Feature Article

Framework for Interaction Between Databases and Microservice Architecture

Mohamed El Kholy
Pharos University in Alexandria

Ahmed El Fatatry
Alexandria University

***Abstract*—Migrating from monolithic applications to Microservices architecture brings about a number of challenges. Splitting and sharing databases among Microservices is a case in point. In this article, we present a framework for managing the splitting of databases among Micorservices. The proposed solution maintains data consistency and independent deployment of Microservices while enhancing the solution to latency problems. The evaluation results show enhancement in performance as a result of using the proposed framework.**

Flexibility is a key design requirement for business software.[1] The Micorservices architecture is a way of building flexible software by independently developing and deploying small scale software components that can be integrated.[2] Applying the concept of independent deployment makes it possible to change or replace individual services without disrupting the whole system.[3]

While the idea of Microservices brings about a number of benefits in terms of flexibility, however, handling database within a system of Microservices is a challenging issue.[4] In a typical monolithic application, the database is usually separated from the application process according to the MVC model. In a system where data is distrusted among services, data consistency has to be handled. Any solution should not affect response time negatively. This is especially important in data-driven web applications that deal with large sizes of data.[5] There is a need for an efficient and coherent solution for managing databases in Microservices context.

Several approaches have been proposed for dealing with this issue. One approach suggests keeping the data stored in one database and managing service interaction with the monolithic database.[6] Having a database per service is another way to deal with data in Microservices architecture.[7]

This article presents a framework for managing database interaction in Microservices architecture. Managing database for microservices

Digital Object Identifier 10.1109/MITP.2018.2889268
Date of current version 11 September 2019.

Published by the IEEE Computer Society

architecture (MDMA) allows each Microservice to control its own data, according to its functionality and share data from databases associated with other services. The main contribution of the framework is facilitating database transactions with Microservices without the need for a central node. The transaction is performed via message transfer between any two collaborating Microservices. The proposed framework was compared with another solutions in the literature especially Synaps framework. The experimental results proved that MDMA achieved all the required functionality for splitting database over different services without the need for a central node. Moreover MDMA showed smaller response time and smaller size of message transfer when compared with another solution.

This article is organized as follows: the "Problem Definition and Approaches For Solution" section introduces the problem definition and different solution approaches reported in the literature. The "Proposed Framework (MDMA)" section presents the proposed framework. The paper is evaluated in the "Evaluation" section. Finally, the "Conclusion" section discusses the conclusions and future work.

PROBLEM DEFINITION AND APPROACHES FOR SOLUTION

Problem Definition

- After splitting a monolithic application into smaller Microservices, the resulting Microservices need a mechanism to manage the interaction of each individual service with its data.[6,7] Implementing business transactions usually involves implementing queries that join data related to different services.[8] A key point in the Microservices paradigm is the concept of independent deployment.[9,10] Hence, traditional approaches that solve data consistency are not applicable in Microservices.[7,11] For instance, Nonstop Structured Query Language manages data which are distributed over several connected machines.[12,13] However, it requires all data to be of Structured Query Language type, and all the databases should be deployed before running Nonstop Structured Query Language.
- As the number of Microservices grows, the communication overhead increases.[14,15] There is a need for a design to optimize inter-service communication.

Approaches for Solution

This section discusses different solutions reported in the literature for the proposed problem.

First Approach: Cloning Database This approach creates a clone of the original database for each service.[16] Each Microservice interacts with its database in an isolated manner. The cloning approach requires more resources to store the data. Scaling up any part of the database affects all other clones. The database of each Microservice includes all tables and queries. However, many tables may not be associated with the business logic of that Microservice. Maintaining data cohesion is another challenge. Each updated field should be broadcasted to all other clones. Any new instance should also be created in all tables in all clones.

Second Approach Private Database (Database per Service) In the second approach, each Microservice has its own private database that contains data associated with its business functionality.[2,7] For instance, Structured Query Language and non-Structured Query Language databases have different stricter of data and requires different search and access methods. The main challenge is supporting data consistency while joining data from different databases that belong to different services. Another challenge is to support efficient interaction between a Microservice and the data associated with another service. Hence, it is not applicable to use a single application interface (API) to aggregate databases for all services.

Richardson[7] introduced a design pattern that facilitates interaction between different databases that are contained in their Microservices. The private database of each service can be accessed by other Microservices only via the API of the service. In such approach, any update or creation of a new instance of a Microservice's database has no impact on any other services. However, such environment affects the autonomy of the Microservices because each service needs to interact with the (API gateway) to perform its functionality.[8] In addition, such design pattern increases the load of on the application to manage the database.[17] The approach includes

several security issues to authenticate each service with the API. Joining data from distributed heterogeneous databases is a challenging issue.

Viennot *et al.* introduced Synaps as solution for interaction between different microcervices.[12] Synaps supports sharing data between different databases associated with different services. However, synaps depends on a central node that manages the encapsulating and managing data from different services. Such central node creates a single point of failure and has a negative effect of system efficiency.

PROPOSED FRAMEWORK (MDMA)

The proposed solution splits complex large database associated with a monolithic application into smaller simple databases corresponding to the business boundaries of each Microservice.

Advantages of the Proposed Solution

The main advantages of this work over the solutions reported in the literature are as follows:

- Allowing each service to interact with the data associated with its functionality while maintaining the spirit of Microservice for independent deployment. Each service is deployed with its database separately without any dependency on other services. However, after deployment a service publishes the information that allows any other service to interact with its database.
- Enabling cross interaction between a Microservices and databases associated with other Microservices while maintaining efficiency and data consistency. The proposed solution supports Microservices interaction with multiple databases avoiding the dependence on a central API gateway. This leads to more efficient execution because the interaction requests are not gathered to a central node.

Solution Design and Implementation

The proposed solution describes each Microservice as a set of input ports and a set of output ports. The input ports describe the functionality that the service is ready to offer (input requests), as well as its communication schema. The output ports list the required external databases (sent as output from the service to another services).

Service Deployment Each service is deployed with its associated database in an independent manner. After deployment, the service checks the presence of other services and broadcast the following information:

1. name of the service and its location;
2. used transport protocol;
3. used data protocol and the service interface and the available fields to be shared and how to access it.

Then, the service discovers services information to find out the services that matches its output ports and broadcast its input port.

Case Study (Online Book Store)

In order to clarify the proposed approach and compare it with the centralized approach, we built an online book store. The online book store is built once using Synaps approach (reported in literature review) and another time using the proposed solution (MDMA).The book store receives orders from registered clients and supplies them with books after confirmation of payment.

Book Store Design According to MDMA Designing the book store using the Microservices architecture will enhance flexibility and scalability. Five Microservices were built to represent the business boundaries of the book store which are

- register service;
- order service;
- store service;
- payment service;
- shipping service.

The "Registration service" allows clients to be registered on the system. The order service allows the registered customers to order their required books. The "Store service" manages the amount of stored books and its prices and some graphics about the books and samples of included figures. The "Payment service" checks the client's credit limit and whether it covers the ordered book price or not. The "Shipping service" manages the delivery of books to customers. It also informs customers about the expected delivery date.

The proposed approach permits each Microservice to own the data attributes that allows the service to store its data and create new instances.

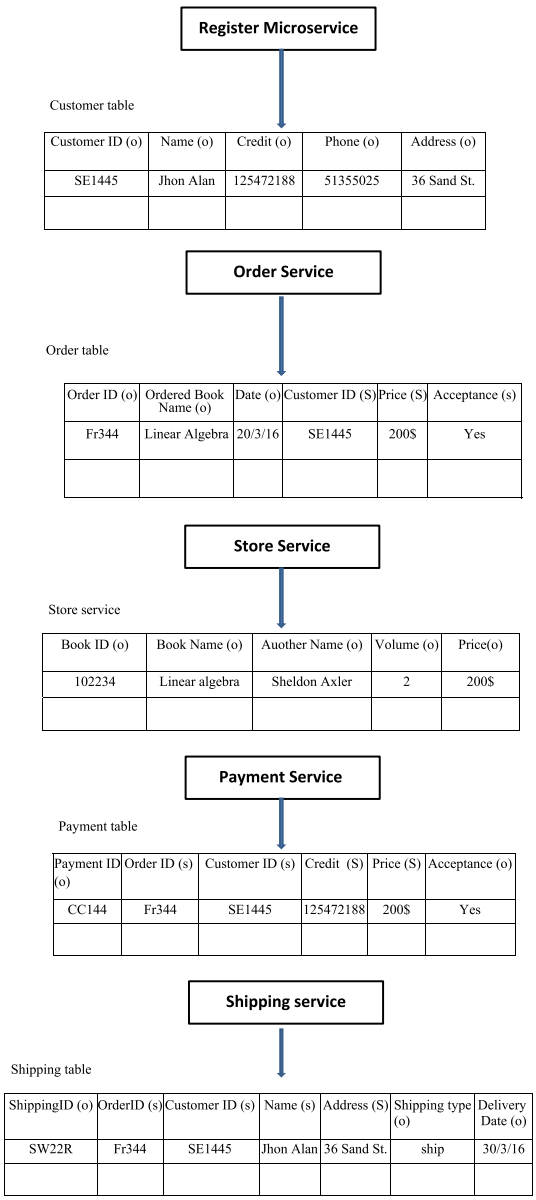

Figure 1. Microservice and interaction with databases. (o) means owned. (s) means shared from another database.

Attributes that are needed to perform interaction with other Microservices are shared from the associated databases. The symbol "O" denotes an owned attribute while the symbol "S" denotes a shared attribute. The owned and shared fields for the five Microservices are shown in Figure 1.

EVALUATION

The proposed framework works in a distributed environment. Hence, the evaluation focuses on the parameters that affect the efficiency of completing the tasks requested by clients in a distributed environment. The evaluation metrics include response time, size of transferred messages, and the scalability of the system against the increasing number of request. The evaluation process measures the enhancement the listed parameters as a result of using the framework. Then, the significance of each enhancement on the overall Microservice application is examined.

Evaluation Metrics

The evaluation process examines the enhancement in efficiency while not breaching the concepts of Microservices such as "independent service deployment." The chosen metrics and the criteria for the choice are listed below.

Response Time The response time measures the time interval between the client request and the application reply. The response time is divided into intervals. First, the message is passed from the client to the application gateway. Second, the request is transferred from the gateway to the application side.[18] In the case of a central node approach, the message is directed to the central node where the request is managed and processed. In the proposed approach, the request is directed from the gateway to the required service which is capable of executing the requested functionality. The message sequence is shown in Figure 2. The enhancement is focused at the application backend where services perform the required calculations and communicate with each other.

Size of Transferred Messages A Microservices application is a network-based application. The less network utilization, the higher application efficiency. The proposed framework is evaluated by calculating the total size of messages for different functionalities offered by the proposed application. Then, the messages size is compared by that of the central node approach which is reported in the related work.

Scalability of the System Against Increasing Number of Request Scalability is a crucial aspect. The response time and the message size are calculated for different numbers of requests. Hence, it is possible to know how well the proposed framework scales up when the number of client requests increases.

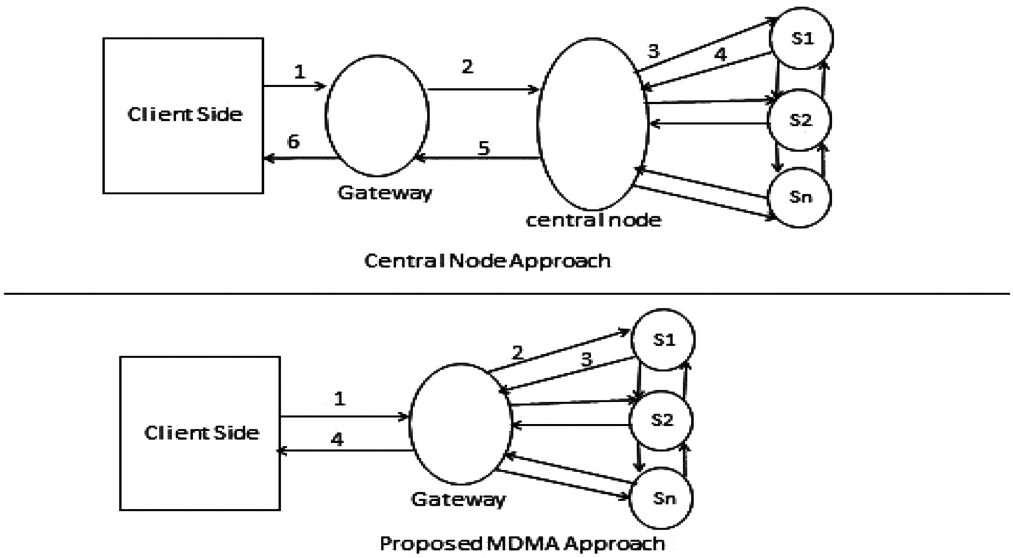

Figure 2. Message path.

Evaluation Process

The online book store application has been used to demonstrate the enhancement in the measured parameters as a result of using the proposed framework. The application has been implemented using C# as a web application in visual studio environment. The hardware used for the evaluation is a super computer that consists of 50 nodes each with two CPUs with 16 cores sharing 64 G.B memory.

The proposed online book store application was implemented twice. First, according to the design of the proposed MDMA framework. Then, the same application was built using central node approach, which has been reported in the literature review.

Book Store Implementation Using MDMA The MDMA framework requires implementing the database associated with each service on the same physical node of the service itself. Hence, each Microservice is able to keep its data and share the required data from other remote databases. There is no central node that manages such interaction between services and different databases. The five proposed Microservices are implemented on five separate nodes of the super computer. The database of each service is implemented locally at the same node of the service. Each database contained 100 000 records structured in SQL database were used. The remaining nodes are used to perform different client requests. The response time and the size of transferred data are measured for a number of client requests.

Book Store Implementation Using Central Node Approach Building the online book store by the central node approach (example Synaps) has been done using a node that manages requests to any database. The five databases have been implemented physically on one central node. Each of the five Microservices was implemented on a separate node. Each request to access any table is passed through the central node. The response time and the size of the transferred data are calculated again for a number of client requests.

Evaluation Results

In the evaluation, we compare MDMA with the central approach using the listed parameters. Theoretically MDMA requires significantly less time compared to central node approach. There is not bottle neck at the central node and each service access its own data without delay of network connections. Table 1 shows the execution time and the size of data transferred for a number of requests.

The comparison shows that MDMA has significantly reduced the execution time as well as the size of the transferred data. The performance

Table 1. Comparison between MDMA and central node approach.

	Execution time		Size of transferred data	
	MDMA	central node	MDMA	central node
100 request	20 ms	34 ms	0.7 kB	1.8 kB
1000 request	85 ms	110 ms	2 kB	15 kB
10 000 request	650 ms	920 ms	17 kB	130 kB
100 000 request	1.1 s	1.75 s	150 kB	1.1 MB
100 000 request	4.3 s	8 s	1.3 MB	9 MB

enhancement of MDMA increases as the number of requests increases. In case of applications that involve large number of client requests, MDMA has an effective impact. In addition, MDMA is more flexible as it does not include a single point of failure. Hence, in case of failure in an individual service, other dependent services may not be affected. In case of deploying new services or replacing one service by another, data transfer is required only once directly after the deployment of the service.

CONCLUSION

This article proposed a solution for data consistency and transaction latency problems in Microservices environment. We presented a framework for managing the splitting of databases among different Microservices. The framework has been evaluated against an increasing number of requests. The evaluation results show enhancement in both response time and total number of messages as a result of using the proposed framework. We believe that using the proposed solution, it is possible to build services with better database independence.

REFERENCES

1. V. Paul de and X. Lai, "Resilience analysis of service-oriented collaboration process management systems," *Serv. Oriented Comput. Appl.*, vol. 12, pp. 25–39, Mar. 2018.
2. J. Thones, "Microservices," *IEEE Softw.*, vol. 32, no. 1, pp. 116–116 Feb. 04 2015. doi: 10.1109/MS.2015.11.
3. C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, "Microservices in practice, Part 1: Reality check and service design," *IEEE Softw.*, vol. 34, no. 1, pp. 91–98, Jan./Feb. 2017. doi: 10.1109/MS.2017.24.
4. S. Larisa, M. Mazzara, F. Montesi, and V. Rivera, "Data-driven workflows for microservices: Genericity in jolie," in *Proc. IEEE 30th Int. Conf. Adv. Inf. Netw. Appl.*, 2016, pp. 430–437.
5. M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *IEEE Comput.*, vol. 40, no. 11, pp. 38–45, Nov. 2007.
6. S. Newman. *Bulding Microservices: Design Fine-Grained Systems*. Newton, MA, USA: O'Reilly, 2016. vol. 4. 971-1-491-95035-7.
7. C. Richardson. A pattern language for microservices, 2014. [Online]. Available at: http://microservices.io/patterns. [Cited: Jun. 18, 2018.]
8. C. Fetzer, "Building critical applications using microservices," *IEEE Security Privacy*, vol. 14, no. 6, pp. 86–89, Nov./Dec. 2016. doi: 10.1109/MSP.2016.129.
9. M. Villamizar *et al.*, "Cost comparison of running web applications in the cloud using monolothic, microservice, and AWs Lambada architecture," *Serv. Oriented Comput. Appl.*, Jun. 2017, vol. 11, pp. 233–247. doi: 10.1007/s11761-017-0208-y.
10. W. D. Hasselbring, "Microservices for scalability: Keynote talk abstract," in *Proc. 7th ACM/SPEC Int. Conf. Perform. Eng.*, 2016, pp. 133–134. doi: 10.1145/2851553.2858659.
11. T. Killalea, "The hidden dividends of microservices," *Commun. ACM*, vol. 59, pp. 42–45, Aug. 2016. doi: 10.1145/2948985.
12. N. Viennot, M. Lecuyer, J. Bell, R. Geambasu, and J. Nieh, "Synapse: A microservices architecture for heterogeneous-database web applications," in *Proc. 10th Euro. Conf. Comput. Syst.*, 2015, Art. no. 21, doi: 10.1145/2741948.2741975.
13. Liam O'. Brien, L. Bass, and P. Merson, "Quality attributes and service-oriented architectures," Dept. Defense, Tech. Rep. CMU/SEI-2005-TN-014, Sep. 2005.
14. R. Heinrich *et al.*, "Performance engineering for microservices: Research challenge and directions," in *Proc. 8th ACM/SPEC Int. Conf. Perform. Eng. Companion*, 2017, pp. 223–226. doi: 10.1145/3053600.3053653.

15. N. Dragoni, M. Mazzara, S. Giallorenzo, F. Montesi, and A. Lluch Lafuente, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. New York, NY, USA: Springer, Sep. 2017, pp. 195–216.
16. G. Granchelli, M. Cardarelli, and P. D. Francesco, "MicroART: A software architecture recovery tool for maintaining microservice-based systems," in *Proc. IEEE Int. Conf. Softw. Archit.*, 2017, pp. 298–302.
17. G. Toffetti, S. Brunner, M. Blochlinger, F. Dudouet, and A. Edmonds, "An architecture for self-managing microservices," in *Proc. 1st Int. Workshop Automat. Incident Manage. Cloud*, Apr. 2015, pp. 19–24. doi: 10.1145/2747470.2747474.
18. M. Antonio, R. Rizzo, P. Storniolo, and A. Urso, "The database-is-the-service pattern for microservice architectures," in *Proc. Int. Conf. Inf. Technol. Bio- Med. Informat.*, 2016. doi: 10.1007/978-3-319-43949-5_18.

Mohamed El Kholy is an Assistant Professor with the Computer Department, Faculty of Engineering, Pharos University in Alexandria, Alexandria, Egypt. His research interests include software engineering, web services, cloud computing, and software flexibility. His work has been published in highly ranked journals, and he is a Reviewer for a number of journals and conferences. He received the Ph.D. degree in information technology from the Institute of Graduated Study and Research, Alexandria University, Alexandria, Egypt. Contact him at eng_mikholy@alexu.edu.eg.

Ahmed El Fatatry is a Professor of software engineering and Head of the Department of Information Technology, Alexandria University, Alexandria, Egypt. His research interests include software processes, information retrieval, and software flexibility. His work has been published in highly ranked journals, and he is a Reviewer for a number of journals and conferences. He received the Ph.D. degree in information technology from University of Manchester Institute of Science and Technology, Manchester, U.K. Contact him at elfatatry@alexu.edu.eg.