Fast Replica of Polyglot Persistence in Microservice Architectures for Fog Computing

Michele Cantarutti, Pierluigi Plebani, and Mattia Salnitri(✉)

Politecnico di Milano, Milan, Italy
michele1.cantarutti@mail.polimi.it,
{pierluigi.plebani,mattia.salnitri}@polimi.it

Abstract. Aiming to break the software monoliths that traditional approaches usually produce as artifacts, solutions that are based on microservices consist of heterogeneous and independent software platforms to manage applications and data. In this scenario, the term polyglot persistence has been introduced to characterize software solutions where the involved microservices rely on different data storage technologies. Especially in Fog Computing where data are expected to efficiently flow among nodes – usually from the edge to the cloud – the polyglot persistence could have a negative impact since a combination of data replication and transformation is required. The goal of this paper is to study the challenges in data management in Fog Computing when microservices are adopted, and to present a solution which combines the advantages of the physical copy approach performed by network file systems to provide a fast data movement and the ability of the logical copy approach to transform the data. The resulting mix is demonstrated to reduce the time of creating the replica up to 70%.

Keywords: Efficient data management · Data movement

1 Introduction

The microservice architectural style is more and more adopted in software solutions due to its ability, among the others, to deal with scalability and ease of maintenance. As discussed in [13], seven main principles constitute the fundamentals of microservice architectures: fine-grained interface, business-driven development, cloud-native design, polyglot programming and persistence, lightweight containers, decentralized continuous delivery, and DevOps lean.

Polyglot programming and the persistence principle aim to enable the production of a software solution as a composition of several independent modules, developed by independent teams, and based on different technologies. This way, developers can break the classical monolithic solutions and use the most suitable technology for a given specific task, without the need to agree on a specific platform. The polyglot principle is in line with the need to get rid of the "one-size fits all" approach [11]. Thus, a microservice-based solution could involve a set of different DBMSs which could even rely on different models: e.g., relational DB, noSQL. This is also important when data to be managed by software solutions come from legacy systems but we want to exploit as much as possible new programming and storage paradigms to properly manage those data. As a consequence, mechanisms are required to keep the alignment of different data stores in the different nodes in which microservices are running.

The literature already proposes some solutions for the alignment of data stored in heterogeneous environments which are ready to be adopted also in a microservice architecture. Conversely, such solutions become no longer useful when considering the deployment of microservices along the continuum between the cloud and the edge, i.e., Fog Computing, which introduces additional requirements in terms of velocity, polyglot persistence, data transformation, and partial replication. In fact, Fog Computing [6], a paradigm for managing distributed systems where nodes, called fog nodes, live in the continuum between the cloud and the edge, implies: (i) a dynamic environment where fog nodes could easily join and leave the system, and (ii) a continuous data movement among different nodes which – due to the polyglot persistence principle – could be based on different storage technologies. In this context, providing a fast access to the data needed by a microservice is fundamental and it can be obtained by locating the required data closer to where the computation is running.

The goal of this paper is twofold. First, to investigate how the adoption of microservices affects software solutions in Fog Computing with respect to data management. As discussed in the paper, an efficient and flexible replica mechanism is fundamental and the current approaches – such as physical and logical copy [10] – either are not able to satisfy all the requirements of Fog Computing or the time required to create the replica is not acceptable. Second, this paper proposes a solution to efficiently ensure the creation of replicas on the nodes which are able to cope with the dynamism of the system and the heterogeneity of the technologies involved. As demonstrated by the performed tests, the proposed solution is comparable to traditional approaches for small databases but outperforms them up to 70% when the size of the database becomes significant.

The rest of the paper is organized as follows. Section 2 motivates the requirements for replica mechanisms in Fog Computing. Section 3 introduces the proposed solution, of which the evaluation results are presented in Sect. 4. Finally, Sect. 5 discusses related work and Sect. 6 concludes the paper.

2 Background and Motivation

Fog Computing has recently emerged as a paradigm for improving the performance of applications where data are generated on the edge but, due to the limited capacity in terms of computation and memory, they are processed on the cloud [2]. As the network can introduce a significant latency, the processing performed on the cloud may experience an unacceptable delay. For this reason, Fog Computing aims to create a synergy among resources on the edge, resources on the cloud, and even resources that connect the edge and the cloud, where all these nodes are generically referred to as fog nodes.

In this context, mechanisms for enabling data and computation movements hold a primary role. When possible, computation should be moved closer to where data are generated. As not all computation is possible on edge nodes due to their limited capacity, the data resulting from the initial analysis – which are less than the generated ones anyway – should be sent or replicated to cloud nodes. Moreover, especially when considering dynamic contexts, a fog node could unpredictably join or leave the system. Hence, when a node is part of the system, it can be a source of data or a place in which the computation can be executed. Thus, it has to access the data to be processed which could be other than the data that the node itself is generating.

Data generated at the sensing layer must, therefore, be replicated for the computation layer, usually organized according to a microservice architecture. Several challenges must be addressed concerning the creation and the management of the database replicas in Fog Computing. **R1**: *fast creation*: when a fog node joins the system, a secondary database should be quickly made available to the newly deployed microservices. **R2:** *polyglot persistence*: fog nodes could be based on heterogeneous storage technologies. **R3:** *partial replication*: for privacy issues or to reduce the amount of data to be transferred, the secondary database could contain a projection or a selection of data stored in the primary database. **R4:** *data transformation*: before moving to the secondary database, data could be transformed for privacy, security, or optimization reasons.

It is worth noticing that the resources available on fog nodes may vary from few cores and few megabytes of RAM and storage for nodes closer to the edge, to powerful nodes when considering the cloud. For this reason, the solution must be lightweight to be deployed in all configurations.

3 Fast Replica for Fog Computing

The replication mechanism proposed to satisfy the requirements introduced above is based on dynamic replication. Thus, secondary databases are added dynamically after the deployment of the primary database [12]. Static replication, where all secondary copies are deployed at the same time of the primary copy, is indeed not an option in our context, as in Fog Computing all fog nodes are not known in advance, as they could change dynamically.

The initial load phase of a database is considered in this paper. Such a phase is required every time a microservice is deployed on a fog node. Depending on the type of analysis and the amount of data generated by the sensing layer, it might happen that the replica creation could require to move a significant amount of data. In the literature [10], (see Fig. 1) two options are commonly considered: i) *logical copy*, which refers to the mechanism of extracting data from the primary copy and importing them onto the secondary copy using queries; and ii) *physical copy*, which refers to directly transferring files containing the DBMS data, from machine to machine, at filesystem level.

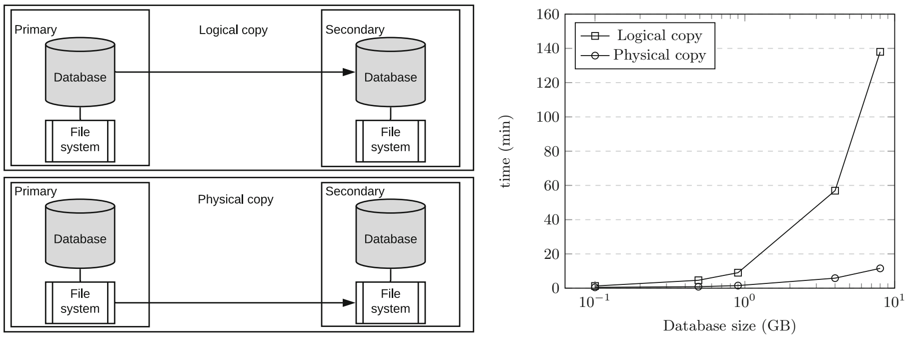

Fig. 1. Comparison of physical and remote copy.

The physical copy is a very fast process (actually the fastest according to our tests) but it only addresses R1. In fact, only full replication can be achieved as a physical copy is obtained by copying the entire data directory of the database onto a remote machine. For the same reason, data cannot be transformed (e.g., anonymized) during the replication process. Indeed, it is not possible to distinguish between columns, rows, or tables at the filesystem level. Finally, a physical copy cannot be used in a heterogeneous setup with different DBMSs, since the data directory, copied onto a remote machine, will be readable only by a DBMS that uses the same technology of the primary one. An adopted workaround to allow a polyglot environment with physical copy, consists of a primary node where data are stored in all the different database technologies that might be needed. When a replica is required, the physical copy of the database with the needed technology is performed. However, this approach is extremely space consuming, so it is not an acceptable solution, especially when considering fog nodes.

When it comes to logical copy, it allows filtering and transforming the data since, once rows are extracted from the primary copy, they can be filtered or transformed before they are sent to the secondary copy (R3, R4), so also different database technologies can be involved (R2). The main drawback of the logical copy concerns the time required to complete the copy on the secondary node. Indeed, as shown in Fig. 1, the performed tests show that the logical copy always requires more time than the physical one and, with an increasing size of the database to be replicated, the replica time has an exponential trend which makes this approach not suitable.

We propose a *hybrid approach* (see Fig. 2) to perform the initial load, that exploits the flexibility of the logical copy while maintaining the higher speed of the physical copy. The hybrid approach consists of four phases:

1. *Temporary node creation.* A DBMS (of the same technology as the technology of the secondary database) is deployed on a temporary node near the primary node (or, if possible, on the same node).
2. *Local logical copy.* The (partition of the) database to be replicated is copied into the new DBMS by using a logical copy. This allows to filter and to

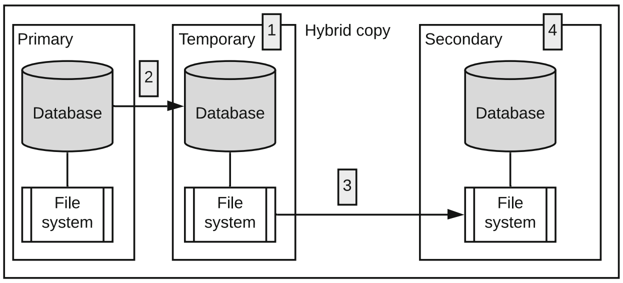

Fig. 2. Proposed hybrid approach

transform the data, and to translate the queries for a DBMS that is different from the primary DBMS. This operation takes less time than it would take to perform remotely onto the secondary node since it is performed on a node that is near (or local to) the primary node. The logical copy is performed by reading the primary copy and it does not need to lock the primary copy.

3. *Remote physical copy.* The newly created database is moved to the secondary machine by using a physical copy which has been demonstrated to be fast.
4. *Finalization.* The secondary DBMS is started on the secondary node, where it can access the newly copied database. The temporary node can be destroyed.

The overhead of this approach is given by the time necessary to deploy the temporary machine and create extra resources (e.g., a temporary DBMS). However, some technologies (e.g., Docker and Kubernetes) allow to deploy these necessary resources in a few tens of seconds, which is a negligible amount of time if compared to the overall benefit provided by the proposed approach. Similarly, the overhead given by the transmission of data between the primary database and the temporary one is negligible since the temporary machine should be created near the primary one (or be connected with a fast network connection).

It is worth noting that the usage of a temporary node is a valid approach in Fog Computing. For nodes located in a cloud environment, the overhead of the creation of a temporary node is negligible, due to the virtually unlimited resources available. Edge nodes, are typically IoT devices which produce data, that are stored in fog nodes. In this case, the temporary node will be created in fog nodes where resources, even if limited, are usually higher than IoT devices and the overhead will have a limited, and temporary, impact on the system.

Better performance can be achieved by starting the third phase (i.e., the remote physical copy) during the second phase (i.e., the local logical copy), without waiting for the second phase to finish. Using this variation, data is copied onto the secondary node while it is being written on the temporary node.

This variation (named *overlapped hybrid approach*) introduces three cases:

No Overlap. If the third phase (i.e., moving the temporary replica to the secondary node) is performed after the second one has finished, then the overlap will be minimum, that is, null. In this case, the total necessary time to create a copy on the secondary node is equal to the time necessary to create a local copy on the temporary node plus the time necessary to perform its entire physical copy onto the secondary node.

Perfect Overlap. If the physical copy is timed perfectly with the logical copy, the overlap will be maximum, and the two phases will be entirely overlapped. In this case, the total necessary time to create a copy on the secondary node is equal to the time necessary to perform a logical copy on the temporary node. This is an ideal scenario that does not happen in reality for two main reasons. First of all, the logical copy creates new files on the temporary node, while the physical copy copies them on the secondary node, and these two processes may follow different orders. Secondly, the exact times needed for logical and physical copies are unpredictable in a Fog Computing environment. Therefore, it is not possible to time the beginning of the physical copy perfectly so that its end coincides with the end of the logical copy.

Partial Overlap. The best obtainable degree of overlap is a partial overlap. The best strategy in order to maximize the overlapping of the two phases is to run the process of the physical copy twice. Keeping in mind that the physical copy is faster than the logical one, the first execution of the physical copy should be timed so as to finish approximately when the logical copy finishes. As soon as the logical copy ends, the second execution of the physical copy should start. This maximizes the amount of raw data copied by the first physical copy leaving to the second run of the physical copy only a small portion of the data. Transactions performed during the first physical copy may lead to integrity problems in the secondary node, because files are physically copied from the temporary node to the secondary one, while writes are occurring on the temporary node. This is not a problem, as the data is initially not accessed on the secondary node. Therefore, just before the second physical copy, the alignment between the primary and temporary node is suspended, and then the second physical copy to align and restore the integrity of the data on the secondary node is performed. Immediately after that, the alignment between the primary copy and the secondary copy starts. All the transactions performed after the beginning of the second physical copy will be propagated to the secondary copy.

3.1 Implementation Details

The proposed hybrid approach (both with and without overlap) has been implemented adopting the most common tools used to deploy and run microservices (i.e., Docker and Kubernetes) as well as existing software (i.e., SymmetricDS) that is able to provide a logical copy.

More in detail, Docker[1] allows to create isolated virtual environments known as *containers*, in which applications can be run. Containers are very lightweight: they use less space and they also take less time to start up compared to other virtualization tools. As a result, Docker allows to: (i) deal with fog heterogeneity,

[1] https://www.docker.com.

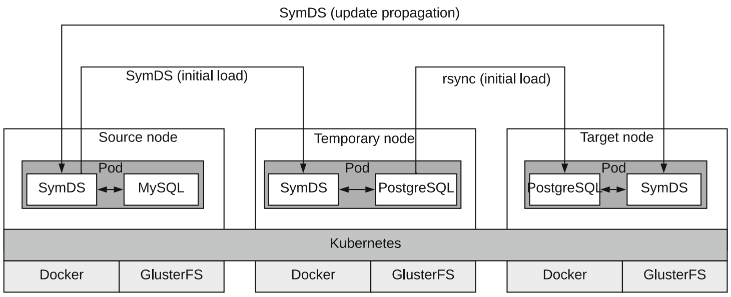

Fig. 3. Architecture of the approach

as applications are containerized and they do not need to rely on the specific hardware of the host machine; (ii) deal with fog dynamicity, as applications can be started in a fast and practical way.

To coordinate multiple nodes, orchestrator tools, such as Kubernetes[2], are involved. More specifically, Kubernetes is an open-source container-orchestration system for automating deployment, scaling and management of containerized applications in distributed systems that, amongst others, supports Docker containers. It provides a container-centric management environment, that orchestrates computing, networking, and storage infrastructure.

In a Fog environment, a *Kubernetes Node* is a worker machine, and it may be a virtual machine or a physical machine that corresponds to a node, a.k.a., Fog node. A set of Kubernetes Nodes makes up a *Kubernetes cluster*. A Kubernetes cluster corresponds to a set of fog nodes. Each microservice can be containerized and, therefore, it belongs to a single Docker container. A *Kubernetes Pod* is a group of containers with shared network and storage, that are always co-scheduled and co-located.

Finally, SymmetricDS[3] is an open source software package for database replication. It performs a type of replication known as transaction replication [5] as opposed to statement replication [5] . This means that the secondary copies do not receive SQL statements to apply, but rather only the changes produced by SQL statements, known as *writesets*. As it is built on top of JDBC, SymmetricDS supports a wide range of databases and it can automatically translate between different SQL dialects. Moreover, SymmetricDS supports filtered replication (to allow replication of specific tables, columns or rows) and it supports data transformation (which allows to anonymize or pseudonymize data before it is replicated).

Figure 3 shows the architecture for the implementation proposed in this paper. The lower part consists of Docker service, and GlusterFS service[4], a network filesystem we used for the creation of persistent volume where DBMS data is stored. Both services are installed in every fog node, that provide the

[2] https://kubernetes.io.
[3] https://www.symmetricds.org.
[4] https://www.gluster.org/.

Table 1. Replica time (in min) for the different approaches

DB Size	Physical	Hybrid			Overlap			Logical
		Logical local	Physical remote	Total	Logical local	Physical remote	Total	
100 MB	0.5	0.45	0.5	1.95	0.48	0.43	1.91	1.27
490 MB	0.87	1.78	0.87	3.65	1.77	0.52	3.29	4.62
900 MB	1.52	3.55	1.52	6.07	3.42	0.6	5.02	9.03
4 GB	5.85	16.32	5.85	22.9	16.65	1.9	19.55	56.93
8 GB	11.57	34.82	11.57	47.39	34.47	2.38	37.85	137.98

primitives for the management of containers. On top of it, Kubernetes provides the infrastructure that groups Fog nodes into a cluster. Kubernetes manages the resources provided by Docker and GlusterFS.

Kubernetes works as a central authority. This makes the scheduling of the resources very efficient since containers, being lightweight, are fast to start (generally less than 5 s).

4 Evaluation

To evaluate the proposed approach we compare the time needed to a create a new replica with traditional approaches, i.e., physical and logical copy, with the time required by the proposed approaches, i.e., the hybrid and overlapped hybrid copy. To obtain reliable results, primary databases of different sizes are considered to check how the results change as the size of the primary database grew. Moreover, to mitigate the influence of the network, the tests were repeated 5 times in each configuration, on different days and different times of the day.

Evaluation Setup and Execution. Three nodes were used to simulate the Fog nodes: two in the same physical location (Zurich - Switzerland) and one remote (Miami - USA). The nodes in Zurich were local to each other for the reasons explained in Sect. 3. We chose the location of the third node purposely at a great distance from the other two, to simulate a geographically distributed deployment of the Fog nodes, where the connection could be affected by great variations of performance. The three nodes are hosted in cloud resources and share similar characteristics: the two nodes in Zurich have 1 single-core CPU and 4 GB of RAM, while the node in Miami has 1 single-core CPU and 2 GB of RAM.

In order to produce significant results, OLTP-Bench[5] was used to populate the primary database with sample data. We used this benchmark defined by the Transaction Processing Performance Council (TPC) [4] as it emulates transactions of real databases mimicking new observations of the sensing layer.

[5] https://github.com/oltpbenchmark/oltpbench.

Results. Table 1 shows the results of the conducted tests. Here, the logical and physical copy represent, respectively, the upper and lower bound.

Hybrid and overlapped approaches are decomposed in two execution times:

- Logical local: the execution time of the logical copy in the temporary node.
- Physical remote: the execution time of the physical copy from the temporary machine to the secondary node.

The difference between the observed total time and the time to perform both the logical local copy and the remote physical one is related to the rescheduling time needed to bootstrap the third node.

The results clearly highlight that the proposed approaches have a lower execution time that the classical one (logical copy). Such a difference is up to 73% in case of a 8 GB database with the overlap method. The tests show that the time of the traditional approach grows over twice as faster than that of our proposed approaches. However, the hybrid approaches are advantageous only beyond a certain size of the database. Indeed, when the database is small, the traditional approach of the remote logical copy is faster.

5 Related Work

Database replication has been extensively studied in the literature and, as discussed in this section, there are solutions which inspired the proposed approach but that also have limitations which hamper their adoption in Fog Computing.

Among these approaches most of them propose a middleware. Since [3] offers a read-one/write-all approach, its proposed solution requires a lock of the primary copy, thus reducing the efficiency of the replica creation. The middleware proposed in [1] is based on a scheduler accepting transactions from users which will be sent to replicas with a distributed conflict aware approach. Such an approach parses SQL statements while users must declare at each transaction which tables are being modified. This approach permits to fine tune the amount of transactions to be sent on each DBMS, however, it does not support polyglot persistence. MIDDLE-R [9] is a middleware mainly focused on granting consistency among the copies, but it is unable to deal with dynamic environments, as it only considers systems with a fixed number of nodes. Moreover, it is unable to recover nodes after they crash, and, when nodes are falsely suspected to have crashed, they are forced to commit suicide regardless. In [8], authors propose a middleware to distribute requests based on the locality of the data and, therefore, increasing the likelihood of using the cache of the DBMS. However, this approach is based on static replication and static content, and so it does not support updates on the replica, but it rather focuses on the distribution of content. Finally, [7] compares some peer-to-peer solutions, where data storage and processing are distributed across completely autonomous peers. These solutions support a write-anywhere approach, and, consequently, they require reconciliation algorithms to fix the divergences that arise among the replicas. In dynamic environments where data is continuously updated, this can drain a lot of computational power from the nodes. Also, most of these solutions are based on a weak type of replication, known as passive replication, where a piece of data is specifically replicated only after the user tries to access it.

6 Conclusion

Due to the provided flexibility and scalability, the microservice architectural style represents a good approach to developing applications according to the Fog Computing paradigm. Nevertheless, the dynamicity of fog nodes requires a data management that is able to quickly react to the re-deployments that may occur to satisfy the quality of service that the applications have to ensure. In particular, this paper has identified the need for mechanisms able to quickly create replicas. As the typical physical copy does not provide the proper support for fog environments and the logical copy is too slow, this paper proposes a hybrid approach that is able to exploit both the advantages of the classical solutions. The performed tests demonstrated how the hybrid approach can save up to 70% of the time usually required to create replicas for an almost 10 GB database.

References

1. Amza, C., Cox, A.L., Zwaenepoel, W.: A comparative evaluation of transparent scaling techniques for dynamic content servers. In: Proceedings of ICDE 2005, pp. 230–241 (2005)
2. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the Internet of Things. In: Proceedings of MCC 2012, pp. 13–16. ACM (2012)
3. Cecchet, E.: C-JDBC: a middleware framework for database clustering. IEEE Data Eng. Bull. **27**(2), 19–26 (2004)
4. Difallah, D.E., Pavlo, A., Curino, C., Cudre-Mauroux, P.: OLTP-bench: an extensible testbed for benchmarking relational databases. VLDB **7**, 277–288 (2014)
5. Cecchet, E., Candea, G., Ailamaki, A.: Middleware-based database replication: the gaps between theory and practice. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, pp. 739–752 (2008)
6. IEEE: IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing. IEEE STD 1934–2018, pp. 1–176, August 2018
7. Martins, V., Pacitti, E., Valduriez, P.: Survey of data replication in P2P systems. Technical Report RR-6083, INRIA (2006)
8. Pai, V.S., et al.: Locality-aware request distribution in cluster-based network servers. ACM SIGPLAN Not. **33**(11), 205–216 (1998)
9. Patiño-Martinez, M., Jiménez-Peris, R., Kemme, B., Alonso, G.: MIDDLE-R: Consistent database replication at the middleware level. ACM Trans. Comput. Syst. (TOCS) **23**(4), 375–423 (2005)
10. Plattner, C., Alonso, G., Özsu, M.T.: Extending DBMSs with satellite databases. The VLDB J. **17**(4), 657–682 (2008). https://doi.org/10.1007/s00778-006-0026-x
11. Stonebraker, M., Cetintemel, U.: "One size fits all": an idea whose time has come and gone. In: Proceedings of ICDE 2005, pp. 2–11 (2005)

12. Tos, U., Mokadem, R., Hameurlain, A., Ayav, T., Bora, S.: Dynamic replication strategies in data grid systems: a survey. J. Supercomput. **71**(11), 4116–4140 (2015). https://doi.org/10.1007/s11227-015-1508-7
13. Zimmermann, O.: Microservices tenets. Comput. Sci. Res. Dev. **32**(11), 301–310 (2016). https://doi.org/10.1007/s00450-016-0337-0