Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2022, 9(6):87-95

Research Article ISSN: 2394 - 658X

The Role of NoSQL in Microservices Architecture: Enabling Scalability and Data Independence

Mahesh Kumar Goyal[1], Rahul Chaturvedi[2]

[1]maheshgoyal0718@gmail.com, Google LLC
[2]r.chaturvedi2302@gmail.com, Gilead Sciences

ABSTRACT

Microservices architecture has completely changed how software systems are architected and are being constructed and the advantages are enhanced agility, scalability, and resilience. In this paper, we study the critical role NoSQL databases play in driving microservices into the success they enjoy today, with respect to scalability and data independence. Unlike relational databases, NoSQL databases have different data models and are distributed which suit the principles of microservices and each service can pick the most suitable database for the specific data it needs. The purpose of this polyglot persistence approach, along with the sharding and replication inherent to NoSQL, allows companies to create highly flexible and high performing apps. We take a look at various types of NoSQL databases: key value stores, document databases, wide column stores and graph databases, looking at pros and cons from the microservices point of view. In addition, it discusses how the NoSQL databases solve problems such as data consistency, distributed transaction, and schema change in a distributed database system.

The paper illustrates how NoSQL is utilized by organizations to achieve data independence and fault tolerance, and to optimize performance, through case studies and examples. The paper examines the operational complexities and the required skill set to manage a polyglot persistence environment and reaches a conclusion that, though complicated, strategic adoption of NoSQL databases are a key enabler for organizations who seek a return on implementing microservices architecture. The future promises more synergy and innovation in the form of more resilient, scalable, and data driven applications of the NoSQL and Microservices.

Keywords: NoSQL, Microservices, Scalability, Data Independence, Distributed Systems, Polyglot Persistence, CAP Theorem, Eventual Consistency, Sharding, Replication, Data Modeling, Database, Cloud-native, Architecture, Key-Value Stores, Document Databases, Wide-Column Stores, Graph Databases, Performance, Fault Tolerance, Data Consistency, Distributed Transactions, Schema Evolution, Cloud Computing

INTRODUCTION

The Rise of Microservices Architecture

There has been a massive shift within the software industry moving away from traditional monolithic architectures to a more modular approach called microservices. A monolithic application is a single unit having all components and are tightly coupled; a microservice architecture is a set of tightly coupled services focusing on specific business capabilities. There are several benefits from this architectural style, for example faster development cycle leads to higher agility, independent scaling of services towards better scalability, and option to choose whatever technologies you want. However microservice systems bring some complexities, most notably with regard to data consistency and distributed transaction management, since each service owns its own data. [1,2,3]

Introduction to NoSQL Databases

For the developers who are struggling between traditional relational database management systems (RDBMS) and NoSQL, are NoSQL databases a compelling choice? On the contrary to the behavior of RDBMS which includes schema commitment and adheres to ACID (Atomicity, Consistency, Isolation, Durability) properties, NoSQL databases present a more relaxed way of dealing with data. There are key-value stores, document databases, wide column stores and graph databases listed under the typical data model which are built for different use cases. A property of many NoSQL databases is that they are distributed and how they handle the CAP theorem, which states

that it is impossible to have a distributed system simultaneously providing Consistency, Availability, and Partition Tolerance. [4,5]

The Synergy Between NoSQL and Microservices

In this research, we look at how microservices architectures and NoSQL databases can leverage off each other. Because of its scalability, data independence and fault tolerance, NoSQL databases are particularly well suited to microservices which are distributed and independent. A polyglot persistence approach frees us from the pain of choosing a right database for the job, and by allowing each microservice to pick the database that fits with its specific requirements best, it allows us to optimize both performance and flexibility again. This combination allows organizations to leverage an application that is highly scalable, resilient and adaptable that can then respond rapidly to changing business requirements. [6,7,8]

UNDERSTANDING MICROSERVICES ARCHITECTURE

Principles of Microservices Architecture

Microservices architecture can be constructed with a number of core principles to guide the design and development of service. These principles offer modularity, independence and resilience.[9]

Single Responsibility Principle: Each microservice should have a single responsibility, enlarging only a single, well defined business capability. In this case, this is a good follow up from object oriented design, as this principle ensures that services stay focused and within reason. By following this principle, teams have the option to develop, deploy, and scale services with low risk to the rest of the application. This also keeps it easier to understand and maintain each service overtime [9,10].

Decentralized Governance: The Microservices architecture advocates decentralization of technology governance. That is, the development teams have the freedom to select the most appropriate programming languages, frameworks, databases, etc., as per requirements of the service they are building. In doing so, teams have the freedom to optimize for performance, scalability, and developer productivity and innovation and agility. It also requires a mindful approach in regards to the interoperability and communication inter service. [11]

Independent Deployability: A core aspect of the micro service manifesto and one of the most critical foundational technologies is independent deployability. It helps in releasing quickly and at the same time decreases the risk in monolithic large releases. Without a redeployment of the whole application, changes to one service can be rolled out making it easy for teams to iterate quickly and respond quickly to changing business needs. It also allows for independent deployability for continuous delivery and continuous integration practices to function. [12]

Fault Isolation: Microservices are fault tolerant. It should not be possible for one service to take down the whole application if one fails. Techniques like circuit breakers, timeouts and retries help isolate this fault such that cascading failure is avoided and the system is able to keep functioning when individual serviços are having issues. Building highly available and reliable applications is critical and this resilience is vital to it.[13,14]

Data Management Challenges in Microservices

While microservices bring in a lot of advantages, their use also complicates things, especially on the data management fronts. While microservices are distributed, they present inherent challenges of data consistency, transaction management and schema evolution.

Data Consistency Across Services: In a microservices architecture, maintaining data consistency across multiple, independent services each with its own database is difficult. Data has to be synchronized on different services but ensuring that data persists consistently across different services requires careful planning and implementing appropriate synchronization mechanisms. The lack of a central database makes it really difficult to enforce traditional ACID properties across the whole application. [15]

Distributed Transactions: Implementing Transactions Distributedly, transactions are complicated and often must be implemented differently than they appear in the classic two-phase commit protocols. Performance bottlenecks and the availability of the system as a whole are reduced by distributed transactions. In a microservices environment we often need to use patterns like Sagas, which we will talk about later, to manage transactions. [16]

Data Duplication and Synchronization: For Example, to achieve data independence and performance optimization, microservices duplicate data that are relevant to its domain. Yet, there is a difficulty in keeping these data in sync with various services. Propagation of changes made in one service to other services with their copy of the data is a complex and error prone process. [17]

Schema Evolution: Data Schema Evolution Schema evolution in a microservices architecture consists of each service evolving independently and includes its own data schema. While managing schema changes, ensuring that it does not disrupt other services, requires careful planning and coordination. Versioning schemas, and proactively implementing backward compatibility mechanisms are very important, for guaranteeing that services can seamlessly continue to interoperate throughout their evolution. [18]

NoSQL DATABASES: A DEEP DIVE

Types of NoSQL Databases

NoSQL Databases There are NoSQL databases each for specific use cases. There is important data to be collected to understand the characteristics of each type in order to select the most suitable database for the specific microservice. [19,20]

Key-Value Stores: Key-value stores are NoSQL databases of very simple type which store data as pairs of keys and corresponding values. They are extremely efficient for storing and querying data based upon some unique key. The most popular examples of key–value stores are Databases such as Redis and Memcached, which are built for caching, session management and trivial set data structures, where the read/write performance should be of the top notch. Following image shows the Key Value NoSQL database [21]

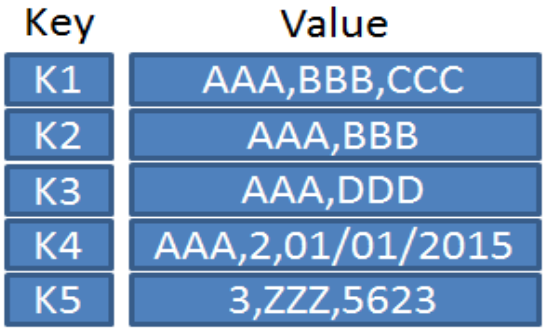

Figure 1

Document Databases: Document databases like MongoDB, Couchbase, and RavenDB are flexible, self describing, document oriented databases used to store data in self describing documents such as JSON and BSON. This schema less is convenient, and enables evolution of data structure over time. Applications with frequently changing data models, such as content management systems, user profiles, and product catalogs, fit well for document databases. Following image shows the example of a document database. [22]

```
first_name: "Mary",
last_name: "Jones",
cell: "516-555-2048",
city: "Long Island",
year_of_birth: 1986,
location: {
        type: "Point",
        coordinates: [-73.9876, 40.7574]
},
profession: ["Developer", "Engineer"],
apps: [
 { name: "MyApp",
   version: 1.0.4 },
 { name: "DocFinder",
   version: 2.5.7 }
],
cars: [
   { make: "Bentley",
     year: 1973 },
   { make: "Rolls Royce",
     year: 1965 }
 ]
```

Figure 2

Wide-Column Databases: Wide column databases such as Cassandra and HBase store data in tables with rows and columns, but the columns can differ from row to row, that is, they are not strictly defined ahead of time, unlike relational databases. The flexible schema is highly efficient to store and retrieve huge volumes of data with different attributes. Typically these are used for cases where write throughput and scalability are important, like time series data, the Internet of Things (IoT) sensor data, and logging. Following image[Fig-3] shows the example of wide column database.[23,24,25,55]

Row A	Column 1	Column 2	Column 3	...
	Value	Value	Value	

Row B	Column 2	Column 3	Column 4	...
	Value	Value	Value	

Figure 3

Graph Databases: Graph databases, such as Neo4j and Amazon Neptune, are good at storing and querying data that is highly interconnected. As a result, they are perfect for graph-like applications in which data is represented as nodes (entities) and edges (relations between entities), that include social networks, recommendation engine or fraud detection. When it comes to data sharing and finding patterns in related data, there is nothing they can't do. Following image [Fig-4] shows the example of wide column database.[26,27,56]

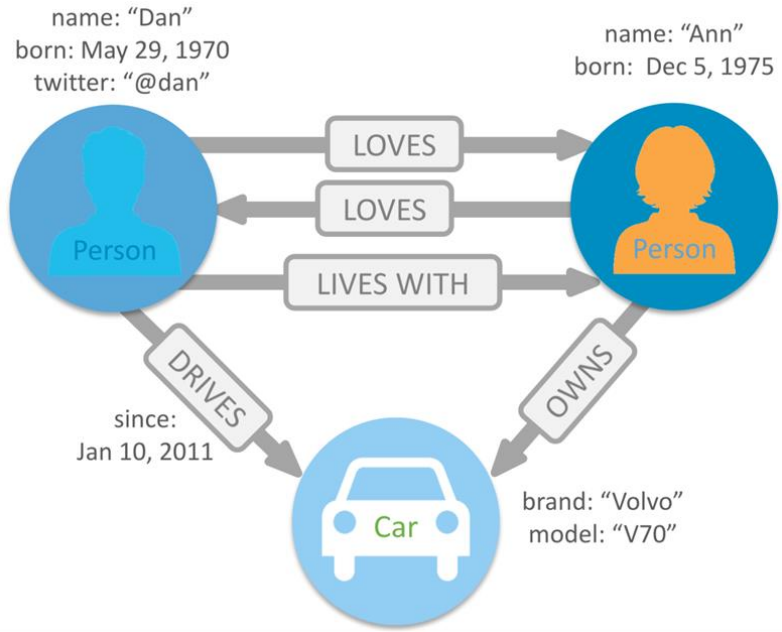

Figure 4

CAP Theorem and its Implications

In fact The CAP Theorem is a fundamental concept in Distributed Systems with implications on NoSQL Database design and selection. If you're considering using a microservices architecture, I'll show you how understanding the trade offs between Consistency, Availability, and Partition Tolerance can help you make an informed decision about which database to use.[28]

Consistency, Availability, Partition Tolerance: The CAP theorem states that a distributed system can only guarantee two out of the following three properties: Consistency (all nodes see the same data, at the same time), Availability (every request gets a response) and Partition Tolerance (the system continues to operate even if some network partitions are lost). Consistency and availability are both desirable properties in a distributed system, and unfortunately, network partitions are unavoidable in practice.[29,30]

Choosing the Right Trade-off: Different NoSQL databases align with different aspects of the CAP theorem. Databases like Cassandra sacrifice consistency for availability and partition tolerance, and conversely, HBase offers this at the cost of partition tolerance. This is all application dependent. For example, a financial application may need to have very strong consistency, and a social media site might rank availability the highest.[31]

Eventual Consistency and its Suitability for Microservices

Thousands of NoSQL databases follow the principle of eventually consistent data, which essentially means that the data will be eventually consistent across all nodes but there may be some time where different nodes have different

views about the data. This is a model often applicable to microservices architectures where services are able to tolerate short lived inconsistencies. The benefits of eventual consistency are higher availability and performance, because services do not need to wait for every node to be updated before completing a response to a request. We will discuss what the eventual consistency model requires in terms of being able to handle possible data conflicts as well as making sure the system converges to a consistent state. [9, 32]

NoSQL AS THE ENABLER OF MICROSERVICES SCALABILITY AND DATA INDEPENDENCE

Scalability Through Sharding and Replication

NoSQL databases are built with scalability in mind, which means they scale horizontally by breaking the data down to be distributed across hardware where the database can collect and process it. Complicated tasks such as sharding and replication make this possible.[33]

Horizontal Scaling with NoSQL: NoSQL databases for horizontal scaling means no database scales beyond the capacities of one server; therefore, NoSQL databases support horizontal scaling by distributing the data between multiple servers, commonly called sharding. A shard has a subset of the data and the database takes care of this data distribution & routing. It allows easy scaling by just adding more servers on the cluster as the cluster will be able to handle increasing amounts of data and traffic. [34,35]

Data Replication for High Availability: Replication entails creating multiple copies of the data and saving them onto multiple servers for high availability purposes. This allows that if some of the servers fail, the system remains available. Different replication strategies used in NoSQL databases, like master slave, master master, peer to peer have varying trade-offs in terms of consistency, performance and complexity. It is the availability and consistency requirement of the application which will dictate the replication strategy that can be chosen.[36]

Data Independence and Polyglot Persistence

Another huge benefit NoSQL databases provide in a microservices architecture is data independence and polyglot persistence. This means each microservice can choose the most suitable database type and schema for its specific needs, whether that's a document store for flexible JSON data, a graph database for complex relationships, or a key-value store for simple caching. Additionally, this independence allows teams to evolve their data models and make schema changes without impacting other services, enabling faster development cycles and more robust system architecture.[37]

Choosing the Right Database for Each Service: With microservices architecture, along with NoSQL databases, you can select the database suitable for that service. For instance, if you are building a user session service, for fast access you can pick up a key value store like redis, if it’s a product data service you might use a document database like mongodb for flexibility in schema. And this freedom to choose the right tool for the job optimizes performance and simplifies development. [31,38]

Schema Flexibility and Evolution: NoSQL databases, in particular document and wide-column stores, have flexible schemas which are easy to evolve over time. In a microservices environment services are developed and updated independently so this is very important. Unlike traditional databases, new attributes can be added to documents or columns, and no schema migration is required — new services can be built to scale to new business requirements without affecting other parts of the system.[39]

Fault Tolerance and Resilience

NoSQL databases are by design fault tolerant and resilient which is critical in a distributed microservices world.[40]

Handling Data Store Failures: 'Easily failing' is a design principle of NoSQL databases. They can continue to operate by means of replication and automatic failover mechanisms even if some of the servers in the cluster become unavailable. This puts microservices that are individual, and the system as a whole, so that they remain resilient to failure. These databases are inherently more robust because they are distributed.[41]

Data Replication and Disaster Recovery: In addition to offering availability, data replication is of great importance to disaster recovery. Organizations maintain multiple copies of the data at different locations to allow quick recovery from data center outage, or some other catastrophic events. The replication and backing options are varied in NoSQL databases that can support robust disaster recovery strategies to ensure that business operations will continue with some unforeseen circumstances. [41, 42]

Performance Optimization in NoSQL

NoSQL performance optimization is important because without it microservices will not be able to endure high traffic loads and provide a smooth user experience.[43]

Data Modeling for Performance: As such, keys are used to achieve optimal performance in NoSQL databases; however, effective data modeling is critical for that performance to be realized. Unlike relational databases that normally tend to normalize data, NoSQL databases need denormalization in order to minimize need for joins, which are expensive in a distributed environment. If you want to choose the right data model, you have to understand the access patterns of a given application and then design or craft your own data structures around it.[44]

Indexing Strategies: Proper indexing is important for fast query performance in any database and NoSQL databases can’t take a pass on this either. Each NoSQL database supports different indexing options, and we can

decide the right index based on the application's query patterns. Proper indexing can save on query latency and improve the whole system as a whole. Indexing can be done on the full message field or on a combination of the receive time field and message content field, but this must be balanced between index size, write performance and query performance.[45]

CASE STUDIES AND EXAMPLES

E-commerce Platform Using Microservices and NoSQL

Hypothetical e commerce platform built using microservices architecture. For product catalog service, the platform can use the document database such as MongoDB where the schema for product catalog can evolve flexibly as new product attributes are added. The shopping cart service could be a Redis, as a key value store with fast access to session data. A wide column store like Cassandra is used by the order management service due to its ability to have high write throughput and store lots of order data. This example shows how the different NoSQL databases can be picked depending on what each microservice really needs.[46]

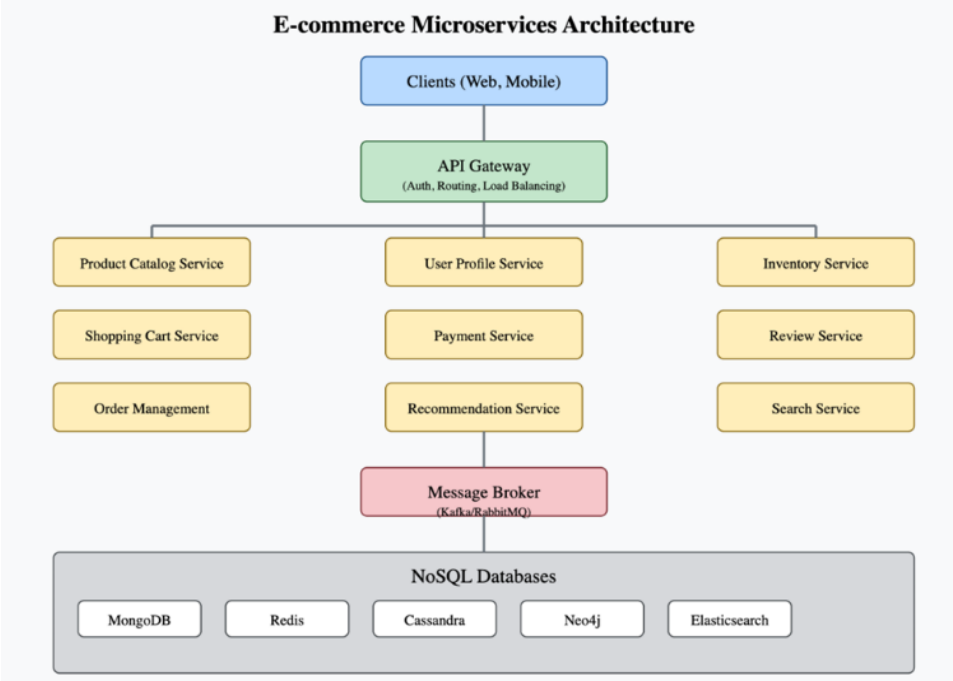

Figure 5

This [Figure 5] is a SOA based e-commerce architecture with a highly distributed NoSQL database system with features of microservices architecture. Fundamentally speaking, it has an API Gateway as the central point for the client requests where basic functions like security, route, and Service-Load balancer are performed. These are the core services (Product Catalog, Shopping Cart, Order Management, User Profile) and the basic services (Payment, Recommendation, Inventory, Review & Rating) that are all working in isolation levels with its own NoSQL database. MongoDB as a master data storage works with products, users' accounts, and inventory; specialized databases are used as followers and operate as follows: Redis for shopping carts as an in-memory highly-accessible database, Cassandra for orders as a highly-scalable writing type of database, Neo4j for recommendation as a graph database, and Elasticsearch for search functionality.

The architecture establishes a message broker (Kafka/RabbitMQ) where services can communicate asynchronously with an event driven mechanism which leads to high service independence and scalability. This design targets high availability, fault tolerance and horizontal scalability with possible eventual

CHALLENGES AND CONSIDERATIONS

Data Consistency and Transaction Management

For the microservices, although NoSQL databases provide us with a lot of advantages they come with the challenge of data consistency and transaction management.[48]

Implementing Sagas for Distributed Transactions: The Saga pattern enables us to manage Distributed Transactions where each service in a distributed architecture has an atomic task running. A Saga is a sequence of local transactions, each that's attempting to update its own database. In case of a failure on any transaction,

compensating transactions take place to undo the previous operations. This gives us sagas that can help maintain data consistency across services without the overhead of distributed two phase commit and complexity.[49]

Handling Data Inconsistencies: In an eventually consistent system, traversing such a system might mean that, at some point in time, data is not available everywhere. These potential inconsistencies have to be handled by the developers when we are designing the applications. It may involve conflict resolution strategies, read repair mechanisms, or even giving up on some data for a short while being stale. Important consideration must be given to the application consistency requirements. [41,48,50]

Operational Complexity

Managing a microservices architecture with different NoSQL databases can certainly be complex.

Monitoring and Management of Multiple Databases: Robust monitoring and management tools are required in a polyglot persistence environment that operates (multiple) databases. The particular requirements for monitoring and procedures under each database type vary. In order to know the health and performance of all their databases, and thus be able to solve problems and plan capacity efficiently, organizations need to invest in tools and processes that provide a single pane of glass.[51]

Skill Set Requirements: Skillset pros and cons Developers or administrators tasked with managing multiple NoSQL databases are likely to have a bunch of skills to wrangle. To support teams in highly effective and high throughput usage of the database types in the theme cloud, teams need to know the specific features, operational procedures, and performance tuning techniques for each database type. The organization may then involve specialized training and hiring to make sure that the organization has the required expertise to handle its polyglot persistence environment.[52]

Security in a NoSQL Environment

In fact, securing NoSQL databases is as important as securing traditional relational databases. As a result, standard security features and best practices for each database are included in NoSQL. To protect sensitive data organizations should implement such authentication, authorization, and encryption mechanisms. Regular security audits and system wide vulnerability checks should be done to secure the NoSQL environment. We shall therefore have to define access control policies with care with regard to the least privilege principle, so as to limit access to sensitive information on a need to know basis.[53]

CONCLUSION

Summary of Key Findings

The paper has shown that NoSQL databases prove crucial to making microservices architecture scalable and data-independent. We have looked into features of the NoSQL databases including distributed nature, flexible schema and supporting horizontal scaling and how these fit with the microservices design principles. Organizations can achieve agility, performance, and resilience not practical with monolithic architecture and relational databases by allowing each service to pick the most suitable database for its own particular needs. This paper discussed the advantages of polyglot persistence, pointing out as well data consistency, distributed transactions, and operational complexity issues.

Future Trends in NoSQL and Microservices

NoSQL databases and Microservices will continue to be a fast moving field. Further advancements will be seen in automated scaling, self healing database and better tools to manage polyglot persistence environments. The growing use of serverless computing and the Edge Computing phenomenon will certainly have an impact on the development and deployment of microservices and NoSQL databases. Because the data volumes will continue to increase and the application requirements will only become more demanding, the integration between NoSQL and microservices is just getting started.[54]

Final Thoughts: Embracing the Right Tools for the Job

The right combination of NoSQL databases and microservices architecture is the right tool for the job when you want to build modern, scalable, and resilient applications. A polyglot persistence approach, coupled with choosing the right database allows us to embrace the diversity of our data, while at the same time implementing it according to proven best practices.

REFERENCES

[1]. Tapia, Freddy, et al. "From monolithic systems to microservices: A comparative study of performance." Applied sciences 10.17 (2020): 5797.

[2]. Baškarada, Saša, Vivian Nguyen, and Andy Koronios. "Architecting microservices: Practical opportunities and challenges." Journal of Computer Information Systems (2020).

[3]. Santos, Nuno, and António Rito Silva. "A complexity metric for microservices architecture migration." 2020 IEEE international conference on software architecture (ICSA). IEEE, 2020.

[4]. Sahatqija, Kosovare, et al. "Comparison between relational and NOSQL databases." 2018 41st international convention on information and communication technology, electronics and microelectronics (MIPRO). IEEE, 2018.
[5]. Frank, Lars, et al. "The cap theorem versus databases with relaxed acid properties." Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication. 2014.
[6]. Viennot, Nicolas, et al. "Synapse: a microservices architecture for heterogeneous-database web applications." Proceedings of the tenth european conference on computer systems. 2015.
[7]. Liu, Zhen Hua, et al. "Closing the functional and performance gap between SQL and NoSQL." Proceedings of the 2016 international conference on management of data. 2016.
[8]. Holbeche, Linda. "Designing sustainably agile and resilient organizations." Systems Research and Behavioral Science 36.5 (2019): 668-677.
[9]. Li, Shanshan, et al. "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review." Information and software technology 131 (2021): 106449.
[10]. Nadareishvili, Irakli, et al. Microservice architecture: aligning principles, practices, and culture. " O'Reilly Media, Inc.", 2016.
[11]. Zimmermann, Olaf. "Microservices tenets: Agile approach to service development and deployment." Computer Science-Research and Development 32 (2017): 301-310.
[12]. Aksakalli, Işıl Karabey, et al. "Deployment and communication patterns in microservice architectures: A systematic literature review." Journal of Systems and Software 180 (2021): 111014.
[13]. Boucher, Sol, et al. "Putting the" micro" back in microservice." 2018 USENIX Annual Technical Conference (USENIX ATC 18). 2018.
[14]. Miller, Loïc, et al. "Towards secure and leak-free workflows using microservice isolation." 2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR). IEEE, 2021.
[15]. Furda, Andrei, et al. "Migrating enterprise legacy source code to microservices: on multitenancy, statefulness, and data consistency." Ieee Software 35.3 (2017): 63-72.
[16]. Salah, Tasneem, et al. "The evolution of distributed systems towards microservices architecture." 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST). IEEE, 2016.
[17]. Smid, Antonin, Ruolin Wang, and Tomas Cerny. "Case study on data communication in microservice architecture." Proceedings of the Conference on Research in Adaptive and Convergent Systems. 2019.
[18]. Bushong, Vincent, et al. "On microservice analysis and architecture evolution: A systematic mapping study." Applied Sciences 11.17 (2021): 7856.
[19]. Gupta, Adity, et al. "NoSQL databases: Critical analysis and comparison." 2017 International conference on computing and communication technologies for smart nation (IC3TSN). IEEE, 2017.
[20]. Nayak, Ameya, Anil Poriya, and Dikshay Poojary. "Type of NOSQL databases and its comparison with relational databases." International Journal of Applied Information Systems 5.4 (2013): 16-19.
[21]. Osemwegie, Omoruyi, et al. "Performance benchmarking of key-value store NoSQL databases." International Journal of Electrical and Computer Engineering (IJECE) 8.6 (2018): 5333-5341.
[22]. Mason, Robert T. "NoSQL databases and data modeling techniques for a document-oriented NoSQL database." Proceedings of informing science & IT education conference (InSITE). Vol. 3. No. 4. 2015.
[23]. Mehra, Raghav, Nirmal Lodhi, and Ram Babu. "Column based NoSQL database, scope and future." International Journal of Research and Analytical Reviews 2.4 (2015): 105-113.
[24]. Hewitt, Eben. Cassandra: the definitive guide. " O'Reilly Media, Inc.", 2010.
[25]. Vora, Mehul Nalin. "Hadoop-HBase for large-scale data." Proceedings of 2011 International Conference on Computer Science and Network Technology. Vol. 1. IEEE, 2011.
[26]. Atemezing, Ghislain Auguste. "Empirical Evaluation of a Cloud-Based Graph Database: the Case of Neptune." Knowledge Graphs and Semantic Web: Third Iberoamerican Conference and Second Indo-American Conference, KGSWC 2021, Kingsville, Texas, USA, November 22–24, 2021, Proceedings 3. Springer International Publishing, 2021.
[27]. Miller, Justin J. "Graph database applications and concepts with Neo4j." Proceedings of the southern association for information systems conference, Atlanta, GA, USA. Vol. 2324. No. 36. 2013.
[28]. Shim, Simon SY. "Guest editor's introduction: The cap theorem's growing impact." Computer 45.02 (2012): 21-22.
[29]. Vogels, Werner. "Eventually Consistent: Building reliable distributed systems at a worldwide scale demands trade-offs? between consistency and availability." Queue 6.6 (2008): 14-19.
[30]. Dragojević, Aleksandar, et al. "No compromises: distributed transactions with consistency, availability, and performance." Proceedings of the 25th symposium on operating systems principles. 2015.
[31]. Lourenço, João Ricardo, et al. "Choosing the right NoSQL database for the job: a quality attribute evaluation." Journal of Big Data 2 (2015): 1-26.

[32]. O'Connor, Rory V., Peter Elger, and Paul M. Clarke. "Continuous software engineering—A microservices architecture perspective." Journal of Software: Evolution and Process 29.11 (2017): e1866.
[33]. Sapar, Nazir. "SOLUTIONS FOR BUILDING HIGH-AVAILABILITY WITH NOSQL DATABASES." Интернаука 16-3 (2021): 83-85.
[34]. Naskos, Athanasios, Anastasios Gounaris, and Panagiotis Katsaros. "Cost-aware horizontal scaling of NoSQL databases using probabilistic model checking." Cluster Computing 20 (2017): 2687-2701.
[35]. AGARWAL, SARTHAK. Performance Analysis of Spatial Queries and Routing in NoSQL databases. Diss. International Institute of Information Technology, Hyderabad, 2019.
[36]. Pankowski, Tadeusz. "Consistency and availability of Data in replicated NoSQL databases." 2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE). IEEE, 2015.
[37]. Sadalage, Pramod J., and Martin Fowler. NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Pearson Education, 2013.
[38]. Khazaei, Hamzeh, et al. "How do I choose the right NoSQL solution? A comprehensive theoretical and experimental survey." Big Data & Information Analytics 1.2&3 (2016): 185-216.
[39]. Scherzinger, Stefanie, Meike Klettke, and Uta Störl. "Managing schema evolution in NoSQL data stores." arXiv preprint arXiv:1308.0514 (2013).
[40]. Ganesan, Aishwarya, et al. "Redundancy does not imply fault tolerance: Analysis of distributed storage reactions to file-system faults." ACM Transactions on Storage (TOS) 13.3 (2017): 1-33.
[41]. Abed, Amira Hassan. "Recovery and concurrency challenging in big data and NoSQL database systems." International Journal of Advanced Networking and Applications 11.4 (2020): 4321-4329.
[42]. Abadi, Aharon, et al. "Holistic disaster recovery approach for big data NoSQL workloads." 2016 ieee international conference on big data (big data). IEEE, 2016.
[43]. Dipietro, Salvatore. "Performance modelling and optimisation of NoSQL database systems." ACM SIGMETRICS Performance Evaluation Review 47.3 (2020): 10-13.
[44]. Atzeni, Paolo, et al. "Data modeling in the NoSQL world." Computer Standards & Interfaces 67 (2020): 103149.
[45]. Qader, Mohiuddin Abdul, Shiwen Cheng, and Vagelis Hristidis. "A comparative study of secondary indexing techniques in LSM-based NoSQL databases." Proceedings of the 2018 International Conference on Management of Data. 2018.
[46]. Nalla, Lakshmi Nivas, and Vijay Mallik Reddy. "Comparative Analysis of Modern Database Technologies in Ecommerce Applications." International Journal of Advanced Engineering Technologies and Innovations 1.2 (2020): 21-39.
[47]. Nicoara, Daniel, et al. "Hermes: Dynamic Partitioning for Distributed Social Network Graph Databases." EDBT. 2015.
[48]. Al Fatah, Jabir, and Liaquath Chowdhury. "Consistency Issues on NoSQL Databases: Problems with Current Approaches and Possible Solution (s)." Update 4.4 (2016): 3.
[49]. Ramzan, Shabana, et al. "Challenges in NoSQL-based distributed data storage: a systematic literature review." electronics 8.5 (2019): 488.
[50]. Diogo, Miguel, Bruno Cabral, and Jorge Bernardino. "Consistency models of NoSQL databases." Future Internet 11.2 (2019): 43.
[51]. Grolinger, Katarina, et al. "Data management in cloud environments: NoSQL and NewSQL data stores." Journal of Cloud Computing: advances, systems and applications 2 (2013): 1-24.
[52]. Gurcan, Fatih, and Nergiz Ercil Cagiltay. "Big data software engineering: Analysis of knowledge domains and skill sets using LDA-based topic modeling." IEEE access 7 (2019): 82541-82552.
[53]. Gupta, Neha, and Rashmi Agrawal. "NoSQL security." Advances in computers. Vol. 109. Elsevier, 2018. 101-132.
[54]. McGrath, Garrett, and Paul R. Brenner. "Serverless computing: Design, implementation, and performance." 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). IEEE, 2017.
[55]. openlogic “Guide to Open Source Wide Column Databases” https://openlogic.com/blog/guide-open-source-wide-column-databases
[56]. Jatin Sharma. “Graph Databases vs Relational Databases: What and why?” https://j471n.in/blogs/graph-databases-vs-relational-databases