


ORBITER: a Lightweight Framework for Automatic Deployment of Big Data Applications on Serverless Architectures

João Loureiro  [Universidade Federal Fluminense | joao_loureiro@id.uff.br]

José Maria Monteiro  [Universidade Federal do Ceará | monteiro@dc.ufc.br]

Marcos Bedo  [Universidade Federal Fluminense | marcosbedo@ic.uff.br]

Daniel de Oliveira   [Universidade Federal Fluminense | danielcmo@ic.uff.br]

 *Institute of Computing, Universidade Federal Fluminense, Av. Gal. Milton Tavares de Souza, s/n, São Domingos, Niterói, RJ, 24210-590, Brazil.*

Received: 11 March 2023 • **Published:** 22 December 2023

Abstract The Serverless Computing paradigm has become a reality, with various public cloud environments offering serverless infrastructure and functionalities for general use. One of the main advantages of deploying applications in serverless architectures is that developers can focus on implementing the application and business logic while avoiding the complexities of deployment. However, despite these advantages, there are still challenges to consider, such as the delay in building the architecture of the distributed application. In this article, we introduce a lightweight framework designed for deploying big data applications in a serverless architecture, specifically following the Function-as-a-Service (FaaS) model. The proposed framework uses open-source tools and was thoroughly evaluated through the implementation of two practical applications for Crime Hot Spot Analysis and Rainfall Analysis in the Microsoft Azure cloud. The results of our evaluation demonstrated the potential of the proposed framework in streamlining the deployment process with acceptable overhead.

Keywords: Serverless, FaaS, Big Data Applications

© Published under the Creative Commons Attribution 4.0 International Public License (CC BY 4.0)

1 Introduction

The world has observed a massive increase in data production and availability in the last decade [Sznaier *et al.*, 2014; Nandury and Begum, 2016]. Data are produced by several applications ranging from social networks to IoT devices, such as rainfall stations [Nascimento *et al.*, 2022] and wearables [Marchioro *et al.*, 2022] and are gathered in many formats, *i.e.*, structured, semi-structured, or unstructured [Liu, 2021]. While this data volume opens room for many commercial and academic development opportunities, it also includes new research challenges. One of those challenges is how to improve the accessing, querying, and processing of those evergrowing *big data* repositories.

Different solutions have been proposed to foster the development of applications that process and extract useful information from big data, which include big data frameworks with in-memory processing (*e.g.*, Apache Spark [Zaharia *et al.*, 2016]), NoSQL Database Management Systems, and middlewares with in-memory capabilities (*e.g.*, DuckDB [Raasveldt and Mühleisen, 2019], Redis/Redis++ [Zhang *et al.*, 2018], Tarantool¹), and query engines (*e.g.*, Apache Calcite [Begoli *et al.*, 2018]). This myriad of solutions has created a complex software ecosystem that the application developer must somehow traverse for each new project. Although cloud providers [González *et al.*, 2009] have simplified some of the setup steps by supplying the

virtual machine with pre-installed big data frameworks, the configuration and maintenance of the entire development stack still require expressive human effort due to the number of optimization parameters and continuous software evolution [de Oliveira *et al.*, 2021]. The serverless paradigm has emerged to soften the complexity of developing and deploying distributed, particularly big data, applications [Hellerstein *et al.*, 2019]. Serverless is an alternative to the traditional cloud application deployment process, where developers are no longer responsible for managing the servers that execute the programs.

Therefore, the paradigm contrasts with the classical cloud computing model² by allowing users to focus only on the development while delegating issues, *e.g.*, scalability and fault tolerance [Hassan *et al.*, 2021], to the cloud provider. Most cloud providers offer serverless mechanisms based on “serverless containers”, which refers to technologies that allow cloud users to execute containerized applications without the need to manage the underlying servers or computing infrastructure on which the containers are executed. Deploying containers at scale in a production environment may be far from trivial. Solutions like Kubernetes [Thurgood and Lennon, 2019] play an important role in this regard. However, two open issues are still encountered in practice: the configuration of the cluster of containers that includes the ap-

¹<https://github.com/tarantool/tarantool>

²The Serverless Computing paradigm also follows the idea of *pay-as-you-go* [de Oliveira *et al.*, 2010], *i.e.*, the developer only pays for the quantum of time the application executes.

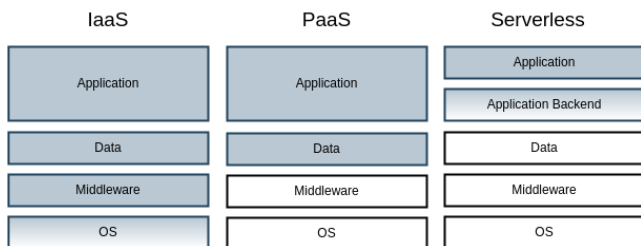


Figure 1. Cloud Infrastructure with IaaS, PaaS, and Serverless (Adapted from <https://1024.page/2019/web/about-Serverless>)

plication, *e.g.*, using Kubernetes, and the portability between different Serverless environments, *e.g.*, Lambda from AWS and Azure Functions from MS Azure [Sousa, 2020].

This article presents ORBITER (framework for BIG data dEployment on seRverless mechanisms), which tackles those open issues by assisting the developer in the deployment of the cluster of containers with a dozen of the required software to execute an application. ORBITER is built as a lightweight framework that provides continuous data processing and supports the automatic deploying of big data applications in a serverless environment, which can be hosted by distinct cloud providers. We show the ORBITER viability and potential through the execution of applications designed to analyze crime hot spots in large urban centers [Sá *et al.*, 2022] and to analyze rainfall data in the city of Niterói [Nascimento *et al.*, 2022]. Those results, a revised and improved background, and a broader comparison with related work provide new empirical shreds of evidence regarding the ORBITER capabilities in practice, extending the conference paper that introduced our framework [Loureiro and de Oliveira, 2022].

The remainder of the article is organized as follows. Section 2 discusses the serverless paradigm, while Section 3 presents the ORBITER architecture and its implementation. Section 4 shows the ORBITER evaluation with a real-world application. Finally, Section 5 addresses related work and Section 6 concludes the study.

2 Serverless Computing in a Nutshell

The serverless computing paradigm can be implemented following two different architectures [Mampage *et al.*, 2021]: (i) Backend as a service (BaaS) and (ii) Function as a Service (FaaS). In BaaS, besides the application itself, developers can access third-party services through specific APIs, *e.g.*, authentication, and database services. On the other hand, FaaS relies on local (in the cluster) services so that developers are responsible for creating the application logic on the server side, whereas the application executes over containers (*e.g.*, Docker [Miell and Sayers, 2019] and Singularity [Godlove, 2019]), which are managed by the cloud provider. Notice that containers are stateless, which implies applications in need of data persistence may face some difficulty in this architecture (the solution being using DBMSs as functions). Although FaaS containers are *ephemeral* (in the sense they are designed to execute in a small time window), the majority of cloud providers already offer FaaS architecture, *e.g.*, *Azure Functions* (MS Azure), and *Lambda* (AWS). The ORBITER framework follows the FaaS architecture.

However, unlike in IaaS (*Infrastructure as a Service*) environments, developers are not required to create a virtual machine in FaaS or even configure/maintain it (*e.g.*, applying security *patches*). Cloud servers become an abstraction during the application development process, easing the operation. Figure 1 presents the components of the software stack that have to be managed by the developer in IaaS, PaaS (*Platform as a Service*), and Serverless architectures (shadow-gray background). On the other hand, in serverless architectures, the developer can skip this configuration stage and start with the packaging of the application code (*i.e.*, function) developed in containers and deploy them in the serverless environment. Consequently, the financial cost of development operations can be reduced with FaaS adoption compared to IaaS and PaaS in some scenarios, *e.g.*, a few requests in time.

Another major gain of FaaS is deployment time. Both virtual machines and containers take longer to run the initial setup than functions in the Serverless architectures, as you need to configure the infrastructure *a priori*. On the other hand, functions typically take a few milliseconds to be deployed because they require no invocation of such an initial setup. Thus far, the great disadvantage of serverless architectures is the complexity of deployment and portability since it is still challenging to configure the serverless services in multiple cloud providers. That is due to the fact that serverless solutions are still quite specific in terms of applied technology, hardening both service and provider portability.

3 The ORBITER Approach

ORBITER is a lightweight framework that aims to ease the deployment of big data applications following the serverless paradigm. It is worth noting that ORBITER is more than just a serverless service; it is a comprehensive framework that configures the environment for deploying container-based services. ORBITER is a “lightweight” solution because it presents minimal impacts to the application being deployed as a service, which means requiring few code changes in the application compared to other frameworks. The ORBITER architecture is presented in Figure 2 and includes three major layers: (i) Initialization, (ii) Ingestion/Storage, and (iii) Data Processing.

The *Initialization Layer* is responsible for creating the container cluster where the application will execute. Notice while the deployed services may be called multiple times, the configuration of the container cluster is performed only once. The main component of this layer is the *Infrastructure Manager*. In the current implementation, ORBITER uses Terraform³ [de Carvalho and de Araújo, 2020] to implement this function. Terraform is a cloud-based framework that enables infrastructure automation for provisioning and managing clouds, data centers, and services. It enables deploying (Figure 2 – Step ②) the Kubernetes cluster (container orchestration system) on multi-cloud environments, *i.e.*, Amazon Web Service (AWS), Google Cloud Platform (GCP) and Microsoft Azure, based on declarative configuration files defined by the developer (Figure 2 – Step ①). ORBITER gains the ability to deploy services in a wide range of existing cloud

³<https://www.terraform.io/>

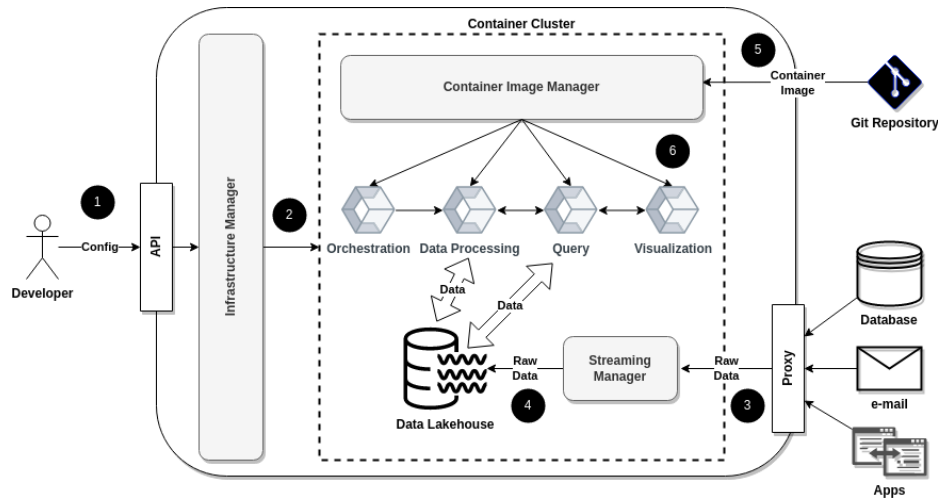


Figure 2. The ORBITER architecture.

environments by adopting Terraform. The *Infrastructure Manager* also provides the setup for the Kubernetes version that will be used and the configuration of nodes that are part of the cluster. The *Infrastructure Manager* also feeds from the ArgoCD⁴, a declarative GitOps delivery tool for Kubernetes. In the context of ORBITER, ArgoCD is used to arrange the components in both *Ingestion/Storage* and *Data Processing* layers. In ORBITER, Kubernetes is used combined with the Helm package manager⁵, which enables the setup of the execution order of components in the Kubernetes cluster for the packed application. Thus, each part of the application is a Helm *Chart*, allowing the framework to manage resources such as CPU, memory, and disk.

The *Ingestion/Storage Layer* is responsible for receiving data from external sources (Figure 2 – Step 3) and storing it in the cluster of containers previously created (Figure 2 – Step 4). The ingestion layer must present scalable and fault-tolerant features. It also must load data efficiently to enable real-time and *batch* processing. Such features are carried out by the *Streaming Manager*. In the current ORBITER implementation, the *Streaming Manager* is built on top of Apache Kafka⁶, which is a distributed real-time event platform in which events run on *producer* threads to be captured by one or more *consumers* through API requests. We also use Kafka Connect, which creates custom producers and consumers for a user-provided tool and environment, e.g., PostgreSQL, MySQL, or Amazon S3. Finally, ORBITER relies on a *Data Lakehouse* [Behm et al., 2022] to store the resulting data. The idea behind a *Data Lakehouse* is to provide a data repository for large volumes and different data formats while ensuring ACID properties for transactions. ORBITER uses the *Delta Lake*⁷ [Armbrust et al., 2020] for deploying the *Data Lakehouse*, which integrates data processing and data exploration tools. Behind the data lake, ORBITER relies on Minio⁸, which is a multi-cloud object storage that communicates using the Amazon S3 protocol.

The last ORBITER component is the *Processing Layer*. It is responsible for sanitizing and manipulating the data within the *Data Lakehouse*. In this layer, containers are configured by the *Container Image Manager* component, which provides multiple *functions* for the application being developed/deployed (Figure 2 – Step 6), including (i) task orchestration (e.g., Apache Airflow), (ii) distributed processing (e.g., Apache Spark [Zaharia, 2013]), (iii) data queries (e.g., Apache Hive [Camacho-Rodríguez et al., 2019]), and (iv) visualization (e.g., Apache Superset). The *Container Image Manager* follows the concept of GitOps [Beetz and Harrer, 2022] to access the image catalog (Figure 2 – Step 5). The idea behind GitOps is to have a Git entry containing the declarative descriptions of the infrastructure and one automated process to make the production environment match the state described in the repository. In the current ORBITER implementation, the *Container Image Manager* is implemented on top of ArgoCD, which is a tool for GitOps on Kubernetes. Accordingly, the containers are instantiated to execute the application from the *Container Image Manager*. The source code for ORBITER is being constantly updated in the open source repository <https://github.com/UFFeScience/orbiter>.

4 Experimental Evaluation

In this section, we present ORBITER evaluation. We examine our proposal with two real-world applications (analysis of crime hot spots and rainfall analysis) and then introduce the setup of the empirical evaluation. Finally, we present the performance results and discuss the experimental findings.

4.1 Analysis of Crime Hot Spots

The first real-world application we have chosen to showcase ORBITER is the analysis of *Crime Hot Spots*. A Crime Hot Spot can be described as a cluster of crime events distributed across a geographical area [Kikuchi et al., 2012]. This type of application provides data-driven observations that enable the identification of geographical regions with high crime

⁴<https://argoproj.github.io/cd/>
⁵<https://helm.sh/>
⁶<https://kafka.apache.org/>
⁷<https://github.com/delta-io>
⁸<https://min.io/>



Figure 3. Crime occurrences in “Sapopemba” in November/2019.

rates and promote *predictive policing* strategies. In the large urban areas of South America, such as São Paulo, Rio de Janeiro, or Buenos Aires city, the analysis and decision-making on *Crime Hot Spots* are far from trivial due to the cities’ massive area, number of inhabitants, or the frequent lack of human resources [Lourenço *et al.*, 2018].

For example, let us consider the “Sapopemba” region of São Paulo, Brazil. Sapopemba is a district located in the east part of the city, and it presented high crime rates during the year 2019, which ranges from theft to violent crimes⁹. Figure 3 presents an accumulated heat map for street crimes as of November/2019. The color ranges from dark red (very frequent) to white (non-occurrences). Data presented in Figure 3 summarize the counting of crime events during 2019. Furthermore, we emphasize that the scope of this case study does not extend to any conclusive inference regarding criminal hot spots/patterns to be used by law enforcement officers. The primary focus of the case study presented in this section (and the remaining experiments) is to demonstrate the viability of ORBITER and its potential application in a serverless architecture instead of providing either detailed crime analysis or actionable insights for law enforcement purposes.

Notice although the heat map provides an overall picture of the region, the identification of hot spots depends on aggregating crime data by space and time, *i.e.*, street segments, and period/hour of the year. Such a refined analysis would provide police officers with more accurate data so that they can define optimized police patrol routes as well as identify strategic locations for outposts. Accordingly, we evaluate two fictitious, highly-volume queries that would be executed within the Hot Spot analysis. They involve aggregating crime events by street segments (each approximately 100 meters-long) and by time (daily/per hour) and filtering by distance, *i.e.*, radius in kilometers.

Experimental data were downloaded from the Department of Public Safety of the São Paulo State¹⁰. We pre-process data to filter crimes that occurred during the year 2019. To reduce the number of crime events, we also filtered the crimes labeled by the Safety Department as femicide, car theft, car robbery, cellphone theft, cellphone robbery, theft, armed robbery, and homicide. We also discarded the crime events that were registered without georeferences as they are insufficient

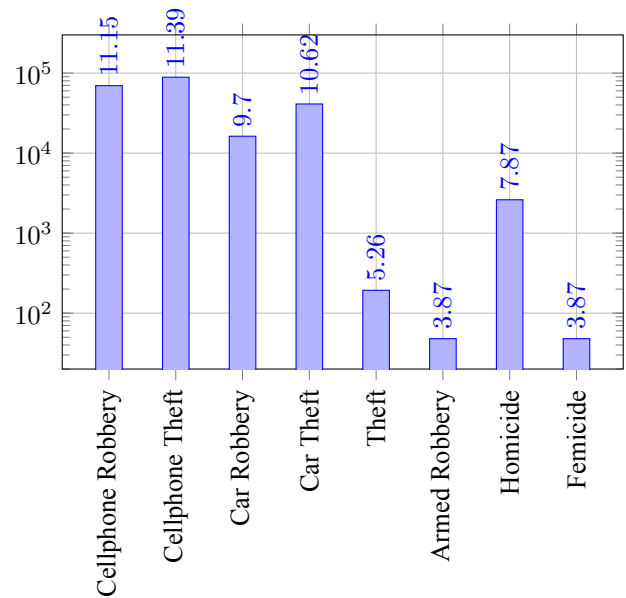


Figure 4. Number of crime events in the downloaded dataset.

to associate with street segments. The dataset after aggregating the crime events by segment can be downloaded at <https://osf.io/mxrgr/>. Figure 4 summarizes the total number of crime events in the city loaded in the experimental evaluation (accumulated occurrences in \log_{10} scale). The downloaded dataset has approximately 1GB size.

4.1.1 Environment Setup

For this first experiment, we have deployed ORBITER on the Microsoft Azure cloud¹¹. Microsoft Azure is one of the top players in the cloud market and provides different types of virtual machines to be deployed on demand. A Kubernetes cluster was configured with a Master instance with 2 vCPUs and 4 GB RAM and a Worker instance (common node) with 2 vCPUs and 16 GB RAM. Each instance was configured with a 30GB disk. We also set up a public IP and URLs to access the services.

4.1.2 ORBITER Setup

We deployed the *Crime Hot Spot* application using ORBITER with the downloaded crime dataset. The first execution step was to load external data into ORBITER through proxy. We use Kafka Connect with S3 connectors to ingest data into the Data Lakehouse in the Parquet file format with Snappy compression¹² (this compression library does not ensure maximum compression rate, but provides high compression speed). The Apache Airflow service executes a scheduled task that invokes Apache Spark after data is loaded into the Data Lakehouse. Then, Spark transforms the data into Delta Lake tables format. Finally, data become available for querying and browsing at runtime. The application uses Trino (a query mechanism for distributed analytical queries), whose results can be presented in a *dashboard* generated by *Apache Superset*.

The following customized container images are required for the application to be executed using ORBITER: (i) An im-

⁹<https://www1.folha.uol.com.br/fsp/especial/fj2501200438.htm> and <https://www.estadao.com.br/brasil/avenida-sapopemba-e-a-mais-perigosa-de-sp/>

¹⁰<https://www.ssp.sp.gov.br/>

¹¹<https://azure.microsoft.com/pt-br/>

¹²<http://google.github.io/snappy/>

age containing Kafka Connect and the JAR files connectors for the file system and S3; (ii) An image containing Apache Airflow and Kubernetes so that Airflow can communicate with the Kubernetes cluster and Apache Spark; (iii) An image containing the Apache Spark application (the application’s PySpark code) and the Delta-Spark library, which ensures the application can write data in the proper Delta Lake format; and (iv) An image containing the Apache Superset for data visualization and the *sqlalchemy-trino* library for Superset to retrieve data through Trino.

4.1.3 Discussion for the case study

We deployed the *Crime Hot Spot* application and executed two queries commonly-to-be submitted by this application. The first query (Q1) processes the downloaded crime dataset for November/2019 and aggregates the total number of crimes for each street segment in São Paulo city. The second query (Q2) aggregates the total number of crime events by segments within a radius r (in kilometers) of a given point in the city (as presented in Figure 5). Both queries were defined together with a police officer expert, following real-world setups for data-driven decision-making. Therefore, Q2 has to process fewer data in comparison to the first query (Q1), but the search presents a costlier spatial component. We measured the overall performance by considering both the elapsed time in ORBITER and the financial cost involved in the execution. The total elapsed time using ORBITER depends on a pipeline composed of three activities: (i) deployment of the Kubernetes cluster by the cloud provider, (ii) deployment of all services in the Kubernetes cluster (e.g., Apache Kafka and Apache Airflow), and (iii) the queries execution. Figure 6 presents the average elapsed time for ten executions (in seconds) for each activity of the pipeline. Cluster Deployment presented an average elapsed time of 620.53 seconds, while Service Deployment showed a mean elapsed time of 322.12 seconds.

These execution times align with our initial expectations, considering that the environment setup requires a sequence of actions within the cloud provider. For instance, when deploying in the Azure cloud, several crucial steps must be performed, such as defining the resource group, creating a node pool, configuring IP ranges, etc. These actions are vital for establishing the required infrastructure to support the deployment of services. Finally, the execution of query Q1 presented an average elapsed time of 205.22 seconds and the execution of query Q2 presented an average elapsed time of 45.74 considering the spatial-based query scenario.

In total, all activities of the aforementioned pipeline demanded less than 20 minutes to execute. Notice the deployment of the Kubernetes cluster (as well as that of services) is executed only once (excluding service updates), but queries Q1 and Q2 can be submitted several times after that (since the environment is already configured). Thus, when police experts need to perform new hot spot analyses, the total elapsed time will be (much) lower than 205.22 seconds since the cluster deployment and service deployment steps will not be executed. Regarding the financial cost, the entire execution of the ORBITER queries over our particular cloud setup has cost nearly 0.81 USD (deployment of images plus ten executions

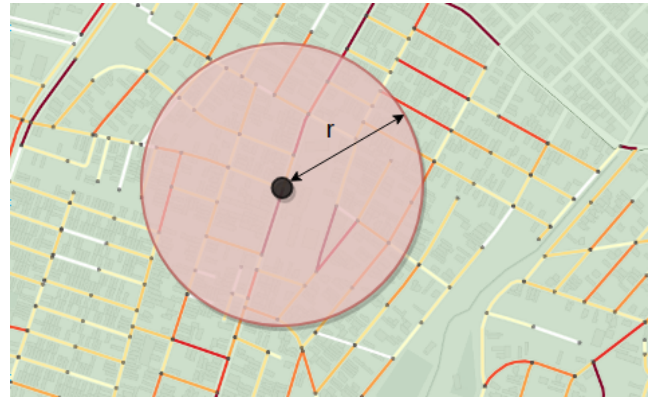


Figure 5. Range query with radius r in the “Sapopemba” region.

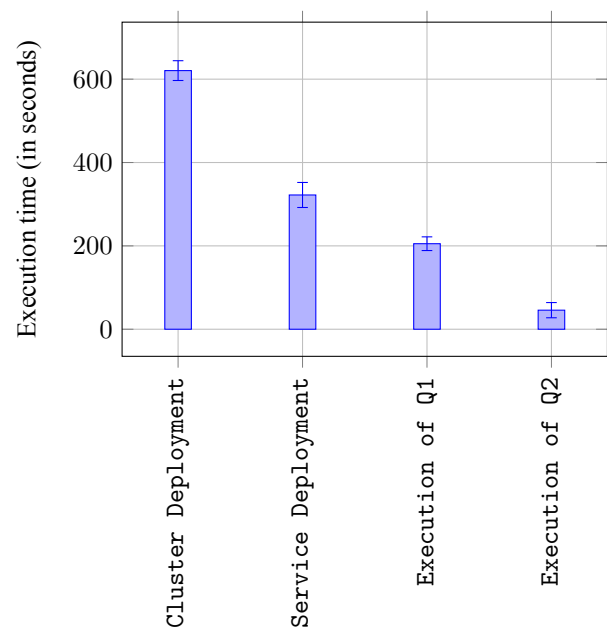


Figure 6. Execution time for deploying and executing the crime hot spot analysis application in ORBITER.

of each query), which was reported as financially viable by police experts.

4.2 Rainfall Analysis

The second real-world application we have chosen as a case study is the rainfall analysis of Niterói city, as previously proposed by Nascimento *et al.* [2022]. We defined the queries for this case study by profiling the TEMPO tool [Nascimento *et al.*, 2021], whose goal is to capture multi-source rainfall data with different granularities and integrate them into a single database to allow interactive visual analyses.

Over the past decade, there has been a growing recognition of the importance of data-driven climate studies, primarily because of the escalating number of extreme weather events [Mizutori and Guha-Sapir, 2020]. In cities, due to uncontrolled urbanization, there is a lack of sufficient green areas, leading to rapid runoff of rainwater into rivers that may not have the capacity to accommodate such high volumes, resulting in flooding [Thorndahl and Willems, 2008].

Consequently, the effective monitoring of rainfall plays a critical role in avoiding these floods. Numerous rainfall stations equipped with sensor-based data acquisition sys-

tems have been strategically installed across several cities in Brazil to facilitate comprehensive monitoring. Data from Niterói City, where our campus is located, was picked for the analysis presented in this section. The data obtained from Niterói city rainfall stations serve as a valuable resource but come from different sources.

Thus, all data employed in our evaluation are obtained from an existing Data Warehouse (DW) [Kimball and Ross, 2002], which, in its current version, adheres to the ROLAP (Relational OLAP) model and operates on a PostgreSQL Relational Database Management System. Notably, this DW contains a substantial dataset amounting to 50GB of data, which makes the entire DW a consolidated source for our evaluation. Figure 7 illustrates the schema of the DW, which follows the star schema and comprises one fact table and four dimension tables. The fact table, named *Fact_Rainfall*, encompasses essential attributes for analysis, including *rainfall_index* (the quantity of interest) and foreign keys to multiple dimensions: *Dim_Source* (data sources), *Dim_Time* (time dimension), *Dim_Locality* (locality of the stations), and *Dim_Station* (rainfall stations). Figure 8 summarizes the total number of tuples in each table of the DW (number of tuples is presented in \log_{10} scale).

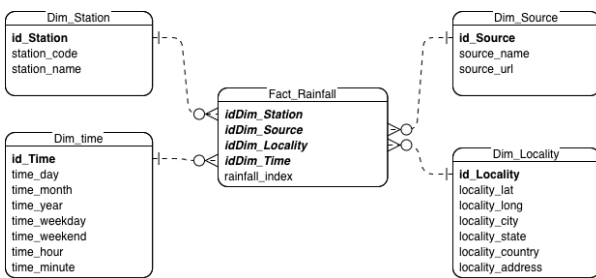


Figure 7. The DW schema used as source for the rainfall analysis.

By profiling the user interactions within the TEMPO tool, we have successfully identified a set of commonly executed queries/patterns, as presented in Table 1. We have focused on two of the most frequently submitted queries for this experimental evaluation. Query Q1 calculates the average rainfall for different time granularities (*i.e.*, 15 minutes, 30 minutes, 1 hour, 1.5 hours, 12 hours, and complete 24 hours) for each rainfall station. On the other hand, query Q2 returns the average rainfall with the same time granularity but for the whole of Niterói City.

We highlight there are numerous queries submitted by users during their analyses using the TEMPO tool, but we have intentionally selected these specific two queries to showcase due to their prevalence and significance. The decision to limit the analysis in these two queries is because of the scope of this article, since conducting a complete statistical summarization of the entire TEMPO tool and all the queries it encompasses is a task that does not depend on ORBITER.

4.2.1 Environment Setup

For this second experiment, we also have deployed ORBITER on the Microsoft Azure cloud. A Kubernetes cluster was con-

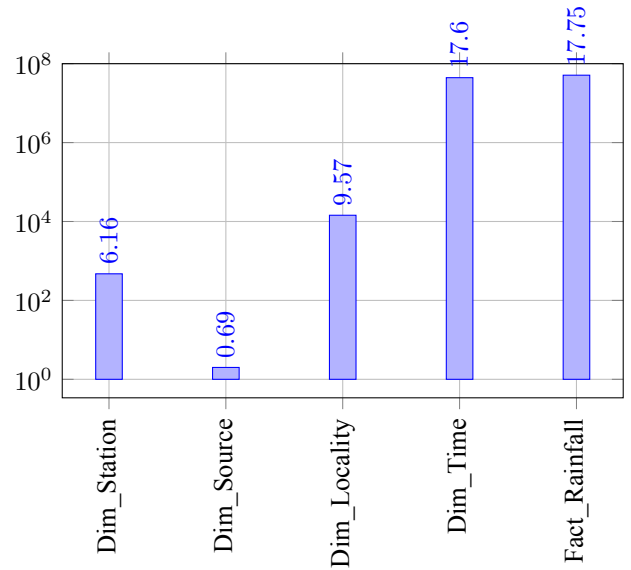


Figure 8. Number of tuples in each table of the DW.

figured with a Master instance with 2 vCPUs and 4 GB RAM and three Worker instances with 4 vCPUs and 16 GB RAM. Each instance was configured with a 30GB disk. We also set up a public IP and URLs to access the services.

4.2.2 ORBITER Setup

The configuration of the ORBITER framework used in this experiment is identical to the one presented in Subsection 4.1.2. The only difference lies in the nature of the data and the queries involved, which are specifically tailored for rainfall analysis. This targeted adaptation ensures that the framework efficiently processes and addresses the unique demands of rainfall-related data and queries, enabling effective analysis of the given dataset.

4.2.3 Discussion for the case study

We deployed and executed the *Rainfall Analysis* application for performing both queries Q1 and Q2. The first query is more time-consuming since it has to aggregate data for each station, while the second aggregates data in a coarse-grain, *i.e.*, city-level. Similarly to the previous experiment, the total execution time using ORBITER depends on a pipeline composed of three activities: (i) deployment of the Kubernetes cluster by the cloud provider, (ii) deployment of all services in the Kubernetes cluster, and (iii) the query execution.

Table 1 presents the average execution time for ten executions (in seconds) for each query. The required time to deploy the Kubernetes cluster and the services did not vary compared to the previous experiment. It was somewhat expected since the deployed services were almost the same. The three pipeline activities demanded less than twelve minutes to execute Query Q1 and less than ten minutes for Query Q2 in total. As in the other case study, while the deployment of the Kubernetes cluster (as well as that of services) is performed only once, queries can be submitted multiple times after the complete environment is up.

Table 1. Queries considered in the rainfall analysis experiment.

| Query | Description | Execution Time (seconds) |
|-------|---|--------------------------|
| Q1 | Returns the average rainfall for the last 15 minutes, 30 minutes, 60 minutes, 90 minutes, 120 minutes, 12 hours, 24 hours for all the rainfall stations | 697.01 |
| Q2 | Returns the average rainfall for the last 15 minutes, 30 minutes, 60 minutes, 90 minutes, 120 minutes, 12 hours, 24 hours per city | 569.28 |

5 Related Work

There are some literature approaches that have been proposed to help developers deploy their big data applications following serverless architectures. In this article, we follow their indications by adopting a simplified forward snowballing strategy [Wohlin, 2014] by considering some seed papers as input. The seed set comprises papers of an area. Then, the papers that cite the seed papers and that meet specific criteria are considered in a new snowballing round. Although this process may repeat for several rounds or until no new papers are included in the next snowballing, in this article we fixed the number of rounds as two since serverless computing has been popularized in the last few years, *i.e.*, papers are recent.

We relied on Google Scholar to get the seed set by searching “*Big Data Application Deployment in Serverless*”. Since we are analyzing papers that cite the chosen papers, we used Google Scholar because it already accesses many repositories such as IEEEExplore and ACM Digital Library. As inclusion criteria, we considered peer-reviewed papers, papers published after 2018, and papers published in English. Any paper that is not written in English or peer-reviewed was excluded from the snowballing process. We obtained 11 papers. These papers were published in 2 distinct journals, and 9 conferences or workshops. Table 2 presents the paper citation, the conference or journal where the paper was published, and the year of publication. Following we discuss each work.

Kuhlenkamp *et al.* [2019] propose an evaluation method called SIEM to measure quality in different big data application scenarios using serverless architectures. Although the authors do not propose a framework for deploying such type of application, they show used SIEM to evaluate a series of FaaS providers and applications for big data processing.

Castellanos *et al.* [2019] propose an approach that follows the DevOps and Domain Specific Model concepts to design, deploy, and monitor the performance of serverless big data applications. The authors explicitly describe the deployment strategies that are used to configure the environment and the application. The authors evaluated the proposed approach with a real-world application for mid-air collision detection.

Wen *et al.* [2021] presents a study that aims to understand the challenges of developing serverless-based applications. Their study is based on questions obtained from Stack Overflow. The subjects of this study were application developers, researchers, and cloud providers. One of the challenges identified by the authors is how to deploy applications in serverless architectures, as discussed in this article. Bhat *et al.* [2022] presents a study to evaluate serverless computing for performing large-scale data processing using cloud-

native primitives. One of those primitives is to deploy a container in the cloud.

Bhasi *et al.* [2021] proposed Kraken, a workflow-oriented resource management framework that minimizes the number of containers provisioned for deploying an application in Serverless architectures. Although Kraken represents a step forward in FaaS, it does not take into account data management services, including storage and querying. An important observation is that the cost model proposed by Kraken to estimate the necessary amount of containers can be applied to ORBITER. Yussupov *et al.* [2022] proposed a method and a framework to model and deploy Serverless functions orchestrations.

It uses a Business Process Model and Notation (BPMN), but the approach does not take into account data management issues and assumes that there is a repository (or Data Lake) where data are gathered and stored, *i.e.*, the framework does not support data management transparently. Similarly, Sokolowski *et al.* [2021] proposed μs , a framework to orchestrate Serverless functions to implement an entire system. They performed a study with 73 IT professionals who manually orchestrated their functions due to the lack of existing frameworks. The final framework, however, overviews data management issues.

Another category of frameworks focuses on deploying Machine Learning (ML) applications in Serverless architectures. Suppa *et al.* [2021] proposed a framework to deploy BERT (Bidirectional Encoder Representations from Transformers) models using Amazon AWS Lambda. Since models are typically oversized for Serverless architectures, the approach uses knowledge distillation to reduce their size. This approach is representative of ML applications but disregards characteristics expected for other application types. Analogously, the approach of Christidis *et al.* [2020] introduces a series of techniques to convert ML codebase to functions in a Serverless architecture, lacking efficient and modular support for data management.

On the flip side of the coin, other approaches specifically target data management, disregarding other functionalities, such as service orchestration. For instance, Wang *et al.* [2020] proposed a low latency ephemeral storage service based on *cache* to support data persistence in Serverless architectures, while Perron *et al.* [2020] proposed mechanisms for executing queries as functions to reduce latency and the overhead of the Serverless paradigm. Despite advances, these approaches focus only on supporting data storage and not the entire application development.

Table 2. Published works on “Big Data Application Deployment in Serverless”, their publishing vehicle, where C stands for a conference, J for Journal, and BC for Book Chapter.

| Publication | Journal/Conference Name | Type | Year |
|----------------------------------|--|------|------|
| Kuhlenkamp <i>et al.</i> [2019] | IEEE/ACM International Conference on Utility and Cloud Computing | C | 2019 |
| Castellanos <i>et al.</i> [2019] | 13th European Conference on Software Architecture | C | 2019 |
| Christidis <i>et al.</i> [2020] | IEEE Access | J | 2020 |
| Wang <i>et al.</i> [2020] | 18th USENIX Conference on File and Storage Technologies | C | 2020 |
| Perron <i>et al.</i> [2020] | ACM SIGMOD/PODS Conference | C | 2020 |
| Sokolowski <i>et al.</i> [2021] | ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering | C | 2021 |
| Suppa <i>et al.</i> [2021] | 2021 Conference of the North American Chapter of the Association for Computational Linguistics | C | 2021 |
| Bhasi <i>et al.</i> [2021] | ACM Symposium on Cloud Computing, SoCC '21 | C | 2021 |
| Wen <i>et al.</i> [2021] | ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering | C | 2021 |
| Bhat <i>et al.</i> [2022] | IEEE/ACM 8th International Conference on Big Data Computing, Applications and Technologies | C | 2022 |
| Yussupov <i>et al.</i> [2022] | Software Practice and Experience | J | 2022 |

6 Conclusions and Future Work

Despite gaining a lot of traction in recent years, the use of the Serverless computing paradigm to process big data still presents relevant challenges, such as the need to generate container images for deployment and the lack of portability between cloud providers. In this article, we tackled this portability gap by introducing ORBITER, a lightweight framework that automatically deploys applications following the Serverless model in multiple cloud providers.

ORBITER was evaluated with two case studies in different domains (*i.e.*, *Crime Hot Spots Analysis* and *Rainfall Analysis*), showing acceptable computing performance with promising results. As future work we plan to introduce a cost model to define the best configurations for each service according to user constraints (*e.g.*, deadline, and budget). We also plan to incorporate different tools and libraries in the ORBITER ecosystem to replace Spark, *etc.* This future work is important in order to offer options to the users to configure their applications.

Acknowledgements

The authors extend their gratitude to Wagner Santos for his invaluable assistance in validating the hot spot analysis application within the Military Police of Rio de Janeiro. Additionally, the authors would like to express their appreciation to Lorena Christina Nascimento and Fabio Victoriuno for their valuable contributions in validating the Rainfall Analysis application. Their support and expertise have been instrumental in ensuring the accuracy and effective-

ness of the respective applications, and their collaborative efforts have significantly enriched the quality of this research.

Funding

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. This research was also funded by CNPq and FAPERJ (Grant numbers SEI-016517/2021 and E-26/202.806/2019).

Authors’ Contributions

Joao Loureiro, José Maria Monteiro, Marcos Bedo, and Daniel de Oliveira contributed to the conception of this study. Joao Loureiro performed the experiments. Joao Loureiro is the main contributor and writer of this article. All authors read, write, and revised the final article.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The datasets generated and/or analyzed during the current study are available in <https://github.com/UFFeScience/orbiter>.

References

- Armbrust, M., Das, T., Paranjpye, S., Xin, R., Zhu, S., Ghodsi, A., Yavuz, B., Murthy, M., Torres, J., Sun, L., Boncz, P. A., Mokhtar, M., Hovell, H. V., Ionescu, A., Luszczak, A., Switakowski, M., Ueshin, T., Li, X., Szafranski, M., Senster, P., and Zaharia, M. (2020). Delta lake: High-performance ACID table storage over cloud object stores. *Proc. VLDB Endow.*, 13(12):3411–3424. DOI: 10.14778/3415478.3415560.
- Beetz, F. and Harrer, S. (2022). Gitops: The evolution of devops? *IEEE Softw.*, 39(4):70–75. DOI: 10.1109/MS.2021.3119106.
- Begoli, E., Camacho-Rodríguez, J., Hyde, J., Mior, M. J., and Lemire, D. (2018). Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18*, page 221–230, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3183713.3190662.
- Behm, A., Palkar, S., et al. (2022). Photon: A fast query engine for lakehouse systems. *SIGMOD '22*, page 2326–2339, New York, NY, USA. ACM. DOI: 10.1145/3514221.3526054.
- Bhasi, V. M., Gunasekaran, J. R., Thinakaran, P., Mishra, C. S., Kandemir, M. T., and Das, C. (2021). Kraken: Adaptive container provisioning for deploying dynamic dags in serverless platforms. In *Proceedings of the ACM Symposium on Cloud Computing, SoCC '21*, page 153–167, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3472883.3486992.
- Bhat, A., Park, H., and Roy, M. (2022). Evaluating serverless architecture for big data enterprise applications. In *2021 IEEE/ACM 8th International Conference on Big Data Computing, Applications and Technologies (BDCAT '21)*, BDCAT '21, page 1–8, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3492324.3494169.
- Camacho-Rodríguez, J., Chauhan, A., Gates, A., Koifman, E., O'Malley, O., Garg, V., Haindrich, Z., Shelukhin, S., Jayachandran, P., Seth, S., Jaiswal, D., Bouguerra, S., Bangarwa, N., Hariappan, S., Agarwal, A., Dere, J., Dai, D., Nair, T., Dembla, N., Vijayaraghavan, G., and Hagleitner, G. (2019). Apache hive: From mapreduce to enterprise-grade big data warehousing. In Boncz, P. A., Manegold, S., Ailamaki, A., Deshpande, A., and Kraska, T., editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1773–1786. ACM. DOI: 10.1145/3299869.3314045.
- Castellanos, C., Varela, C. A., and Correal, D. (2019). Measuring performance quality scenarios in big data analytics applications: A devops and domain-specific model approach. In *Proceedings of the 13th European Conference on Software Architecture - Volume 2, ECSA '19*, page 165–172, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3344948.3344986.
- Christidis, A., Moschoyiannis, S., Hsu, C.-H., and Davies, R. (2020). Enabling serverless deployment of large-scale ai workloads. *IEEE Access*, 8:70150–70161. DOI: 10.1109/ACCESS.2020.2985282.
- de Carvalho, L. R. and de Araújo, A. P. F. (2020). Performance comparison of terraform and cloudify as multicloud orchestrators. In *CCGRID*, pages 380–389. IEEE. DOI: 10.1109/CCGrid49817.2020.00-55.
- de Oliveira, D., Baião, F. A., and Mattoso, M. (2010). Towards a taxonomy for cloud computing from an e-science perspective. In Antonopoulos, N. and Gillam, L., editors, *Cloud Computing, Principles, Systems and Applications*, Computer Communications and Networks, pages 47–62. Springer. DOI: 10.1007/978-1-84996-241-4_3.
- de Oliveira, D. E. M., Porto, F., Boeres, C., and de Oliveira, D. (2021). Towards optimizing the execution of spark scientific workflows using machine learning-based parameter tuning. *Concurr. Comput. Pract. Exp.*, 33(5). DOI: 10.1002/cpe.5972.
- Godlove, D. (2019). Singularity: Simple, secure containers for compute-driven workloads. In Furlani, T. R., editor, *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning), PEARC 2019, Chicago, IL, USA, July 28 - August 01, 2019*, pages 24:1–24:4. ACM. DOI: 10.1145/3332186.3332192.
- González, L. M. V., Rodero-Merino, L., Caceres, J., and Lindner, M. A. (2009). A break in the clouds: towards a cloud definition. *Comput. Commun. Rev.*, 39(1):50–55. DOI: 10.1145/1496091.1496100.
- Hassan, H. B., Barakat, S. A., and Sarhan, Q. I. (2021). Survey on serverless computing. *J. Cloud Comput.*, 10(1):39. DOI: 10.1186/s13677-021-00253-7.
- Hellerstein, J. M., Faleiro, J. M., et al. (2019). Serverless computing: One step forward, two steps back. In *CIDR*. www.cidrdb.org.
- Kikuchi, G., Amemiya, M., and Shimada, T. (2012). An analysis of crime hot spots using GPS tracking data of children and agent-based simulation modeling. *Ann. GIS*, 18(3):207–223. DOI: 10.1080/19475683.2012.691902.
- Kimball, R. and Ross, M. (2002). *The Data Warehouse Toolkit: The complete guide to dimensional modeling*. Wiley, New York.
- Kuhlenkamp, J., Werner, S., Borges, M. C., El Tal, K., and Tai, S. (2019). An evaluation of faas platforms as a foundation for serverless big data processing. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing, UCC'19*, page 1–9, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3344341.3368796.
- Liu, X. (2021). *Learning from the Data Heterogeneity for Data Imputation*. PhD thesis, Arizona State University, Tempe, USA.
- Loureiro, J. and de Oliveira, D. (2022). Orbiter: um arcabouço para implantação automática de aplicações big data em arquiteturas serverless. In *Anais do XXXVII Simpósio Brasileiro de Bancos de Dados*, pages 379–384, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbbd.2022.225369.
- Lourenço, V., Mann, P., Guimaraes, A., Paes, A., and de Oliveira, D. (2018). Towards safer (smart) cities:

- Discovering urban crime patterns using logic-based relational machine learning. In *2018 International Joint Conference on Neural Networks, IJCNN 2018, Rio de Janeiro, Brazil, July 8-13, 2018*, pages 1–8. IEEE. DOI: 10.1109/IJCNN.2018.8489374.
- Mampage, A., Karunasekera, S., and Buyya, R. (2021). A holistic view on resource management in serverless computing environments: Taxonomy, and future directions. *CoRR*, abs/2105.11592.
- Marchioro, T., Kazlouski, A., and Markatos, E. P. (2022). How to publish wearables' data: Practical guidelines to protect user privacy. In Sérroussi, B., Weber, P., Dhombres, F., Grouin, C., Liebe, J., Pelayo, S., Pinna, A., Rance, B., Sacchi, L., Ugon, A., Benis, A., and Gallos, P., editors, *Challenges of Trustable AI and Added-Value on Health - Proceedings of MIE 2022, Medical Informatics Europe, Nice, France, May 27-30, 2022*, volume 294 of *Studies in Health Technology and Informatics*, pages 949–950. IOS Press. DOI: 10.3233/SHTI220635.
- Miell, I. and Sayers, A. (2019). *Docker in practice*. Simon and Schuster.
- Mizutori, M. and Guha-Sapir, D. (2020). Human cost of disasters 2000-2019. Technical report, United Nations Office for Disaster Risk Reduction.
- Nandury, S. V. and Begum, B. A. (2016). Strategies to handle big data for traffic management in smart cities. In *ICACCI 2016, India*, pages 356–364. IEEE. DOI: 10.1109/ICACCI.2016.7732072.
- Nascimento, L. C., Chagas, R. P., Lage, M., and de Oliveira, D. (2022). Beyond click-and-view: a comparative study of data management approaches for interactive visualization. *J. Inf. Data Manag.*, 13(3). DOI: 10.5753/jidm.2022.2513.
- Nascimento, L. C., Knust, L., Santos, R., Sá, B., Moreira, G., Freitas, F., Moura, N., Lage, M., and Oliveira, D. (2021). Análise de dados pluviométricos multi-fonte baseada em técnicas olap e de visualização: uma abordagem prática. In *Anais do XII Workshop de Computação Aplicada à Gestão do Meio Ambiente e Recursos Naturais*, pages 1–10, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wcama.2021.15731.
- Perron, M., Fernandez, R. C., DeWitt, D. J., and Madden, S. (2020). Starling: A scalable query engine on cloud functions. In *SIGMOD J, June 14-19, 2020*, pages 131–141. ACM. DOI: 10.1145/3318464.3380609.
- Raasveldt, M. and Mühleisen, H. (2019). Duckdb: an embeddable analytical database. In Boncz, P. A., Manegold, S., Ailamaki, A., Deshpande, A., and Kraska, T., editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1981–1984. ACM. DOI: 10.1145/3299869.3320212.
- Sá, B. C., Muller, G., Banni, M., Santos, W., Lage, M., Rosseti, I., Frota, Y., and de Oliveira, D. (2022). Polroute-ds: a crime dataset for optimization-based police patrol routing. *J. Inf. Data Manag.*, 13(1). DOI: 10.5753/jidm.2022.2355.
- Sokolowski, D., Weisenburger, P., and Salvaneschi, G. (2021). Automating serverless deployments for devops organizations. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021*, page 57–69, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3468264.3468575.
- Sousa, F. (2020). Computação serverless e gerenciamento de dados. In *Anais do XXXV Simpósio Brasileiro de Bancos de Dados*, pages 199–204, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbbd.2020.13641.
- Suppa, M., Benešová, K., and Švec, A. (2021). Cost-effective deployment of BERT models in serverless environment. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, pages 187–195, Online. Association for Computational Linguistics. DOI: 10.18653/v1/2021.naacl-industry.24.
- Sznaier, M., Camps, O. I., Ozay, N., and Lagoa, C. M. (2014). Surviving the upcoming data deluge: A systems and control perspective. In *53rd IEEE Conference on Decision and Control, CDC 2014, Los Angeles, CA, USA, December 15-17, 2014*, pages 1488–1498. IEEE. DOI: 10.1109/CDC.2014.7039611.
- Thorndahl, S. and Willems, P. (2008). Probabilistic modelling of overflow, surcharge and flooding in urban drainage using the first-order reliability method and parameterization of local rain series. *Water Research*, 42(1):455–466. DOI: <https://doi.org/10.1016/j.watres.2007.07.038>.
- Thurgood, B. and Lennon, R. G. (2019). Cloud computing with kubernetes cluster elastic scaling. In *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems, ICFNDS '19*, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3341325.3341995.
- Wang, A., Zhang, J., et al. (2020). Infinicache: Exploiting ephemeral serverless functions to build a cost-effective memory cache. In Noh, S. H. and Welch, B., editors, *USENIX FAST*, pages 267–281. USENIX Association.
- Wen, J., Chen, Z., Liu, Y., Lou, Y., Ma, Y., Huang, G., Jin, X., and Liu, X. (2021). An empirical study on challenges of application development in serverless computing. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021*, page 416–428, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3468264.3468558.
- Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14*. ACM.
- Yussupov, V., Soldani, J., Breitenbücher, U., and Leymann, F. (2022). Standards-based modeling and deployment of serverless function orchestrations using BPMN and TOSCA. *Softw. Pract. Exp.*, 52(6):1454–1495. DOI: 10.1002/spe.3073.
- Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S.,

- Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. (2016). Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65. DOI: 10.1145/2934664.
- Zaharia, M. A. (2013). *An Architecture for and Fast and General Data Processing on Large Clusters*. PhD thesis, University of California, Berkeley, USA.
- Zhang, P., Xing, L., Yang, N., Tan, G., Liu, Q., and Zhang, C. (2018). Redis++: A high performance in-memory database based on segmented memory management and two-level hash index. In Chen, J. and Yang, L. T., editors, *IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications, ISPA/IUCC/BDCLOUD/SocialCom/SustainCom 2018, Melbourne, Australia, December 11-13, 2018*, pages 840–847. IEEE. DOI: 10.1109/BDCLOUD.2018.00125.