International Journal of Computer Applications (0975 – 8887)
Volume 186 – No.54, December 2024

Efficient Data Integration in Retail SCM using Inbound and Outbound Data Interfaces

Sachin More
SME in Supply Chain Management
Smyrna, GA - 30080

ABSTRACT

The modern business landscape is characterized by a diverse array of IT software systems, each serving specific functions within the organization. This has resulted in a complex network of heterogeneous systems spanning across on-premises and multi-cloud environments including multi-regional data centers. The prevalent use of microservices architecture adds to the complexity, presenting challenges in maintaining data synchronization, communication, and orchestration to ensure a cohesive operation. These systems must work in concert, akin to an orchestra, sending and receiving signals to maintain seamless data and application synchronization ensuring atomicity of the transactions.

In this Framework, we have proposed several data interface methodologies and built solutions to ensure Data is synchronized on a timely basis in preventing losses caused purely by data discrepancies and assure data quality across the enterprise application ecosystem. The intent is to save millions of dollars in losses and create avenues for growth achieved through aspects of data quality and integrity.

General Terms

Banking, Event sourcing, E-commerce, Enterprise application, Interfaces, Supply Chain retail, stock management,

Keywords

ERP, CDC, Data sync, inbound, outbound, Real time, Batch mode, Traces, PUSH, PULL, XML agents, snapshots, RESTful Web Services

1. INTRODUCTION

In today's competitive retail world, enterprises are challenged by the evolution of customer demand and macroeconomic factors due to inflation, product availability, and competitive pricing. Adding to the complexity, businesses are operating in the landscape of heterogeneous systems, as they intend to bring the best solutions to handle specific business goals. With these heterogeneous systems in place, it introduces another challenge of time-bound data synchronization and data accuracy. Every application in the retail ecosystem brings its own set of technology, platform, and database complexity making the problem even bigger. This has resulted in data discrepancies leading to inaccurate pricing and promotions, and erroneous perpetual inventory cascading its effect on replenishment impacting revenue, and customer dissatisfaction in turn affecting consumer loyalty.

Delays or inaccuracies in the data sync process can result in customer dissatisfaction and potential loss of business. Exploring various methods, techniques, and technologies for data synchronization is vital for optimizing these processes and ensuring the robust performance of enterprise systems.

This paper attempts to deep dive into several interfacing techniques and methodologies that can be adopted by enterprises to address the data inaccuracy problem at the core depending on the use case with a real-world example.

2. BUSINESS PROCESS

In a typical enterprise business setup, the ecosystem has a series of applications, which may include both internal and third-party external platforms. They all need to exchange data and information to execute the business to meet the desired goal. The business process in this ecosystem represents users carrying out certain business functions like the creation of products, definition of product sales prices, promotions, order management, etc.

Fig 1 depicts the enterprise application ecosystem for an omnichannel retail enterprise. As shown below, it is an ecosystem with a complex web of internal and external systems that operate on heavy data synchronization to maintain the business and data integrity ensuring seamless communication of data and information favorable to the business functions.

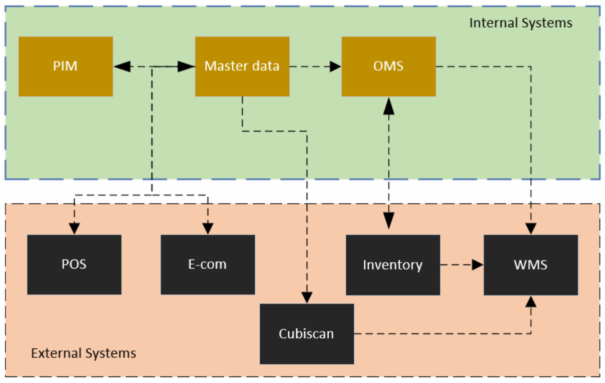

Fig 1: Enterprise ecosystem of software systems

The above fig represents the depth of complexity involved in an enterprise of the supply chain business that represents the functions from sourcing to the point of sale to inventory management.

The challenges associated with these heterogeneous systems are immense and contribute heavily to the day-to-day operations ensuring smooth execution. Incorrect information or data delays can cause revenue loss and long-term reputational challenges for the enterprise.

3. INTERFACE CONCEPTS

To address the data sync challenges, we are first defining what an interface is and delve deep into each of the types and methodologies involved.

3.1 Interface Definition

An interface is a point of interaction between two or more systems, components, or groups that allows them to communicate effectively and exchange information. Interfaces are important in technology because they allow different systems to work together seamlessly and ensure compatibility. It's an important facet of any application platform as it not only allows systems to communicate but opens avenues to do more than what a standard system can offer, simply put, it is an agent that sits between data producers and consumers. An interface can be as simple as a database table where a third-party system drops a piece of information that can be read by the consuming systems and housed in the core platform for the processes to do the next set of decision-making or business operations.

Interface is based on the event-sourcing design pattern in which changes to the state of an application are stored as a sequence of events. This approach allows the capture of all changes to an entity's state over time. In other words, event sourcing is like keeping a diary of everything that happens in a system. Event stores like Apache Kafka are like digital diaries.

For instance, a retailer can rely on a data partner or a third-party data provider like WERCS for product attributes about hazardous material or any other relevant information exchange specific to certain Product UPCs, these attributes may be used within the ERP to drive decisions onto their warehouse management systems like ASRS (Automatic storage and retrieval system). To transmit this piece of information, the third-party data provider may use a traditional Secured File Transfer Protocol (SFTP) and use a flat file format.

An Interface in Java programming language is defined as an abstract type used to specify the behavior of a class. An interface in Java is a blueprint of a behavior and contains static constants and abstract methods. A java interface is predominantly used to achieve abstraction (information hiding) and must not be confused with the inbound/outbound data interfaces [1][2].

3.1.1 Inbound interfaces

Inbound interfaces deal with the incoming data that flows into the enterprise's core business apps like CRM or ERP systems. It is read from the 3rd party system and housed in the enterprise's storage platform representing the "Source of truth" for any decision making by other enterprise services. An example of inbound data is the purchase transaction done by a customer at a physical or online store, for which a payment acknowledgement is expected from the banking system. The payment transaction is bundled, transmitted to the bank's centralized server, processed by the incoming interface, authenticated, and authorized by the approval engine and stored in the database. The bank would then send a confirmation message to the sender (store in this case), ensuring transaction closure [4].

One of the most popular use cases of inbound interfaces is data migration from legacy application platforms or databases. This process is popularly known as bootstrap or initialization mode. Interfaces help take out the complexity behind the new systems as the inbound interface layer normalizes the data onboarding process through the abstract orchestration layer. It greatly normalizes the inbound data layer without having to worry about how and what goes into the core system during the migration. Inbound interfaces just do not take care of the data itself but also the data integrity and data quality ensuring data completeness before housing it into the business applications.

Inbound interfaces are integral to various applications across multiple industries. In healthcare, for example, they are used to manage patient information, appointments, and charges, ensuring seamless data flow into the electronic health records systems. In enterprise resource planning (ERP) systems like SAP, inbound interfaces are used for integrating different business processes and systems, such as synchronizing customer data between an ERP system and a Customer Relationship Management (CRM) system or facilitating the exchange of procurement data between SAP S/4HANA and Ariba solutions. Another common use case is in Customer Relationship Management (CRM) platforms like Salesforce, where inbound interfaces bring external data into the system to help establish it as the single source of truth for company data. These interfaces allow for the efficient and secure transfer of data, which is essential for maintaining the integrity and reliability of business operations.

When setting up an inbound interface, it's important to consider some common pitfalls that can compromise the system's data security cascading its effect on business functions. One of the primary issues is the lack of necessary data quality checks, which can lead to the system accepting incorrect or malicious data. Another significant concern is the misconfiguration of network settings, which can result in data not reaching the intended landing zone or causing network congestion. It's also crucial to be mindful of capacity constraints and avoid overloading the interface with too much data, as this can cause performance issues and potential data loss [3].

Additionally, failing to implement adequate security measures, such as encryption and authentication, can leave the system vulnerable to cyber threats and attacks. To prevent these issues, thorough testing and monitoring should be conducted to ensure the interfaces function within the desired framework and securely transmit the data to its destination.

An important pattern while processing inbound data is a concept of dead letter queues, even after all validations there are chances of failures, and such data fragments must be moved to a separate queue for further analysis without holding up the ingestion of subsequent data cycles.

Fig 2 represents the example of the Master Data ecosystem, and the components involved in the inbound interface module.

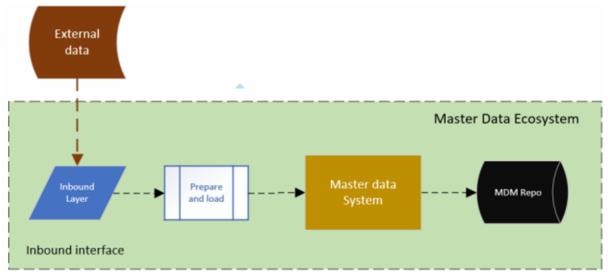

Fig 2: Example of Inbound data interface

3.1.2 Outbound interfaces

Outbound data interfaces are crucial components in outgoing data management and integration into the consuming systems. They facilitate the electronic data transfer from one system to another, which includes internal to internal or from an internal to an external application or system.

The primary purpose of the outbound data interface is to identify the changed data in the source system, prepare and send it to a target system. This may include transfer of data between internal systems within the enterprise, external to the organization or a cloud storage service like a GCP bucket (Buckets in GCP are basic containers where you can store data in the cloud) [6]. For example: A retailer might use an EDI outbound interface to send the Purchase order (Vendor orders) information outside of its ecosystem to communicate the replenishment demand for a store or warehouse location. These interfaces often use standard protocols and formats ensuring data is securely and efficiently transferred. Common protocols include HTTP, SFTP, SOAP, and REST APIs, while the popular data serialization formats include CSV, XML, JSON, AVRO and/or YAML [7].

Organizational data is its intellectual property and must be secured using state of the art security protocols. One such example is PGP data encryption key which is generated by the enterprise and shared with the consumer ahead of time. This key is then used to decrypt the data packet, ensuring its privacy before the consumer can use it.

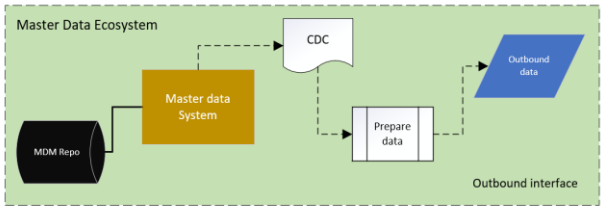

Fig 2: Example of Inbound data interface

4. INTERFACE MODES

Both the inbound and outbound interfaces can be classified into 2 modes,

1. Batch mode (scheduled updates)
2. Real time mode (instant updates)

4.1 Batch mode

This mode works on a frequency model defined over a window of operations. The method collects user or background activities done by a batch service over a period, followed by batch data preparation and dispatch the data either on an hourly, daily, or weekly cadence, depending on how frequently the data updates are needed by the consuming system or application. For instance, the Point of Sale (POS) system could publish transactions to the finance IT system once the point-of-sale register is closed for the day, and hence the frequency is daily. The finance system could aggregate the store sales on a weekly basis and publish the reports to the leadership for the organizational health check [8].

Several techniques can be adopted for batch mode interfaces like Oracle PL/SQL stored proc that can spool the data into temp views or flat files scheduled on a cronjob scheduler to execute at a set time each day; this task records all the activities within a specific timeframe. These interfaces are generally designed on top of the change data capture (CDC) or over window defined on the last and current run time. The popular terms used for last run and current run time are last awake and current awake [5].

4.2 Real time mode

As the name suggests, it deals with real time data updates, the new or modified data is immediately identified, captured, and communicated to the consuming systems. In real time mode, the listener is always looking for a modified version of the data, once it is identified the data extractor service will capture the data and publish to the consumers.

In real time mode, the data or messages can be published using two options: either via PUSH or PULL. The PUSH method publishes one or more messages in real time basis from the producing data source to the target queue, in which case the data is available right away for consumption and the consumers can read it at their ease, whereas in case of the PULL method it solely depends on the consumer when the data or information is needed. Consumers can request the information from producers as and when needed by the application. A PULL method depicts “data on demand”, in which case the producer must provide the latest copy of the requested information to the consumer. It’s a classic use case of the GET HTTP method in RESTful API, in which case the returning service sends a 200-response code indicating success. In the response body, the method returns a representation of a source data.

A real-world example of the PUSH method is a Kafka messaging queue, where an SDK (Software development kit) built on top of the application platform could capture new and/or modified versions of the source data and continue publishing to the Kafka topic as and when changed data is captured. The consumer can read the published messages on a set schedule as per their desired frequency and make necessary decisions using the provided data. As Kafka maintains offsets per consumer, it's easy to add more consumers to the topic without disrupting existing subscribers or the data reliability. As far as data retention in the Kafka topic is concerned, a standard shelf life of published messages is a period of 7 days, after which the data or messages are purged and will not be available anymore to the consumers [12].

As briefly discussed above, a real-world example of the PULL method is RESTful API invocation. In a stock trading platform like Robinhood, an investor could be looking for the current trade price for a NYSE listed stock. When the user searches in the app, a request is triggered from the front end and served by the application server. In this example, a front-end app is requesting data response by sending a company symbol to the app server, the request will first be authenticated, and a response is returned to the end user.

The application server here can be a containerized backend service like TomEE, JBOSS, WebLogic or could simply be a serverless microservice built on Spring boot technology. As far as the actual response is concerned, the server-side code either fetches the data from the database or computes the price on the fly before sending the response back to the consumer. The choice of backend service solely depends on the volume and workload of the application and is directly proportional to the average number of users and requests per user.

The PULL method is a classic representation of real time mode and PUSH is meant to be a good use case for outbound batch mode. However, in some cases the PUSH can also be used in real time mode, where a producer publishes the information right after it is captured by the CDC process.

PUSH mechanism is commonly used in OLAP (Online analytical processing) systems where there is no rush for real time data, and an end of day snapshot should serve the purpose, as the data is mostly used in generating reports or KPIs. PULL is more appropriate for OLTP applications as the decision must be made immediately based on the state of the data at the time of transaction processing, it would mean a binary decision of success or failure.

To conclude, both PUSH and PULL extraction methods can be used in both inbound or outbound data interfaces, where a source (internal or external) publishes the information using PUSH protocol and the target system may consume the data using PULL option.

Note: Every piece of information published using a PUSH process should be complemented by a data PULL and vice versa in case of the opposite. Webhook is a popular REST API communication design pattern used in implementing the PUSH method. It involves inverting the role of consumer and producer in an API exchange. In this pattern the consumer registers an API endpoint with the producer of data, whenever the producer has new information, it invokes the registered API endpoint and sends the data to the server of the consumer (instead of consumer pulling for new information at regular intervals)

5. MANAGEMENT TECHNIQUES

There are several data management techniques used in the inbound and outbound interfaces, and the orchestration layer, we will discuss those in detail in the section below,

To send or receive a new or modified version of data, there must be an information identifier that can be used to distinguish between the various CRUD (Create, Read, Update, Delete) operations. For instance, if the customer address is modified for a customer id on a certain date, then the record must be flagged using an indicator or a metadata for the record to be captured to distinguish among the others. Ultimately the new customer address must be sent over to the consuming application.

We will discuss 2 common industry practices in this section,

1. Trace management - Metadata of the internal events (CRUD operation)
2. Timestamp based Window

5.1 Trace management (Metadata)

One of the popular methods that is widely used in data exchange is trace management, as the term indicates its a trace of the data that is either created, modified, or deleted. Using this reference metadata, the data extractors can prepare the inbound or outbound data and publish it to the targeted application queues. The systems can use simple numerical representation techniques for each of the DML operations in a transaction, for example:

1- Create
2- Modify and
3- Delete or any other representation that suits the custom design and development practice that may fit the need.

In a standard outbound data interface, whether you are publishing the data to a Kafka topic or a traditional batch processing mode that uses flat files like .csv/.txt or even EDI, trace metadata play an important role by taking out the burden of identifying changed data thereby saving a roundtrip for the extraction process. You may then decide whether a full data load or a delta load is desired by the consumer. Delta extract is an efficient way to manage outbound processes as it only targets to capture a true change without having to churn through the entire data set (In database, read operations can be costly, as it may scan the entire dataset blocks to produce an output). For large enterprises like Dollar General, McDonald’s or Walmart, churning through the massive transactional data to find the deltas can be an expensive operation which may slow down the OLTP systems. A trace managed solution design will significantly help in avoiding an expensive CPU operation and do the needful. As software systems landscape and data repository grow, it’s essential for enterprises to ensure the efficient utilization of system resources, particularly CPU and memory.

Traces can be created using multiple methods, and the 2 most popular ones are,

1. System triggers (Implicit)
2. Exclusive trace generation

5.1.1 System triggers (Implicit)

Databases like Oracle or MS SQL Server have built in features like DB table triggers that are invoked on every DML operation and can be used to keep track of the changed data. Triggers can help in effectively capturing traces of the modified data without putting any overhead on the transaction or database. In an ongoing transaction, DB triggers are seamlessly executed without consuming any additional resources and are popular choices; provided are coded with all the right considerations. The choice of whether to use row or statement level triggers depends on the individual use case.

In a RDBMS or NoSQL database, traces could simply be an additional set of a DB tables with only required pieces of information in it, just like any DB metadata views, for example: a trace could be just built with rowid or primary key column (s) with an indicator for type of DML operation. Traces should contain only bare minimum information necessary to identify the changed data but must include key details necessary for the data preparation process, which will be complemented by the extractor in producing the required output for the outbound queue.

P.S: Triggers are nothing, but database stored procedures executed by the core DB service when an event occurs. For instance, an INSERT statement internally creates a ROWID and stores the information in the available block provided by the server's OS. Even though it appears implicit for the DB user, Oracle executes it explicitly as part of its core process.

5.1.2 Exclusive trace generation

The second method of exclusive trace creation is embedded as part of the main transaction; as soon as the primary DML operations are complete, the trace is generated being the last step of the transaction that captures metadata of the changed data (CDC). In some cases, it could cause an overhead to the main transaction and depends on the size of the transaction. This exclusive trace generation method is widely used in most of the master data management systems, in contrast large transactions involving hundreds of steps could use the inbuilt implicit triggering mechanism to avoid the overhead.

An example below represents how traces can be used in a retail price modification scenario. Fig1 shows the current retail price for a Coca-Cola item sold in Smyrna, GA store as of 7/31/2024.

Price List	Location	Item	Price	Priority	Last change
100101	Smyrna Store	Coca-Cola	$2.50	100	7/31/2024

Table 1: Snapshot of the retail price data as of 7/31/2024

As of 8/17/2024, the price is likely to change and an additional new item (Ginger Ale) to be sold in the same store,

Price List	Location	Item	Price	Priority	Last change
100101	Smyrna, GA	Coca-Cola	$2.75	100	8/16/2024
100101	Smyrna, GA	Ginger Ale	$2.65	100	8/16/2024

Table 2: New version of retail price data as of 8/17/2024

See the trace output below,

Price List	Item	Operation	Last change
100101	Coca-Cola	2	8/16/2024
100101	Ginger Ale	1	8/16/2024

Based on the information captured by the trace process, the outbound routines can detect the changes, prepare the output data, extract the required information, and post it to the consumers either in a pub/sub data layer, Kafka topic, or using an ETL tool.

If your enterprise is dealing with large datasets, trace management could add a small amount of overhead on top of the standard storage requirement, approximately 10-15% of additional storage, even though it's not a big portion compared to the benefit it provides. You must ensure there is a single-entry point that is generating the trace. One of the best practices used in trace generation is having a dedicated database schema or a hibernate library, enabling a modular approach per function in a central GIT code repository.

The choice of whether to use a trace or not is determined by the size of the datastore and could vary case to case. As discussed, if the dataset used is between light and medium, then this is an option to consider. For large to very large datasets, there are other approaches that we will discuss in the section below [9].

5.2 Timestamp based

Another common way to manage outbound data interfaces is by using dates or timestamps. This data preparation is driven by a piece of data that acts as metadata within the original data set, for example: a dedicated column in the dataset to indicate the last modified date and time. This can be managed as part of the database or standard data structures like JSON, XML, etc. The idea behind this method is to determine whether the state of the data has changed from the last time it was read. It is a window of operations that takes place between a dynamically defined start and end time. Based on the selection criteria defined in the window, the changed data is captured, prepared, and transmitted out or could be simply dropped into a GCP or Azure cloud bucket.

Managing data transfer using date and time stamps is another popular technique that helps avoid the additional storage requirement compared to trace as the identifier stays as part of the original dataset. For large datasets with hundreds of thousands or millions of transactions like customer orders, this technique is very useful. Imagine a database like Oracle that manages an online transaction processing platform (OLTP) with a heavy transaction workload, by using the proposed approach it can reduce the cost of a full table scan in the outbound process.

This technique is also popularly used in user-interactive applications that may be supporting a back-office operation or a B2C, in which user activities are tracked and communicated back to the database through app servers, a typical 3 tier client-server architecture and is just not limited to outbound interfaces. A web user interface written in ReactJS, or native JavaScript/HTML may be using data structure collection like JDOM to interact with servers through SOAP or REST web service to store or retrieve the information.

JDOM (Java Document Object Model) is a Java-based document processing model that represents XML documents in a way that is easy to read, manipulate, and output. Unlike traditional XML parsers, JDOM is designed specifically for the Java programming language, providing a more intuitive and efficient approach to handling XML data in Java applications.

6. CONCLUSION

This study has demonstrated that interfacing techniques play a pivotal role in maintaining data synchronization and ensuring a decoupled architecture for the core applications. This approach safeguards business applications from becoming overwhelmed due to data pull requests from consumers. While there are additional concepts and aspects related to inbound and outbound interface management, the focus of this paper is intentionally limited to key methodologies and concepts.

One noteworthy concept that is not discussed in detail is the utilization of ETL (Extract, Transform, Load) tools such as Datastage and MuleSoft. These tools offer robust capabilities for data integration and transformation tasks. They enable the seamless extraction of data from various sources, the transformation of the data to conform to specific business requirements, and the subsequent loading of the transformed data into target systems.

In addition to ETL tools, other concepts that could have been explored further include: Data warehousing, Data Governance, Data virtualization, and Data retention policies.

By delving deeper into these concepts, we would gain a more comprehensive understanding of the inbound and outbound interface management landscape. However, the scope of this paper is deliberately focused on the core concepts and key methodologies, leaving room for future research and exploration of additional topics.

7. REFERENCES

[1] W. Xiaojin, S. Shucai, X. Yehua, J. Tao and L. Hongkun, "Research on Data Standardization and Unified Data Interface Based on Digital Station System," 2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Chongqing, China, 2022, pp. 1372-1376, doi: 10.1109/IMCEC55388.2022.10019942.

[2] A. Singh, A. Fisher, C. Allwardt and R. B. Melton, "A Data Exchange Interface for a Standards-Based Data Integration Platform," 2020 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT), Washington, DC, USA, 2020, pp. 1-5, doi: 10.1109/ISGT45199.2020.9087646.

[3] R. B. Melton, K. P. Schneider, E. Lightner, T. E. Mcdermott, P. Sharma, Y. Zhang, et al., "Leveraging Standards to Create an Open Platform for the Development of Advanced Distribution Applications", IEEE Access, vol. 6, pp. 37361-37370, 2018

[4] L. Gifre, M. Ruiz and L. Velasco, "Interfaces for Monitoring and Data Analytics Systems," 2019 21st International Conference on Transparent Optical Networks (ICTON), Angers, France, 2019, pp. 1-4, doi: 10.1109/ICTON.2019.8840489.

[5] J. Sreemathy, I. Joseph V., S. Nisha, C. Prabha I. and G. Priya R.M., "Data Integration in ETL Using TALEND," 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2020, pp. 1444-1448, doi: 10.1109/ICACCS48705.2020.9074186.

[6] P. S. Diouf, A. Boly and S. Ndiaye, "Variety of data in the ETL processes in the cloud migration and validation: State of the art", 2018 IEEE International Conference on Innovative Research and Development (ICIRD), pp. 1-5, 2018.

[7] L. Munoz, J. Mazon and J. Trujillo, ETL Process Modeling Conceptual for Data Warehouse: A Systematical Mapping Study, vol. 2016, June 2011.

[8] T. Mi, R. Aseltine and S. Rajasekaran, "Data Integration on Multiple Data Sets," 2008 IEEE International Conference on Bioinformatics and Biomedicine, Philadelphia, PA, USA, 2008, pp. 443-446, doi: 10.1109/BIBM.2008.48.

[9] A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 169-178, 2000.

[10] Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasets. CIKM 02: Proceedings of the eleventh international conference on Information and knowledge management, pages 515-524, 2002.

[11] R. Shree, T. Choudhury, S. C. Gupta and P. Kumar, "KAFKA: The modern platform for data management and analysis in big data domain," 2017 2nd International Conference on Telecommunication and Networks (TEL-NET), Noida, India, 2017, pp. 1-5, doi: 10.1109/TEL-NET.2017.83435