with addition to MDDR APIs, Paolucci et has proposed Semantic Matching of Web Services Capabilities (Paolucci, Kawamura, Payne, & Sycara, 2002) and Kunal Verma et has proposed enhanced semantic match for UDDI and the same manner it can be applied to MDDR as well. As a result, the MDDR server module remains semantics-agnostic, providing FUSION(Kourtesis & Paraskakis, 2008) system implementers with UDDR vendor independence.
A centralized data governance model has been proposed as part of this solution. Data governance ensures data security, accountability, and transferability. Any producer publishing any new messages to the event-streaming platform will have a schema defined and confirmed so that all consumers can understand at any point time. Any new types of messages should conform to business policies, such as a prohibition on Intellectual properties, personally identifiable information (PII), etc., All clients connecting to the cluster use the latest versions of the schema to avoid extra costs for protocol upgrade at the server-side. Also, enforce data governance policies in a unique place. Suggested to have them inside the event-streaming platform, as to avoid audit each consumer adhering to the rules defined. Especially in bigger organizations with many applications that are leveraging the platform to build their real-time data pipelines with specific business logic, enforce such data governance policies is extremely difficult. Hence, it is advised to have a centralized data governance model within the cluster.
Benefits of MDDR in message queues in detail: MDDR provides a central location to agree upon schemas across all your producers and consumer applications. This simplifies the message consumption process very drastically. It allows the producers to unify schemas across business entities which a challenging task, however, a well-designed MDDR framework allows you to support a variety of mutually compatible schemas for a single logical topic and still enjoy the benefits of a unified reporting pipeline. The message streams should be serialized and consumers of the messages are assured to be able to de-serialize makes your data pipelines much more robust. Changes to the data can require very painful re-processing of stored historical data. With flexible schema evolution/ definition, where the date schema would have evolved in a backward-compatible manner. Having a central schema registry helps with message discovery, and allow. This MDDR and the consistent use of schemas enable us to invest in pipelines across the business entities that provide automated tooling around your data. MDDR is will be a good device to enforce policies for your data such as preventing newer schema versions of a data source to break compatibility with existing versions. As illustrated in Fig. 9 each producer belongs to one application will define application schema and register it in the schema registry. As per publication and discovery the process for web services using Universal Description, Discovery, and Integration (UDDI) (Villalba, Pérez, Carrera, Pedrinaci, & Panziera, 2015), introduced similar concepts such as schema registry for message description and discovery. There can be multiple producers for each business entity and each business entity will register a unique schema for all the producers of the same business entity. They store the schema id and schema details or descriptions for each business entity. This schema can be discovered by a few attributes by consumers.

FIG. 9. MDDR – INTEGRATION



On the other hand, from the consumer group side, all the consumers of one group will consume the same schema of the business entity. Before they consume they query schema registry and get the schema details based on the schema id or any indexed attributes of the schema. A message description will be discovered from the repository. From the schema details, the consumer can understand the data format and consume the data from the pipeline

As it enables a unified model for schema definition, the schema registry can ensure the data quality for data cleanliness and completeness as per the schema definition. The producer has to make sure that, the data being pushed into the queue should be as per the schema definition. Hence, the entire ecosystem will be with high quality of the data.

Another aspect in the schema definition is schema evaluation, which is essentially making the schema flexible to cater to future changes. Nowadays data is morphing and hence there is a need for adding additional attributes to the data more often. Therefore, if there are any additional attributes added to data, the schema definition will change and it may affect the existing consumers until they change the logic of transformation. We are proposing that the schema should be defined as backward-compatible format, which means if there is a change in the schema at the producer, without changing at consumer code downstream systems will continue to work with existing old logic for transformation and the system will not break. This will also address the separation of concern issues of the ETL process of a warehouse.

Future work: Since data is produced at a faster rate and to avoid the data latency issues (Milosevic et al., 2016) and fault tolerance, there is a need for parallel, distributed, and fault-tolerant architecture for data pipeline. Besides, as discussed in the earlier sections there are few more issues concerning message auditing, monitoring, and controlling other important attributes for data quality improvement. Besides these data reliability aspects such as reconciliation and retransmission to be considered to have a better and accurate data pipeline. These two aspects will be considered in our future work.

RESULTS AND DISCUSSIONS

The evaluation of the proposed framework can be performed with the help of assessment criteria mentioned below:

Assessment criteria: The evaluation of message discovery can be viewed as similar to the evaluation of the information retrieval system. The metrics used in this field can be used for evaluating the proposed solution The metrics for evaluation include precision, recall, and Unique relevancy recall (URR).

MDDR can be evaluated as per the evaluation process used in the UDDI for web services discovery. The QoS of Schema Registry will be evaluated based on the following assessment consists of metrics such as precision, recall, fallout, and Unique Relevance Recall (URR)(Sambasivam, Amudhavel, Vengattaraman, & Dhavachelvan, 2018). Precision and Recall are very extensively used in message discovery analysis. These two are major metrics in information retrieval for evaluating the performance of the solution.

Precision is the number of messages discovered that are relevant and Recall is the number of relevant messages that are discovered.

Precision(Sambasivam et al., 2018): In this case, precision (p) is defined as the fraction of retrieved message(Lizarralde, Mateos, Zunino, Majchrzak, & Grønli, 2020) structure relevant to the subscriber's request.

$$\text{Precision (p)} = \frac{M_r \cap M_s}{M_r} \quad (1)$$

Where M_r is the number of messages that are relevant to the request

M_s is the number of messages that are retrieved.

Recall: Recall is defined as the fraction of relevant messages that are retrieved(Lizarralde et al., 2020) to the subscriber's request. This is a true positive scenario.

$$\text{Recall (r)} = \frac{M_r \cap M_s}{M_r} \quad (2)$$

Fallout: Fallout is defined as the fraction of non-relevant messages that are retrieved w.r.t total non-relevant messages retrieved(Sambasivam et al., 2018).

$$\text{Fallout (f)} = \frac{M_{nr}}{M_{TNR}} \quad (3)$$

Where M_{nr} are non- relevant messages received
M_{tnr} is a total of non-relevant messages received
Unique Relevant Recall (URR): URR is defined as the fraction of the number of messages that are unique relevant w.r.t the number of messages that are relevantly retrieved(Sambasivam et al., 2018).

$$\text{Unique Relevant Recall (URR)} = \frac{\text{NM}_{\text{ur}}}{\text{NM}_{\text{r}}} \quad (4)$$

Where NM_{ur} is the number of uniquely relevant messages.
NM_r is the number of messages that are relevantly received.
The F-Measure/ F-Score: F-measure is a measure of a test's accuracy and it combines the previous two metrics precision and recall using the harmonic mean(Lizarralde et al., 2020).

$$\text{F-Score} = 2* \frac{\text{Precision}*\text{Recall}}{\text{Precision}+\text{Recall}} \quad (5)$$

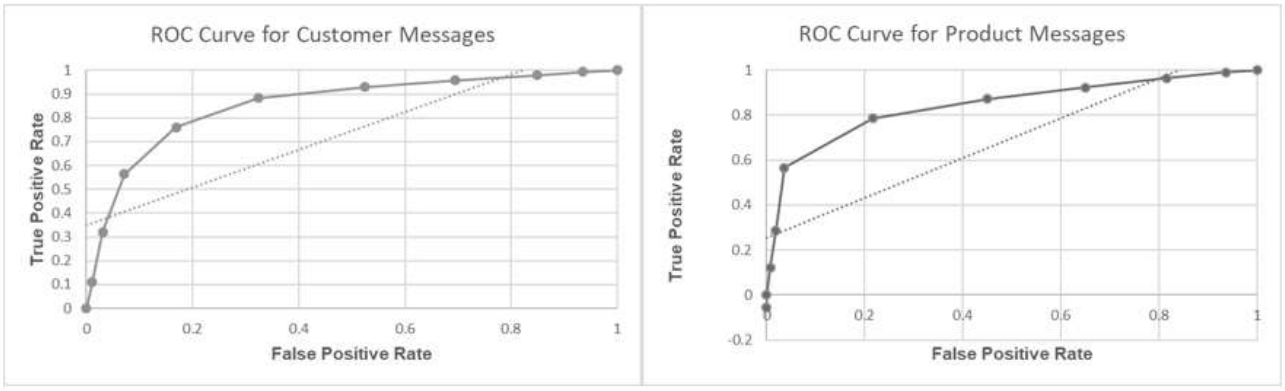
For different 10 types of messages a sample measures of Precision, Recall, Fallout, and F-Scores are depicted as given below Table1. As observed in the data table precision is very high. This means the relevant messages discovered are between 93 and 98%, which is a very good w.r.t message discovery process. Also, the recall of the messages is very good and is between 95% and 99%, which means the number of relevant messages that are discovered is very high and in acceptable condition. F-Score/ F-Measure is a test accuracy measure and as per the below data it between 94% and 98% which meaning perfect precision and recall, at a value of 1. Also, the fallout is very less of close to 0%, which means the non-relevant messages received are very less. With this measure, MDDR has been evaluated with very high precision, recall, and F-Score moreover with very low fallout. It means the quality of service for the MDDR discovery process is excellent and can be considered for the discovery process.

TABLE 1: MESSAGE RETRIEVAL METRICS.

Message Group	Precision	Recall	Fallout	F-Score
Customer	0.9302	0.9524	0.1200	0.9412
Department	0.9673	0.9823	0.0423	0.9748
Distributor	0.9742	0.9875	0.0337	0.9808
Product	0.9751	0.9855	0.0326	0.9803
Service	0.9722	0.9873	0.0363	0.9797
Vendor	0.9692	0.9857	0.0403	0.9774
Partner	0.9727	0.9865	0.0385	0.9796
Location	0.9727	0.9865	0.0384	0.9795
Employee	0.9726	0.9861	0.0417	0.9793
Student	0.9783	0.9888	0.0422	0.9835

Also, another metric receiver operating characteristic curve (ROC) and Area under Curve (AUC) depicted in the following Fig. 10. ROC is a plot of values of the False Positive Rate (FPR) versus the True Positive Rate (TPR) for a specified cutoff value. Created the cumulative values for Failure and Success scenarios and then the values of FPR and TPR. The ROC curve is a step function with the data points shown in Fig.10 Also it is observed that the higher the ROC curve which is the y-axis being very closer (i.e., y = 1) the outstanding fit.
The area under the curve (AUC) is a sum of areas of each of the rectangles in the step function. The purpose of this AUC formula represents the quality of information retrieval. If closer AUC to 1 the better the fit. In Fig 10. The formula for calculating the AUC is a calculated value of .889515 excellent fit. In general, an AUC of 0.5 suggests no discrimination, 0.7 to 0.8 is considered acceptable range, 0.8 to 0.9 is considered excellent, and more than 0.9 is considered outstanding.

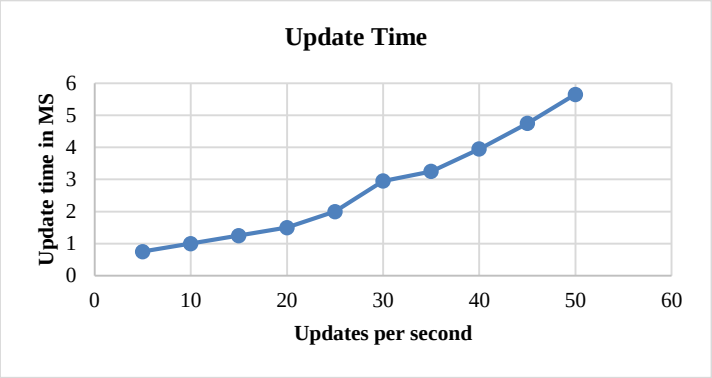
FIG. 10: ROC CURVE FOR CUSTOMER MESSAGE AND PRODUCT MESSAGES



As a test of the MDDR client experience, we measured the time required to perform an update message upon the resource information stored in MDDR as the number of simultaneous update operations increased. The average time required to perform a binding Template and update the message on the MDDR was very linearly increasing. The function of the update rate seems to be

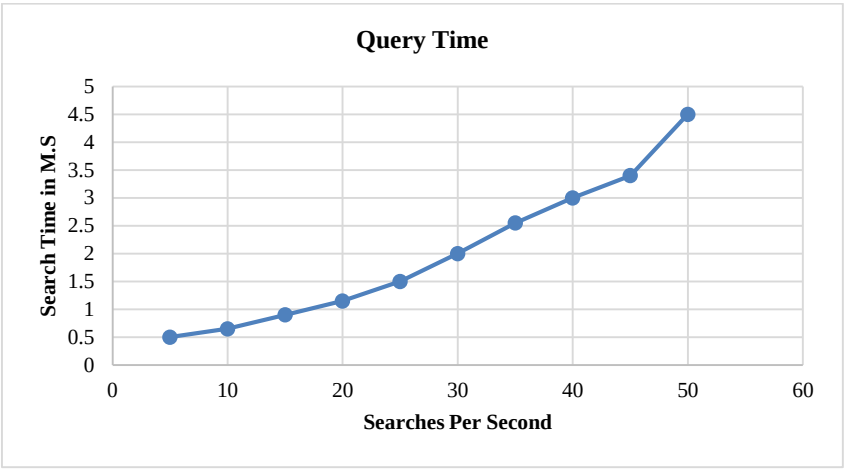
scalable from this below Fig. 11, as up to 10 messages being updated in less than a millisecond, 10 -20 messages are taking an average of 1.25 millisecond and 40-50 simultaneous messages are getting updated in 5.75 milliseconds. So this is scalable and can be considered.

FIG. 11: UPDATE TIME AS A FUNCTION OF UPDATE RATE



As a test of the MDDR client experience, we measured the time required to perform a search upon the resource information stored in MDDR as the number of simultaneous searches increased. The average time required to perform binding Template searches on the MDDR registry was measured using a standard match algorithm. The function of the search rate seems to be scalable from this below Fig. 12, as up to 10 messages being updated in less than 0.5 milliseconds, 10 -20 messages are taking an average of o.75 milliseconds to search and 40-50 messages can search within an average of 3.5 milliseconds. Therefore, this is scalable and can be considered.

FIG. 12: SEARCH TIME AS A FUNCTION OF UPDATE RATE



After analyzing these results the MDDR approach will be scaled up to discover the messages very well. Also, the message update process also scalable to consider. The precision, recall of the message discovery is very high. It can discover the relevant messages. The ROC and AOC curves in an outstanding and excellent fit. This a better with better matching with the addition of FUSION Semantic Registry on top of MDDR.

CONCLUSION

This article illustrates the data pipelines and their evolution over a period to address the challenges of the legacy systems. Also discussed various architecture and frameworks present in the industry now and discussed various challenges present in the state of art frameworks. The proposed solution in this paper, we have presented the design and implementation of MDDR. Business messages can register with the MDD registry and can be discovered at the design time by message consumer. We believe this is a very useful contribution because we not only discussed the integration of MDDR technology into a message services composition framework but also demonstrated how MDDR integration is done operationally. Besides, FUSION necessitates the introduction of semantics to all aspects of the message discovery process. The evaluation of this solution is measured with precision, recall, f-score that are very high and acceptable range. ROC and AOC curves also considered excellent to outstanding. The time to search and update messages in MDDR very liner to no current messages, which is a scalable solution. The work presented in this paper is part of a larger, long-term research effort aiming at developing a framework for dynamic and peer-to-peer provisioning of Message services. Future work will focus on the validation and controller framework.

REFERENCES

[1] Ait Hammou, B., Ait Lahcen, A., & Mouline, S. (2020). Towards a real-time processing framework based on improved distributed recurrent neural network variants with fastText for social big data analytics. *Information Processing & Management, 57*(1), 102122. doi:https://doi.org/10.1016/j.ipm.2019.102122

[2] Aizstrauts, A., Ginters, E., Baltruks, M., & Gusev, M. (2015). Architecture for Distributed Simulation Environment. *Procedia Computer Science, 43*, 18-25. doi:http://dx.doi.org/10.1016/j.procs.2014.12.004

[3] Apache. Apache Kafka.

[4] Barba-González, C., Nebro, A. J., Benítez-Hidalgo, A., García-Nieto, J., & Aldana-Montes, J. F. (2020). On the design of a framework integrating an optimization engine with streaming technologies. *Future Generation Computer Systems, 107*, 538-550. doi:https://doi.org/10.1016/j.future.2020.02.020

[5] Computing, C. C., Wang, M.-H., Li, Y., Liu, Y., Zhang, S., & Zhu, P. (2017). Redesigning Kafka Message Queue System: Toward a Decentralized Stateful Broker System.

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

[7] Elemam, E., Bahaa-Eldin, A. M., Shaker, N. H., & Sobh, M. (2020). Formal verification for a PMQTT protocol. *Egyptian Informatics Journal*. doi:https://doi.org/10.1016/j.eij.2020.01.001

[8] Garcia, C. A., Montalvo-Lopez, W., & Garcia, M. V. (2020). Human-Robot Collaboration Based on Cyber-Physical Production System and MQTT. *Procedia Manufacturing, 42*, 315-321. doi:https://doi.org/10.1016/j.promfg.2020.02.088

[9] Holom, R.-M., Rafetseder, K., Kritzinger, S., & Sehrschön, H. (2020). Metadata management in a big data infrastructure. *Procedia Manufacturing, 42*, 375-382. doi:https://doi.org/10.1016/j.promfg.2020.02.060

[10] https://avro.apache.org/docs/current/. (02/12/2020 20:59:42). *Apache Avro™ 1.9.2.* Retrieved from https://avro.apache.org/

[11] Indrakumari, R., Poongodi, T., Suresh, P., & Balamurugan, B. (2020). Chapter Seven - The growing role of integrated and insightful big and real-time data analytics platforms. In P. Raj & P. Evangeline (Eds.), *Advances in Computers* (Vol. 117, pp. 165-186): Elsevier.

[12] Ivan, C., & Dadarlat, V. (2011). A tool for evaluating event based middleware. *Procedia Computer Science, 3*, 1283-1295. doi:http://dx.doi.org/10.1016/j.procs.2011.01.005

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

[14] Juric, M. B., Sasa, A., Brumen, B., & Rozman, I. (2009). WSDL and UDDI extensions for version support in web services. *Journal of Systems and Software, 82*(8), 1326-1343. doi:https://doi.org/10.1016/j.jss.2009.03.001

[15] Kourtesis, D., & Paraskakis, I. (2008). *Web service discovery in the FUSION semantic registry* (Vol. 7).

[16] Kreps, J., Narkhede, N., & Rao, J. (2011). *Kafka: A distributed messaging system for log processing*.

[17] Lizarralde, I., Mateos, C., Zunino, A., Majchrzak, T. A., & Grønli, T.-M. (2020). Discovering web services in social web service repositories using deep variational autoencoders. *Information Processing & Management, 57*(4), 102231. doi:https://doi.org/10.1016/j.ipm.2020.102231

[18] Milosevic, Z., Chen, W., Berry, A., & Rabhi, F. A. (2016). Chapter 2 - Real-Time Analytics A2 - Buyya, Rajkumar. In R. N. Calheiros & A. V. Dastjerdi (Eds.), *Big Data* (pp. 39-61): Morgan Kaufmann.

[19] Monson-Haefel, R., & Chappell, D. (2000). *Java Message Service*: O'Reilly \& Associates, Inc.

[20] Paolucci, M., Kawamura, T., Payne, T. R., & Sycara, K. (2002). *Semantic Matching of Web Services Capabilities*, Berlin, Heidelberg.

[21] Sadooghi, I., Wang, K., Patel, D., Zhao, D., Li, T., Srivastava, S., & Raicu, I. (2015, 7-10 Dec. 2015). *FaBRiQ: Leveraging Distributed Hash Tables towards Distributed Publish-Subscribe Message Queues.* Paper presented at the 2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC).

[22] Sambasivam, G., Amudhavel, J., Vengattaraman, T., & Dhavachelvan, P. (2018). An QoS based multifaceted matchmaking framework for web services discovery. *Future Computing and Informatics Journal, 3*(2), 371-383. doi:https://doi.org/10.1016/j.fcij.2018.10.007

[23] Santiago-Duran, M., Gonzalez-Compean, J. L., Brinkmann, A., Reyes-Anastacio, H. G., Carretero, J., Montella, R., & Toscano Pulido, G. (2020). A gearbox model for processing large volumes of data by using pipeline systems encapsulated into virtual containers. *Future Generation Computer Systems, 106*, 304-319. doi:https://doi.org/10.1016/j.future.2020.01.014

[24] Sumbaly, R., Kreps, J., & Shah, S. (2013). *The big data ecosystem at LinkedIn*. Paper presented at the Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, New York, New York, USA.

[25] team, c. i. (2019). Ensure Application Development Compatibility. *CONFLUENT*.

[26] TEAM, D. (2020). Kafka Schema Registry | Learn Avro Schema. *https://data-flair.training/*.

[27] Tiwari, S. K., Sharma, A., & Swaroop, V. (2011). Issues in Replicated data for Distributed Real-Time Database Systems. *International Journal of Computer Science and Information Technologies, 2*(4), 1364-1371.

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

[29] Venkatram, K., & Geetha, M. A. (2017). Review on Big Data & Analytics – Concepts, Philosophy, Process and Applications. *17*(2), 3. doi:https://doi.org/10.1515/cait-2017-0013

[30] Villalba, Á., Pérez, J. L., Carrera, D., Pedrinaci, C., & Panziera, L. (2015). servIoTicy and iServe: A Scalable Platform for Mining the IoT. *Procedia Computer Science, 52*, 1022-1027. doi:http://dx.doi.org/10.1016/j.procs.2015.05.097

[31] Vilminko-Heikkinen, R., & Pekkola, S. (2013, 7-10 Jan. 2013). *Establishing an Organization's Master Data Management Function: A Stepwise Approach.* Paper presented at the 2013 46th Hawaii International Conference on System Sciences.

[32] Vuppalapati, C., Ilapakurti, A., & Kedari, S. (2016, March 29 2016-April 1 2016). *The Role of Big Data in Creating Sense EHR, an Integrated Approach to Create Next Generation Mobile Sensor and Wearable Data Driven Electronic Health Record (EHR).* Paper presented at the 2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService).

[33] Wang, M., & Zhang, Q. (2020). Optimized data storage algorithm of IoT based on cloud computing in distributed system. *Computer Communications, 157*, 124-131. doi:https://doi.org/10.1016/j.comcom.2020.04.023

[34] Wang, Z., Dai, W., Wang, F., Deng, H., Wei, S., Zhang, X., & Liang, B. (2015, 1-3 Nov. 2015). *Kafka and Its Using in High-throughput and Reliable Message Distribution.* Paper presented at the 2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS).

[35] Warhade, S., Dahiwale, P., & Raghuwanshi, M. M. (2016). A Dynamic Data Replication in Grid System. *Procedia Computer Science, 78*, 537-543. doi:http://dx.doi.org/10.1016/j.procs.2016.02.099

[36] Xie, H., Zhang, Y., Li, X., & Shi, R. (2012). Research on Routing and Self-Configuration Ability of Topic-Based Pub/Sub System. *Procedia Engineering, 29*, 3212-3216. doi:http://dx.doi.org/10.1016/j.proeng.2012.01.468

[37] Yang, F., Ye, X., Zhang, Y., & Xing, C. (2014, 12-14 Sept. 2014). *DZMQ: A Decentralized Distributed Messaging System for Realtime Web Applications and Services.* Paper presented at the 2014 11th Web Information System and Application Conference.

[38] Zhang, Z., Chen, H., Kim, M., & Lei, H. (2011). *QORG: cloud-based queuing system with strong consistency*. Paper presented at the Proceedings of the Workshop on Posters and Demos Track, Lisbon, Portugal.