JCRES
Journal of Current Research in Engineering and Science
Bi-Annual Online Journal (ISSN : 2581 - 611X)

1 Volume 7- Issue 1, January 2024
2 Paper : 95

Enhancing Real-Time Data Management through Software Architecture Development

Nagendra Herle[1],Jayasimha S R[2]
[1,2]Master of Computer Application
[1,2]R.V. College of Engineering
Bengaluru,India
[1]nagendrah.mca21@rvce.edu.in,[2]jayasimhasr@rvce.edu.in

Abstract—**The process of improving the management of real-time data by leveraging advancements in software architecture. It involves designing and implementing software architectures that are specifically tailored to handle the complexities and requirements of real-time data processing, storage, and retrieval. Real-time database or data management refers to the process of managing and manipulating data in real-time, where data is processed and updated immediately as it is generated or received.The objective of this study is to investigate how software architecture can contribute to efficient real-time data management.The methods employed in this research involve a comprehensive review of existing literature on real-time data management and software architecture.Various approaches and frameworks for real-time data processing, storage, and retrieval are examined, with a focus on their architectural implications.Study emphasizes the value of software architecture development in advancing real-time data management and provides insights into the potential benefits and challenges associated with adopting architectural strategies for managing real-time data effectively.**

***Keywords*—Microservices, Monolith, NoSQL, MongoDB, Real-time data management**

I. Introduction

In today's rapidly evolving digital landscape, the importance of data cannot be overstated. Data has become a valuable asset for organizations across various industries, enabling informed decision-making, driving innovation, and enhancing operational efficiency. As the volume, variety, and velocity of data continue to grow exponentially, it has become essential for organizations to effectively store, manage, and secure their data assets.In today's rapidly evolving digital landscape, the importance of data cannot be overstated. Data has become a valuable asset for organizations across various industries, enabling informed decision-making, driving innovation, and enhancing operational efficiency.As the volume, variety, and velocity of data continue to grow exponentially, it has become essential for organizations to effectively store, manage, and secure their data assets [1].Data storage and management are critical components of any data-driven organization. Proper storage infrastructure and efficient management practices ensure that data is organized, accessible, and available for analysis and decision-making purposes. Organizations employ various data storage technologies, such as databases, data warehouses, data lakes, and cloud storage solutions,to cater to their specific needs.Data lakes allow to quickly consolidating various types of data in one repository[8].Additionally, data management practices encompass data governance, data quality assurance, data integration, and data lifecycle management, among others.Real-time data has emerged as a critical requirement in many industries and applications.

Real-time data refers to the ability to access, process, and analyze data as it is generated, with minimal delay.Real-time data has emerged as a critical requirement in many industries and applications. Real-time data refers to the ability to access, process, and analyze data as it is generated, with minimal delay. This is particularly valuable in scenarios where time-critical decisions need to be made, such as financial trading, supply chain management, and emergency response systems. Real-time data enables organizations to monitor and respond to events as they occur, providing immediate insights and facilitating proactive decision-making.

Real-time database systems (RTDBSs) have been widely applied in many areas[2]. Industrial control [3], vehicle control [4], aircraft control [5], health monitoring [6], and robot control [7] are a few examples of such applications.

Maintaining real-time data poses its own set of challenges. It requires robust and scalable data management systems that can handle high volumes of incoming data in a timely manner. Processing and analyzing data in real-time often necessitate the use of specialized technologies such as stream processing engines, complex event processing, and in-memory databases. Furthermore, ensuring data consistency, reliability, and accuracy in real-time environments is crucial to prevent data loss or inconsistencies that could impact decision-making processes.

The specific difficulties and demands of handling data in real-time give rise to the necessity for software architecture in real-time database administration.Software architecture refers to the high-level design and organization of software components and their interactions within a system.Software architecture can be viewed as an organization of a system that comprehensively includes components interactions, operational environments, design principles, software functionalities, and often covers future evolutionary software perspective [10]-[13].Software architecture has emerged as the initial comprehension of the large-scope structures of software systems[9].The need for software architecture in real-time database management arises from the unique challenges and requirements of handling data in real-time. Real-time databases deal with high volumes of data that need to be processed, stored, and accessed in a time-critical manner. Software architecture allows for the design of systems that can efficiently handle this continuous flow of data and meet the stringent performance requirements.A well-designed software architecture enables scalability and performance in real-time database management. It allows the system to handle increasing data volumes and user demands by efficiently distributing the workload across multiple nodes or clusters. This scalability ensures that the system can maintain responsiveness even during periods of high data influx.Software architecture facilitates data integration and interoperability in real-time database management. It provides mechanisms for seamlessly integrating data from diverse sources and systems, such as IoT devices, sensors, and external data feeds. This integration ensures a unified view of the data and enables real-time access to consolidated and integrated information.

Architectural approaches like monolithic and microservices offer different perspectives on system design. A monolithic architecture represents an application as a single, tightly integrated unit. While simple to develop, it may face challenges in scalability and agility. On the other hand, microservices architecture decomposes an application into independent services that can be developed, deployed, and scaled independently. This modular approach enables the integration of real-time data processing services with the database infrastructure, providing flexibility and scalability.

A. *Motivation and Problem statement*

In today's rapidly evolving world, the generation of data has reached unprecedented levels in terms of its volume, speed, and variety. Across industries, organizations recognize the tremendous value that real-time data holds in gaining a competitive edge. Real-time data offers immediate insights, enabling informed decision-making, agile responses to market dynamics, and the delivery of personalized customer experiences. It empowers businesses to optimize operations, detect anomalies, and capitalize on emerging opportunities. However, fully harnessing the potential of real-time data requires effective management and efficient utilization.

Moreover, the need for scalability and performance in real-time data management cannot be overlooked. As data volumes and processing demands increase, organizations must have scalable architectures and systems that can handle the growing workload efficiently. Real-time data management should also address the latency challenge, ensuring that data is accessible and processed within the required time frames.

The following research questions were posed to guide the study:

- (RQ1) How can organizations effectively manage and process real-time data of various types and structures, considering the inherent data variety?
- (RQ2) How can real-time data integration be optimized to efficiently handle diverse data sources and formats in a unified manner?
- (RQ3) What techniques and technologies can mitigate data latency challenges in real-time data management, ensuring timely access and processing?

The need for scalability and performance in real-time data management cannot be overlooked. As data volumes and processing demands increase, organizations must have scalable architectures and systems that can handle the growing workload efficiently.Real-time data management should also address different methodologies and frameworks can be utilized to ensure the security and privacy of real-time data.

B. *Outline of the Paper*

The rest of the paper is organised as follows. The following section describes details of the databases and architectural styles, and compares and contrasts their advantages and disadvantages.Section 3 discusses related work.The research methods are explained in Section 4. This is followed by Section 5, concludes the study along with suggestions for future research.

II. BACKGROUND

In the subsequent sections, we present more details on software architectures as well as database management.

A. *Understanding the Basics: Data, Information, Database, and DBMS Explained*

Data: Data refers to raw, unprocessed facts, figures, or symbols that represent various aspects of the real world. It typically consists of numbers, text, images, or other forms of input that can be stored and processed by computer systems.

Information: Information is the processed and organized form of data that has meaning and relevance. It results from the analysis, interpretation, or manipulation of data, providing insights or knowledge that can support decision-making, understanding, or communication.

Database: A database is a structured collection of data that is organized, stored, and managed in a systematic way to support efficient data retrieval, manipulation, and sharing. It serves as a central repository for storing and accessing related information, allowing multiple users or applications to interact with the data concurrently.

DBMS (Database Management System): A DBMS is a software system that enables the creation, organization, retrieval, and manipulation of data in a database. It provides a set of tools, functions, and capabilities to efficiently manage data, enforce data integrity and security, and facilitate data interactions between users or applications. A DBMS ensures data consistency, concurrency control, and data recovery in case of failures or errors.

1 Volume 7- Issue 1, January 2024
2 Paper : 95

These definitions highlight the distinction between data and information, where data represents the raw input, while information is the processed output that provides meaningful insights. A database serves as a structured repository for storing and managing data, and a DBMS acts as the software that enables efficient management and utilization of the database.

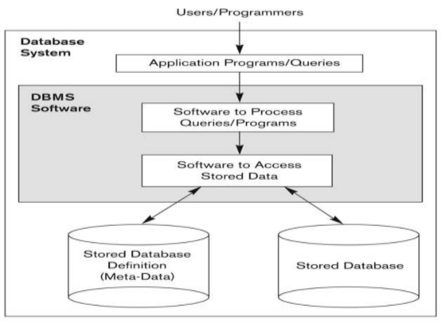
Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

FIGURE 1: Simplified database system environment.

B. Relational Databases

Relational databases are a widely used type of database management system (DBMS) that organizes data into tables with predefined relationships between them.For many different organizations, including large multinational firms such as Google and Facebook, data forms a strategic asset that must be carefully curated and protected [14]. Indeed, fields such as healthcare, science, and commerce often rely on information that is stored in databases [15]. While non-relational "NoSQL" systems have been gaining in popularity, relational databases remain pervasive[20]. For instance, Skype, the widely used video-call software, uses the PostgreSQL database management system (DBMS) [16] while Google makes use of the SQLite DBMS in Android-based phones [17]. Moreover, relational databases form the backbone of Internet web browsers such as Chrome[1] and Firefox[2], mobile applications [18], and even software powering political campaigns [19].

1. https://www.google.com/chrome/browser
2. http://www.mozilla.org/firefox

In a relational database, data is organized into tables, which consist of rows and columns. Each table represents a specific entity or concept, and each row represents a unique record or instance of that entity. Columns, also known as attributes, define the characteristics or properties of the entity. The relationships between tables are established through keys, such as primary keys and foreign keys, which ensure data integrity and enforce referential integrity.

Relational databases use Structured Query Language (SQL) for data manipulation and retrieval. SQL databases are a general term for relational DBMS like MySQL [21], Oracle [22], and PostgreSQL [23].SQL provides a standardized language for creating, modifying, and querying the database. It allows users and applications to perform operations such as inserting data into tables, updating existing records, deleting records, and querying data using powerful filtering and aggregation capabilities.

Despite their widespread use and long history, relational databases have some limitations when it comes to handling real-time data. Relational databases can cause latency in the context of real-time data management due to its disk-based storage and structured query processing, which impedes immediate data access and updates. In order to manage high-velocity and high-volume real-time data streams, they may also encounter difficulties scaling horizontally.

C. NoSQL Databases

NoSQL databases are designed for handling large-scale, unstructured or semi-structured data. They offer flexible schemas and can handle high-velocity data. Examples include MongoDB, Cassandra, and Redis.NoSQL databases have gained significant popularity and are widely used in various applications, including those requiring real-time data processing. In contrast to traditional relational databases, NoSQL databases offer specific advantages that make them well-suited for real-time data management.

Real-time applications often deal with a continuous flow of data that needs to be ingested, processed, and analyzed in near real-time. NoSQL databases, with their distributed architecture and horizontal scalability, excel at efficiently handling large volumes of data and providing high-throughput data ingestion and processing capabilities. This makes them suitable for use cases such as real-time analytics, event-driven applications, and streaming data processing.

All kinds of data are stored and handled in NoSQL databases in a number of ways, such as a document-oriented store, a column-oriented store, a KeyValue store, and a graph-based store.

MongoDB is a document-oriented Non-relational database that can be used to distribute and store large binary files like videos and images [24].It utilizes a binary representation known as BSON (Binary JSON) objects to store data in a JSON-like format. MongoDB's design allows related information to be stored together, enabling efficient querying through its query language. Data in MongoDB is organized into collections, and the structure of documents can vary without the need for declaring it explicitly to the system. This self-describing nature of documents eliminates the requirement for system-wide updates or downtime when adding new fields to a document. As a result, MongoDB delivers superior performance compared to other databases, ensuring optimal resource utilization and efficient long-term storage. Indexing over embedded objects and arrays is also supported by MongoDB. In comparison to other NoSQL databases, it works well with memory storage, complicated data, and dynamic queries. As a scaleout-based system, MongoDB offers flexibility to function in the event of hardware expansion. [25]-[26].

While NoSQL databases provide significant benefits for real-time data management, it's crucial to remember that the decision between NoSQL and conventional relational databases depends on the requirements of the individual application. In situations where intricate transactions, tight data consistency, or significant SQL querying capabilities are required, relational databases may still be desirable. However, NoSQL databases offer useful capabilities and performance characteristics when processing real-time data that is flexible, fast-moving, and changing often.

D. Software Architecture

Software architecture encompasses the fundamental design choices made during the development process. It is considered a fundamental aspect of software engineering that precisely defines the essence of software system design and development. Software architecture plays a crucial role in fulfilling both functional and non-functional requirements and is an essential component throughout the software evolution lifecycle.

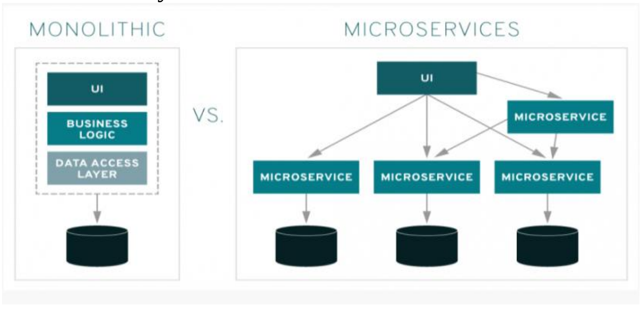

FIGURE 2: Monolithic and microservices architecture[37]

Monolithic architecture is a traditional software design approach where the entire application is built as a single, unified unit. In this architecture, all the components, modules, and functionality of the application are tightly coupled and interdependent.

One of the main strengths of monolithic architecture lies in its simplicity. Unlike distributed applications with various components, monolithic architectures are relatively easier to test, deploy, debug, and monitor. By storing all data in a single database without the need for synchronization, internal communication within the application occurs through intra-process mechanisms. As a result, it offers faster performance and avoids the typical challenges associated with inter-process communication (IPC).

Nevertheless, as the application expands in size and complexity, challenges begin to emerge. Modifying the application's source code becomes increasingly difficult as intricate code begins to exhibit unexpected behaviors. Making changes in one module can lead to unforeseen consequences in other modules, causing a cascade of errors. Additionally, the sheer size of the monolith contributes to longer start-up times, impeding development speed and hindering continuous deployment efforts. Also, as the application grows, the number of developers increases, which often leads to unequal workforce utilization and, in effect, losses in productivity [27].

Microservice architecture is an architectural style that structures an application as a collection of loosely coupled and independently deployable services. Each service is designed to focus on a specific business capability and can be developed, deployed, and scaled independently of other services.Microservices rely on standardized internet protocols for communication, including HTTP and REST. They may also utilize messaging protocols such as JMS (Java Message Service) or AMQP (Advanced Message Queuing Protocol). These protocols provide a consistent and efficient means of communication between microservices, enabling seamless interaction within a microservices architecture.

The most attractive feature of the microservice architecture is the decomposition of complex applications into smaller components which are easier to develop, manage and maintain than a single monolith application [28].Microservices are autonomous and communicate via open protocols, hence they can be developed fairly independently and even with different technologies [29], [30]–[31].Microservice-based applications exhibit excellent horizontal scalability, benefiting not only from technical advantages but also from the streamlined organization and agility of developer teams [29], [33], [34], [35].

In addition to its many benefits, the microservice architecture has certain limitations and drawbacks, particularly due to its distributed nature. The process of deploying, scaling, and monitoring a multi-service system is more intricate compared to a monolithic application.Another challenge arises in the design of data management facilities within the microservice architecture. According to the principles of microservices, it is preferable to have maximum service isolation. As a result, multiple independent database systems are incorporated into the distributed application, which adds complexity and reduces manageability [27].

III. Related Work

In the domain of real-time data management, extensive research has been conducted to investigate diverse scenarios and databases in order to comprehend their functionalities and constraints.Previous studies have predominantly concentrated on evaluating various metrics such as storage capacity, query syntax, query latency, database connection time, and schema design, as emphasized by [24]. However, it is important to note that the existing surveys and systematic reviews in the realm of real-time data processing research have been found to have certain limitations. Specifically, these studies lack comprehensive coverage of the publication channels, approaches, challenges, and solutions that are specifically tailored to address the unique requirements of business intelligence. Therefore, there is a significant research gap in terms of providing a holistic understanding of real-time data processing within the context of business intelligence.

A recent study conducted by [36] aimed to address this gap by conducting a comparative analysis of various data stream analytics frameworks. The evaluation specifically assessed the partitioning, state management, message delivery, and fault tolerance capabilities of popular data stream processing engines such as Storm, Spark Streaming, Flink, Kafka Streams, and IBM Streams. While the study provided valuable insights into the technical aspects of these frameworks, it is important to note that the primary focus was not on extracting knowledge or identifying crucial components for real-time stream analytics, which are fundamental requirements in the field.

In their noteworthy study, [37] dedicated their efforts to investigating the realm of real-time stream processing and big data streaming. Their research took a close look at the intricacies of the Extract, Transform, Load (ETL) architecture for Predictive Analytics and Smart Decision-Making, shedding light on its implementation challenges and advancements in join operations for real-time data processing.
While the study provided valuable insights into these aspects, it is important to note that the researchers did not primarily focus on the utilization of software architecture to enhance real-time data management. Although their exploration of the ETL architecture and its implications for real-time data processing is commendable, the specific role and potential impact of software architecture in optimizing real-time data management were not extensively addressed in their research.
Our review sets itself apart from previous studies by placing a specific emphasis on exploring the publication channels that are closely associated with real-time data processing. We recognize the crucial role played by software architecture in enhancing the management of real-time data, and we delve deeper into this topic by examining key components such as cache memory, message brokers, and their impact on addressing critical challenges such as data integration and data latency. By thoroughly investigating these aspects, our review offers comprehensive insights and valuable contributions to the field of real-time data processing and its related domains.The table below presents the details of the literature survey.

TABLE I: LITERATURE SURVEY

Number	Author and Paper title	Parametrs	Summary of the Paper
1.	Real-Time Context-Aware Microservice Architecture for Predictive Analytics and Smart Decision-Making[43]. Authors: Guadalupe Ortiz; José Antonio Caravaca; Alfonso García-de-Prado; Fràncisco Chavez de la O; Juan Boubeta-Puig	Published in: IEEE Access (Volume: 7) Date of Publication: 18 December 2019 DOI: 10.1109/ACCESS.2019.2960516	This paper introduces a context-aware architecture based on microservices that offers intelligent real-time predictions and context-aware notifications. The system utilizes user subscriptions to deliver personalized and relevant information to users.
2.	Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation[44] Authors:Grzegorz Blinowski; Anna Ojdowska; Adam Przybyłek	Published in: IEEE Access (Volume: 10) Date of Publication: 18 February 2022 INSPEC Accession Number: 21643479 DOI: 10.1109/ACCESS.2022.3152803	It is important to consider the specific context and requirements when choosing between a microservice and monolithic architecture. While microservices offer flexibility and scalability, a monolithic architecture may be more suitable for simple systems. It is crucial for companies to avoid blindly adopting microservices, instead evaluate if scaling up their monoliths can yield better results.
3.	Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments[24] Authors:Mahmoud Moustafa Eyada; Walaa Saber; Mohammed M. El Genidy; Fathy Amer	Published in: IEEE Access (Volume: 8) Date of Publication: 15 June 2020 INSPEC Accession Number: 19799020 DOI: 10.1109/ACCESS.2020.3002164	This paper conducted a comparative analysis of MongoDB and MySQL in handling large-scale heterogeneous IoT data. Multiple scenarios were created to assess the performance of both databases. The findings indicated that as the workload increased, MySQL exhibited a significant decline in performance compared to MongoDB.

Journal of Current Research in Engineering and Science
Bi-Annual Online Journal (ISSN : 2581 - 611X)

Volume 7- Issue 1, January 2024
Paper : 95

4.	Challenges and Solutions for Processing Real-Time Big Data Stream: A Systematic Literature Review [37]. Authors:Erum Mehmood; Tayyaba Anees	Published in: IEEE Access (Volume: 8) Date of Publication: 26 June 2020 INSPEC Accession Number: 19799743 DOI: 10.1109/ACCESS.2020.3005268	This survey aims to provide researchers with valuable guidance in the field of real-time stream analysis for DWH and big data applications. It offers insights into implementation challenges, approaches, tools, and evaluation evidence related to real-time stream processing in various application domains.
5.	Performance Analysis of Not Only SQL Semi-Stream Join Using MongoDB for Real-Time Data Warehousing[40]. Authors:Erum Mehmood, Tayyaba Anees	Published in: IEEE Access (Volume: 7) Date of Publication: 17 September 2019 INSPEC Accession Number: 19001954 DOI: 10.1109/ACCESS.2019.2941925	This paper addresses the challenges of joining NoSQL streams, specifically focusing on the semi-stream join processing of unstructured and structured data streams from disk.

IV. Research Methodology

In this section, we propose implementation method for the real time data module and the software architecture.Architecture shown in figure 1, presents real-time stream processing.In our implementation of real-time data processing, we began by gathering the dataset. To facilitate the extraction of text from the data in real-time, we employed the machine learning algorithm called easyOCR. The collected dataset was then stored in MongoDB, a popular document-oriented NoSQL database. To establish a connection with the database and enable efficient communication, we utilized RabbitMQ as a message broker, which facilitates the exchange of messages between different components of the system. Additionally, to minimize the need for frequent database fetch operations and improve performance, we leveraged Redis, an in-memory caching solution. By employing Redis, we were able to store and retrieve frequently accessed data quickly, reducing the overall response time of the system.

MongoDB Compass with MongoDB Server version 3.0 have been installed for our implementation on a machine running an Intel Core i5 1.70 GHz processor with 4GB of RAM. Proposed applications were coded in python using IDLE (python 3.6 64-bit).

In sub-section A,focusing specifically on the importance of managing data variety within the system. We delve into the strategies and techniques employed to ensure effective management of diverse data types.

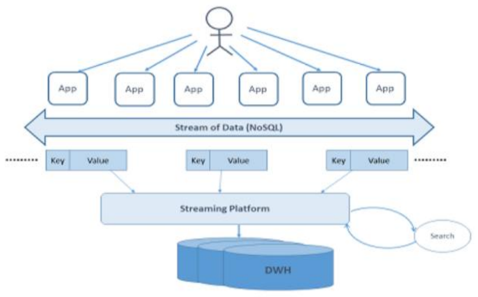

FIGURE 3: Real-time data processing architecture[40]

Moving to sub-section B, we provide a detailed examination of the database, we present a comprehensive overview of the integration and communication processes between various components, placing a strong emphasis on the criticality of efficient data integration and the effective handling of diverse data types.

In sub-section C, our attention turns to the functioning of in-memory databases. We provide a comprehensive overview of how these databases operate within the system, considering their role in optimizing data storage and retrieval. Finally, in sub-section D, we provide a concluding overview of the system's overall functioning, consolidating the various components discussed in the preceding sub-sections.

A. *Enhancing Real-Time Data Management through Microservices Architecture for Diverse Data Types*

In the context of real-time databases, the flexibility in managing diverse data types is facilitated by the microservices architecture. By decomposing the system into individual services, each service can be specifically designed to handle and process a particular data format or type.Microservices architecture provides flexibility in managing diverse data types.

Microservices architecture provides the opportunity to optimize resource utilization by allocating them more intelligently. Unlike monolithic applications where resource allocation is fixed, microservices allow for independent scaling of specific components or functionalities. By implementing a microservices architecture, we can selectively scale the specific components of the code that require additional resources, while keeping resource usage at a lower level for the remaining parts.

This flexibility allows us to allocate resources efficiently to the appropriate sections of the system, optimizing performance and resource utilization. For instance, in a scenario where a monolithic application uses 8GB of RAM, scaling to two instances would result in a total RAM usage of 16GB.

In contrast, with microservices, the heavy-lifting portion of the code can be distributed across multiple instances, each utilizing a fraction of the resources. As a result, the overall RAM usage for two instances of microservices may be significantly lower, such as 9.6GB in this example. This distinction in resource usage is illustrated in the accompanying diagram, highlighting the efficiency gains achievable through microservices architecture.

Developing and running a microservices application presents its own set of complexities, particularly related to containerization and cluster management. These tasks require significant expertise and careful handling to ensure smooth operation.On the other hand, monolithic applications offer relative simplicity during development, as they involve a single codebase and a single library. Running a monolithic application is also straightforward, especially at smaller scales, as developers can easily keep track of all the components.

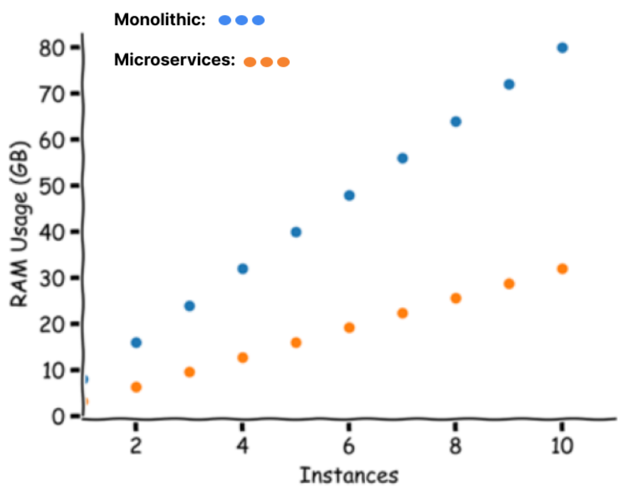

FIGURE 4: Resource usage [38]

However, as applications grow larger and more complex, managing monolithic architectures becomes increasingly challenging. Legacy monolithic applications can become unwieldy, with millions of lines of code that are difficult to manage and maintain. This often leads to the need for migration to microservices architectures, as the complexities of managing and scaling monolithic applications become overwhelming.

Therefore, while monolithic applications may appear simpler initially, when it comes to large-scale applications, microservices architectures are better suited despite the inherent complexities involved. The flexibility and scalability offered by microservices outweigh the challenges, making them a preferable choice for managing complex and expansive systems.

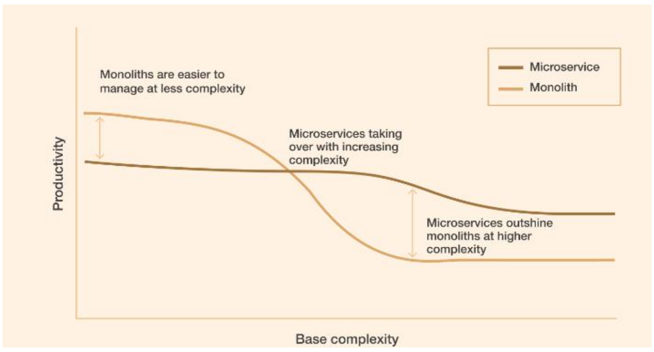

FIGURE 5: A Comparison of Complexity: Monoliths vs. Microservices [39]

Assement of RQ1: By decomposing the system into individual services, each responsible for processing specific data formats, different data types can be efficiently integrated and transformed.In our implementation, we utilize the easyOCR machine learning algorithm to detect text within videos. By employing a microservices architecture, we can develop distinct services specialized in handling different data types. Specifically, we create a text processing service dedicated to parsing and analyzing textual data, while a separate video processing service is responsible for efficient video streaming.

In the context of real-time data processing, we employ the RabbitMQ message broker for seamless integration of text detection results into the database. As a reliable and scalable communication tool, RabbitMQ facilitates efficient and secure communication between various components of the system, ensuring smooth data integration.

B. *Database Integration and Data Handling*

Assement of RQ2: RabbitMQ is a popular open-source message broker that facilitates communication and data transfer between different components of a distributed system. It follows the Advanced Message Queuing Protocol (AMQP) and is designed to handle high volumes of messages efficiently and reliably. The event bus implementation with RabbitMQ lets microservices subscribe to events, publish events, and receive events, as shown in Figure 6.

Volume 7- Issue 1, January 2024
Paper : 95

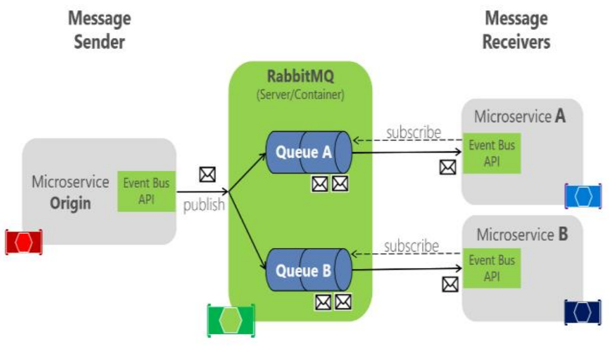

FIGURE 6: RabbitMQ implementation of an event bus[42]

RabbitMQ functions as an intermediary between message publisher and subscribers, to handle distribution.
The detected text is transmitted through the RabbitMQ message broker, where it is consumed by the RabbitMQ consumer. The message body contains the detected text, which is then fetched and processed from the MongoDB database to retrieve the associated data.
RabbitMQ plays a crucial role in enabling efficient communication and decoupling of components in distributed systems, making it a popular choice for building reliable and scalable messaging solutions.
Code for the database connection:

```
client1=pymongo.MongoClient('mongodb://localhost:27017/ais')
db1 = client1['ais']
collection1 = db1['navy_data']
documents = collection1.find({"Pennant Number": body.decode()})
```

Code for the RabbitMQ connection:

```
connection=pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='my_queue')
```

Set up the publish and start publishing messages

```
channel.basic_publish(exchange='',
routing_key='my_queue', body=text)
```

Set up the consumer and start consuming messages

```
channel.basic_consume(queue='my_queue',
on_message_callback=callback, auto_ack=True)
print('Waiting for messages. To exit, press CTRL+C')
channel.start_consuming()
```

C. In-memory Databases

Assement of RQ3:An in-memory database is a type of database management system that stores data entirely in the main memory (RAM) of a computer or server, rather than on traditional disk storage. It is designed to provide fast and efficient access to data by eliminating the need to read from or write to disk.

Redis is an open-source, in-memory data structure store that is commonly used as a database, cache, and message broker. It is designed for high-speed data access.Redis operates primarily in memory, which allows for extremely fast read and write operations. It stores data in key-value pairs and offers various commands and operations for efficient data manipulation and retrieval. With its in-memory nature, Redis excels in use cases that require low-latency and high-throughput data processing, such as real-time analytics, session caching, and messaging systems.In-memory databases like Redis or Apache Ignite can reduce data latency by caching frequently accessed data in memory. This speeds up data retrieval by minimizing disk I/O operations, enabling near-instantaneous access to real-time data.

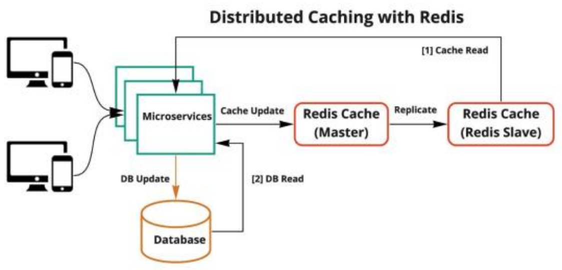

FIGURE 7: Redis cluster architecture for high availability[42]

The retrieved data from the database is cached in Redis, an in-memory data store. To optimize performance, an expiration time is associated with the cached data, ensuring that it is automatically flushed after a specific period. This caching mechanism helps in avoiding repetitive fetch operations from the database.

Whenever duplicate text data is received through the message broker, the system checks if it already exists in the Redis cache. If the data is found in the cache, it eliminates the need for a time-consuming fetch operation from the database. By leveraging the Redis cache, the system can significantly improve response times and reduce the load on the database, resulting in faster and more efficient data retrieval.

Code for the Redis connection:

```
redis_client = redis.Redis(host='localhost', port=6379, db=0)

redis_client.set(body.decode(), json_data)

redis_client.expire(body.decode(), 5)
```

D. Result

Based on the provided information, the following results can be inferred:

Flexibility in managing diverse data types: The implementation of a microservices architecture allows for the decomposition of the system into individual services, enabling efficient handling and processing of specific data formats. This result enhances flexibility in managing diverse data types within the real-time database system.

Successful integration of text detection results using RabbitMQ: By employing RabbitMQ as a message broker, the system achieves seamless communication and data transfer between components. This result ensures the successful integration of text detection results into the

MongoDB database, facilitating the retrieval of associated data.

Improved performance and reduced data latency through Redis caching: The utilization of Redis as an in-memory data store for caching retrieved data results in improved system performance. This caching mechanism reduces data latency by minimizing the need for repetitive fetch operations from the database.

Enhanced efficiency and scalability in real-time data processing: The combination of microservices architecture, RabbitMQ message broker, and Redis caching contributes to enhanced efficiency and scalability in real-time data processing. This result leads to faster and more efficient data retrieval, improving the overall performance of the system.

Overall, the achieved results demonstrate the successful implementation of a flexible and efficient real-time data processing system, leveraging the benefits of software architecture, particularly microservices. This system effectively manages diverse data types, enables seamless integration between components, and significantly reduces data latency, resulting in improved performance. By adopting a microservices architecture, the system decomposes into individual services that handle specific data formats, enhancing flexibility and scalability in data processing. The integration of a message broker, such as RabbitMQ, ensures reliable communication and data transfer, while an in-memory database like Redis optimizes performance by caching retrieved data. Together, these components and architectural choices contribute to the system's overall success in achieving efficient real-time data processing.

V. Conclusion and Future Work

The objective of this survey is to offer valuable guidance to researchers in the field of real-time data management using software architecture. Our aim is to explore various applications, advancements, and challenges in this domain, providing insights into the methodology employed. By conducting a comprehensive investigation, we seek to contribute to the existing body of knowledge and assist researchers in their exploration of real-time data management. Flexibility in managing diverse data types: The implementation of a microservices architecture allows for the decomposition of the system into individual services, enabling efficient handling and processing of specific data formats. This result enhances flexibility in managing diverse data types within the real-time database system.

Successful integration of text detection results using RabbitMQ: By employing RabbitMQ as a message broker, the system achieves seamless communication and data transfer between components. This result ensures the successful integration of text detection results into the MongoDB database, facilitating the retrieval of associated data.

Improved performance and reduced data latency through Redis caching: The utilization of Redis as an in-memory data store for caching retrieved data results in improved system performance. This caching mechanism reduces data latency by minimizing the need for repetitive fetch operations from the database.
Enhanced efficiency and scalability in real-time data processing: The combination of microservices architecture, RabbitMQ message broker, and Redis caching contributes to enhanced efficiency and scalability in real-time data processing. This result leads to faster and more efficient data retrieval, improving the overall performance of the system.

Overall, the achieved results demonstrate the successful implementation of a flexible and efficient real-time data processing system, leveraging the benefits of software architecture, particularly microservices. This system effectively manages diverse data types, enables seamless integration between components, and significantly reduces data latency, resulting in improved performance.

In terms of future scope, several areas can be explored to further enhance the system. These include scaling and performance optimization to handle larger volumes of data, incorporating real-time analytics and insights generation, strengthening security and data privacy measures, integrating with external systems or APIs, and implementing continuous monitoring and error handling mechanisms.

By addressing these future areas of development, the real-time data processing system can continue to evolve and meet the evolving needs of real-time data-driven applications, providing even greater flexibility, scalability, performance, and insights.

REFERENCES

1. “Adaptive Trust Management and Data Process Time Optimization for Real-Time Spark Big Data Systems”,Seungwoo Seo; Jong-Moon Chung,Published in: IEEE Access (Volume: 9),Date of Publication: 22 November 2021
2. Guarantee the Quality-of-Service of Control Transactions in Real-Time Database Systems, Chenggang Deng; Guohui Li; Quan Zhou; Jianjun Li,Published in: IEEE Access (Volume: 8),Date of Publication: 15 June 2020
3. M. Canizo, A. Conde, S. Charramendieta, R. Miñón, R. G. Cid-Fuentes, and E. Onieva, ‘‘Implementation of a large-scale platform for cyber-physical system real-time monitoring,’’ IEEE Access, vol. 7,pp. 52455–52466, 2019.
4. T. Gustafsson and J. Hansson, ‘‘Data management in real-time systems: A case of on-demand updates in vehicle control systems,’’ in Proc. RTAS 10th IEEE Real-Time Embedded Technol. Appl. Symp., May 2004, pp. 182–191.
5. X. Shi, Y. Shen, Y. Wang, and L. Bai, ‘‘Differential-clustering compression algorithm for real-time aerospace telemetry data,’’ IEEE Access, vol. 6, pp. 57425–57433, 2018.
6. J. Ko, C. Lu, M. B. Srivastava, J. A. Stankovic, A. Terzis, and M. Welsh, ‘‘Wireless sensor networks for healthcare,’’ Proc. IEEE, vol. 98, no. 11, pp. 1947–1960, Nov. 2010.
7. S. Han, A. K. Mok, J. Meng, Y. H. Wei, P. C. Huang, Q. Leng, X. Zhu, L. Sentis, K. S. Kim, and R. Miikkulainen, ‘‘Architecture of a

cyberphysical avatar,'' in Proc. ACM/IEEE Int. Conf. Cyber-Phys. Syst., Philadelphia, PA, USA, Apr. 2013, pp. 189–198.
8. Data Lake Lambda Architecture for Smart Grids Big Data Analytics,Amr A. Munshi; Yasser Abdel-Rady I. Mohamed,Published in: IEEE Access (Volume: 6),Date of Publication: 23 July 2018
9. Software Architecture Degradation in Open Source Software: A Systematic Literature Review,Ahmed Baabad; Hazura Binti Zulzalil; Sa'adah Hassan; Salmi Binti Baharom,Published in: IEEE Access (Volume: 8),Date of Publication: 18 September 2020
10. M. Sahlabadi, R. C. Muniyandi, Z. Shukur, and F. Qamar, ''Lightweight software architecture evaluation for industry: A comprehensive review,'' Sensors, vol. 22, no. 3, p. 1252, Feb. 2022.
11. T. Yang, Z. Jiang, Y. Shang, and M. Norouzi, ''Systematic review on nextgeneration web-based software architecture clustering models,'' Comput. Commun., vol. 167, pp. 63–74, Feb. 2021.
12. C. C. Venters, R. Capilla, S. Betz, B. Penzenstadler, T. Crick, S. Crouch, E. Y. Nakagawa, C. Becker, and C. Carrillo, ''Software sustainability: Research and practice from a software architecture viewpoint,'' J. Syst. Softw., vol. 138, pp. 174–188, Apr. 2018.
13. W. Hasselbring, ''Software architecture: Past, present, future,'' in The Essence of Software Engineering. Cham, Switzerland: Springer, 2018, pp. 169–184.
14. P. Glikman and N. Glady, "What's the value of your data?" (2015). [Online]. Available: https://goo.gl/bFZKeR, Accessed on: 6-Dec.- 2016
15. G. M. Kapfhammer, "A comprehensive framework for testing database-centric applications," Ph.D. dissertation, Univ. Pittsburgh, Pittsburgh, PA, USA, 2007.
16. PostgreSQL featured users. [Online]. Available: https://www. postgresql.org/about/users/, Accessed on: 6-Dec.-2016
17. Well-known users of SQLite. [Online]. Available: https://www. sqlite.org/famous.html, Accessed on: 10-Dec.-2016
18. K. Roukounaki, "Five popular databases for mobile." (2014). [Online]. Available: https://goo.gl/rAUAe0, Accessed on: 6-Dec.- 2016
19. B. Butler, "Amazon: Our cloud powered Obama's campaign," Netw. World, 2012.
20. Automatic Detection and Removal of Ineffective Mutants for the Mutation Analysis of Relational Database Schemas,Phil McMinn; Chris J. Wright; Colton J. McCurdy; Gregory M. Kapfhammer,Published in: IEEE Transactions on Software Engineering (Volume: 45, Issue: 5, 01 May 2019),Date of Publication: 27 December 2017
21. M. Ohyver, J. V. Moniaga, I. Sungkawa, B. E. Subagyo, and I. A. Chandra,''The comparison firebase realtime database and MySQL database performance using Wilcoxon signed-rank test,'' Procedia Comput. Sci., vol. 157,pp. 396–405, Jan. 2019.
22. L. Bienvenu and R. Downey, ''On low for speed oracles,'' J. Comput. Syst. Sci., vol. 108, pp. 49–63, Mar. 2020.
23. P. Senellart, L. Jachiet, S. Maniu, and Y. Ramusat, ''ProvSQL: Provenance and probability management in postgreSQL,'' Proc. VLDB Endowment, vol. 11, no. 12, pp. 2034–2037, Aug. 2018.
24. Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments, Mahmoud Moustafa Eyada; Walaa Saber; Mohammed M. El Genidy; Fathy Amer,Published in: IEEE Access (Volume: 8) Date of Publication: 15 June 2020
25. Y.-S. Kang, I.-H. Park, J. Rhee, and Y.-H. Lee, ''MongoDB-based repository design for IoT-generated RFID/Sensor big data,'' IEEE Sensors J., vol. 16, no. 2, pp. 485–497, Jan. 2016.
26. B. Maity, S. Sen, and N. C. Debnath, ''Retracted: Challenges of implementing data warehouse in MongoDB environment,'' J. Fundam. Appl. Sci., vol. 10, no. 4S, pp. 222–228, 2018.
27. M. Kalske, N. Mäkitalo, and T. Mikkonen, ''Challenges when moving from monolith to microservice architecture,'' in Current Trends in Web Engineering (Lecture Notes in Computer Science), vol. 10544, I. Garrigós and M. Wimmer, Eds. Cham, Switzerland: Springer, 2018, doi: 10.1007/978- 3-319-74433-9_3
28. J. Ghofrani and A. Bozorgmehr, ''Migration to microservices: Barriers and solutions,'' in Applied Informatics, H. Florez, M. Leon, J. M. Diaz-Nafria, and S. Belli, Eds. Cham, Switzerland: Springer, 2019, pp. 269–281.
29. B. Terzić, V. Dimitrieski, S. Kordić, G. Milosavljević, and I. Luković, ''Development and evaluation of microbuilder: A model-driven tool for the specification of rest microservice software architectures,'' Enterprise Inf. Syst., vol. 12, nos. 8–9, pp. 1034–1057, 2018.
30. O. Al-Debagy and P. Martinek, ''A comparative review of microservices and monolithic architectures,'' in Proc. IEEE 18th Int. Symp. Comput. Intell. Informat. (CINTI), Nov. 2018, pp. 149–154.
31. A. de Camargo, I. Salvadori, R. D. S. Mello, and F. Siqueira, ''An architecture to automate performance tests on microservices,'' in Proc. 18th Int. Conf. Inf. Integr. Web-Based Appl. Services, Nov. 2016, pp. 422–429.
32. V. Lenarduzzi, F. Lomio, N. Saarimäki, and D. Taibi, ''Does migrating a monolithic system to microservices decrease the technical debt?'' J. Syst. Softw., vol. 169, Nov. 2020, Art. no. 110710.
33. A. Poniszewska-Marańda and E. Czechowska, ''Kubernetes cluster for automating software production environment,'' Sensors, vol. 21, no. 5, p. 1901, 2021.
34. V. Lenarduzzi and O. Sievi-Korte, ''On the negative impact of teamindependence in microservices software development,'' in Proc. 19th Int.Conf. Agile Softw. Develop., Companion, New York, NY, USA, May 2018,pp.
35. F. Ramin, C. Matthies, and R. Teusner, ''More than code: Contributions in scrum software engineering teams,'' in Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng. Workshops, New York, NY, USA, Jun. 2020, pp. 137–140.
36. H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine, and S. Khan,''A survey of distributed data stream processing frameworks,'' IEEEAccess, vol. 7, pp. 154300–154316, 2019.
37. https://pretius.com/blog/benefits-of-microservices/
38. https://thenewstack.io/microservices/microservices-vs-monoliths-an-operational-comparison/
39. https://www.simform.com/blog/monoliths-vs-microservices/
40. Performance Analysis of Not Only SQL Semi-Stream Join Using MongoDB for Real-Time Data Warehousing,Erum Mehmood; Tayyaba Anees,Published in: IEEE Access (Volume: 7),Date of Publication: 17 September 2019
41. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/rabbitmq-event-bus-development-test-environment
42. https://cloudificationzone.com/2021/11/01/distributed-caching-with-redis/
43. Guadalupe Ortiz; José Antonio Caravaca; Alfonso García-de-Prado; Fràncisco Chavez de la O; Juan Boubeta-Puig "Real-Time Context-Aware Microservice Architecture for Predictive Analytics and Smart Decision-Making",Published in: IEEE Access (Volume: 7) ,18 December2019,ISSN:2169-3536,Publisher: IEEE
44. Grzegorz Blinowski; Anna Ojdowska; Adam Przybyłek "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation",Published in: IEEE Access (Volume: 10),Date of Publication: 18 February 2022 ,ISSN: 2169-3536,Publisher: IEEE