FunDa: Towards Serverless Data Analytics and In Situ Query Processing

Suvam Kumar Das*
University of New Brunswick
Fredericton, Canada
suvam.das@unb.ca

Ronnit Peter*
University of New Brunswick
Fredericton, Canada
ronnit.peter@unb.ca

Xiaozheng Zhang*
University of New Brunswick
Fredericton, Canada
xz.zhang@unb.ca

Suprio Ray
University of New Brunswick
Fredericton, Canada
sray@unb.ca

ABSTRACT

Serverless is a cloud computing paradigm that offers unique advantages to the users due to its pay-what-you-use model. It is particularly suitable for running ephemeral short running tasks. However, this framework is not well-suited for stateful long running data analytics and query processing tasks. Although several recent projects have proposed serverless query processing or data analytics systems that are based on AWS Lambda, they are limited by its constraints and the inherent limitations of serverless.

We propose *FunDa*, an on-premises serverless data analytics framework, which aims to address the limitations of existing approaches. *FunDa* extends our previously proposed system for unified data analytics and in situ SQL query processing called DaskDB. We evaluate our prototype with two different workloads with three scale factors. While our experimental results are quite promising, they also reveal significant future research opportunities.

CCS CONCEPTS

• **Networks → Cloud computing**; **Cloud computing**; • **Computer systems organization → Cloud computing**; **Cloud computing**; • **Software and its engineering → Cloud computing**; **Cloud computing**; • **Information systems → Data analytics**.

KEYWORDS

Serverless, Data Analytics, In situ Query Processing, DaskDB

ACM Reference Format:
Suvam Kumar Das, Ronnit Peter, Xiaozheng Zhang, and Suprio Ray. 2024. *FunDa*: Towards Serverless Data Analytics and In Situ Query Processing. In *International Workshop on Big Data in Emergent Distributed Environments (BiDEDE '24), June 09–15, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3663741.3664788

*All three authors contributed equally to this research, ordered by last name.

1 INTRODUCTION

Cloud computing enabled Internet service providers to deploy their services without worrying about owning or managing a physical hardware infrastructure. Serverless cloud computing (or serverless, for short) is considered as the next step in the evolution of cloud computing. In a traditional cloud computing environment, termed as 'serverful' [15], a user needs to over-provision the system to deal with unforeseen workload spikes and to manage workload peaks. This requires paying for resources even when they are not utilized. Serverless is characterized by the lack of a need to reserve cloud resources. The pay-what-you-use model of serverless is quite attractive to the users. Function-as-a-service (FaaS) is a serverless computing model in which a developer can upload a function, called cloud function (CF). This CF can be invoked and launched as part of a Web service. High level abstractions offered by serverless hide the details of the underlying hardware resources, and alleviates the need to configure, and manage virtual or physical servers. Autoscaling is provided by the serverless model by launching resources as they are needed to handle a workload surge, after which unused resources are scaled back. These advantages are naturally driving a movement towards serverless.

While serverless has shown to offer significant benefits for running short running jobs, it is not designed for general purpose data analytics. Serverless platforms, such as AWS Lambda was originally conceived to be a platform for elastic execution of ephemeral jobs. It limited the running time of a job to 15 minutes. In contrast, data analytics, including analytical query processing, and statistical and machine learning (ML) based applications may entail long running tasks requiring significant disk IO and compute resources. These kinds of workloads are not ideally suited for serverless. Since the underlying resources are launched with every job and reclaimed after the job's completion in a serverless platform, each job is run with a cold start initialization phase. Hence, no persistent state is maintained and caching of hot data is not practical. Another issue with serverless is that there is no support for direct communication among functions. Consequently, existing serverless platforms cannot be utilized out-of-the-box for data analytics. Recently, researchers have started to investigate how to address these challenges. A few research projects have built experimental prototypes. These include, Flint [11] and MetaQ [21] among others. These systems utilize AWS Lambda FaaS functions.

While a commercial FaaS offering, such as AWS Lambda, is a popular platform to leverage, there are several limitations of this, particularly for data analytics and query processing. **First**, a system like AWS Lambda imposes certain constraints, including a limit on the execution time, limited size (maximum 10GB RAM), a lack of persistent local storage, and inter-instance communication facilities [15]. This forced a few projects [12, 14] to use S3 cloud storage for intermediate data processing. Hence, there is a need to offer more flexibility and better control to the end users to design a serverless data analytics platform. **Second**, commercial serverless offerings from AWS, Azure and Google Cloud charge a premium price for FaaS. This may be acceptable to the users for short-running ephemeral jobs, but for long running data analytics and query processing this may incur significant expenses to the end users. **Third**, data privacy and security are major concerns for many enterprises. For services involving highly sensitive data, it may be preferable not to upload the data to a public cloud. Therefore, for some users it may be preferable to use their own serverless data analytics platform than to utilize a public cloud [13]. **Finally,** serverless data analytics frameworks and research prototypes are still in their early stages of development. There are still many research opportunities to pursue in this direction [1, 17].

In this paper, we report our current effort in developing a general purpose serverless data analytics system called *F*u*n*Da. Our focus is on implementing an serverless platform for unified data analytics and in situ analytical query processing. We leverage our previously proposed DaskDB system [20]. DaskDB is a distributed data science system that supports unified data analytics and in situ SQL query processing on heterogeneous data sources. DaskDB enables users to embed SQL queries with Python applications that can be directly run on raw data files. Moreover, users can create User-Defined Functions (UDF) using Dask's Python API [5]. So, it can be easily integrated with most existing Python data science applications. DaskDB also supports geospatial query processing and analytics [4].

The system architecture of *F*u*n*Da is presented in Figure 1. *F*u*n*Da extends DaskDB's capability with serverless functionalities by leveraging Fn project [8] serverless framework. Fn is an open-source container-native serverless framework that can run on a cloud platform or on-premises. It supports several programming languages, including Javacript, Java and Python. By harnessing Fn to initiate Docker [6] containers, *F*u*n*Da dynamically scales computing resources, embodying true serverless flexibility while encapsulating DaskDB's analytical power.

We conduct experimental evaluation with two workloads: three queries from TPC-H benchmark and two tasks from the DaskDB UDF benchmark running machine learning tasks. We evaluated these workloads on three scale factors: 1, 5 and 10. Our experimental evaluation reveals that while the query/analytics task execution times are encouraging, but there are significant overheads due to startup time, which includes the container launch time and preprocessing time in the cold start initialization phase. The preprocessing time includes time to load the data files from disk, time to load libraries among others. Due to the stateless nature of a serverless system, this startup time is unavoidable with the current FaaS platforms, as previous research also noted [21]. Our findings suggest that there are significant research opportunities, including developing caching, and local persistent storage layers and fast inter-container communication primitives.

The rest of the paper is organized as follows. In Section 2 we discuss related work. In Section 3 we present our approach and system organization. Our experimental evaluation is described in Section 4. The paper is concluded in Section 6.

2 RELATED WORK

In this section we describe previous work that are related to our proposed system.

2.1 Scalable data analytics

Hadoop [9], which is based on MapReduce paradigm, is a popular system for distributed data analytics. Hive [18] builds on Hadoop to provide analytical query execution functionalities. Due to the inefficiencies with the Hadoop framework, Apache Spark [16] framework was proposed. It supports both analytical query processing, and statistical and machine learning in a distributed execution environment. However, there are some limitations with Spark. First, data analytics functions are usually written as UDFs, which are restricted to use in Spark APIs and data types. This is inconvenient for users who are not experienced in Spark. Secondly, already existing Python UDFs cannot be directly imported to Spark, but it has to be re-written again using their own data types. This is not ideal for a situation when a user already has thousands of lines of existing UDF code that needs to be used as is. DaskDB [20] was introduced to address the issues with Spark. For a given SQL query, DaskDB emits Python code comprised of Dask operators, which are then submitted to Dask Scheduler for execution. Since DaskDB emits Python code, existing Python UDFs can be integrated with it easily without much effort.

2.2 Serverless cloud data processing

The topic of serverless cloud computing has been an active area of research in recent years. However, only a few projects have explored how to develop serverless data analytics and/or query processing. PyWren [10] is one of the earliest systems to leverage serverless architecture with stateless functions as the abstraction for data analytics. It exposes a Python based map API on top of AWS Lambda. PyWren supports three computation patterns, namely, Map, Map + monolithic Reduce, and MapReduce with a remote storage (e.g. S3) for data shuffling. Unlike PyWren, Flint [11] targets general purpose data analytics with Apache Spark and utilizes AWS Lambda to enable serverless data analysis. It requires all the input data to reside in an S3 bucket. The output is also written to another S3 bucket. Flint utilizes most of the existing Spark components, while providing a serveless implementation of the backend-scheduler. .

Starling [14] and Lambada [12] are both SQL query execution frameworks using AWS Lambda functions. Starling uses several techniques to support serverless query processing. It uses Amazon S3 to shuffle data. Its coordinator generates C++ source code for an input query, compiles it into a executable that is packaged, compressed and uploaded to AWS Lambda. It supports optimization of a query for cost or latency by adjusting the number of function invocations at each task stage. Lambada also employs several solutions to address the challenges of serverless distributed query execution. For fast batch-start of many serverless workers it uses a tree-based invocation. Besides, it proposes a purely serverless exchange operator, which communicates through external storage based on S3.

Pixels-Turbo [2] is a distributed query engine that offers a hybrid cloud approach where elasticity can be provided to a system instantiated on long-running VMs by leveraging AWS Lambda function. The long-running VMs handle stable workloads, whereas workload spikes and unpredictable workload changes are gracefully handled by AWS Lambda. MetaQ [21] proposed the idea of *ephemeral per-query engines* (EPQE) on serverless architectures using AWS Lambda, where given a query, a query engine is dynamically instantiated. All of the above-mentioned systems leveraged AWS Lambda functions.

The goal of *FunDa* is to offer a serverless in situ data analytics framework over DaskDB that can be deployed both in a commercial cloud and on-premises private cloud. Unlike serverless query processing systems Starling, Lambada and Pixels-Turbo, our system aims to support both query processing and analytical workloads involving statistical/ML analysis. Moreover, these systems use custom-built execution engines. To that end, *FunDa* shares a similar vision as the system recently proposed by Wawrzoniak et al [22]: to support an existing unmodified data analytics engine over a serverless platform. However their approach, like previous approaches, is based on AWS Lambda serverless framework. We subscribe to the idea [2] that using an existing commercial serverless platform such as AWS Lambda, comes at the expense of relinquishing the control over the hardware and execution environment to the cloud service provider. *FunDa* is based on the serverless framework Fn, which can be deployed on-premises and potentially adopted by commercial cloud providers.

3 APPROACH

In this section we describe the system organization of *FunDa*. It consists of *Client*, *Service Coordinator*, *Driver*, *DaskDB Scheduler* and *DaskDB Worker*. The architecture of *FunDa* is shown in Fig. 1. We will first describe the individual components followed by the flow of execution, once a job is submitted to *FunDa*.

3.1 Client

The *FunDa* Client provides a command line interface and utilizes curl [3] for a user to launch SQL query or data analysis jobs and monitor the job status information during the execution. When the user launches a job, the Client's request is communicated to Service Coordinator by sending out a HTTP request. The HTTP request encapsulates all the necessary information needed by the Service Coordinator to execute the job. When the job finishes, the Client receives a HTTP response back from the Service Coordinator, which contains status information and results corresponding to the job execution.

3.2 Service Coordinator

An incoming job is appended into a job queue, which is maintained by the Service Coordinator. When there are enough resources for a new job, the next job from the job queue is dequeued, the necessary resources are provisioned and the job execution is initiated. The process involves an automated Docker image launching. Necessary operational data for DaskDB, such as job metadata, configuration, flow control, scaling information, are consolidated into Docker images through dockerfiles, providing easy deployment and flexible scaling.

3.3 Driver

The Driver is responsible for launching the containers to execute a job and control the process. The Driver performs a sequence of operations, in which each operation materializes a Docker container. Utilizing Docker Swarm [7], the Driver efficiently orchestrates the deployment, scaling, and management of these containers across multiple nodes. Among the commonly used container operations, the *InitScheduler* operation configures and operates the DaskDB Scheduler, *InitWorkers* leverages Docker Swarm to manage and scale the number of workers dynamically. It keeps track of the job status as well as progress information. The *InitWorkers* operation manipulates both the DaskDB Scheduler and Worker containers, and registers the Workers with the Scheduler. The *ShutdownWorkers* operation monitors the job status and releases the system resources by detaching the Worker containers from the DaskDB Scheduler. It also returns the status information of the DaskDB cluster back to Service Coordinator and notify the completion of a job.

3.3.1 *Container Orchestration with Docker Swarm.*

To optimize the deployment, scaling, and management of containers across *FunDa*'s architecture, Docker Swarm is employed as the primary orchestration tool. Docker Swarm allows *FunDa* to handle container operations dynamically, adjusting to the workload demands efficiently and ensuring high availability, and resilience. Integrating Docker Swarm into *FunDa*'s architecture not only bolsters its scalability and reliability but also upholds the principles of serverless computing by abstracting the complexity of hardware resource management from end-users.

3.4 DaskDB Scheduler

The Scheduler container is initiated through one of the operations from the Driver and is kept running beyond the lifetime of any single job. An analytics/query processing job is submitted by the Driver to the Scheduler, which is responsible for distributing the execution of the sub-tasks of a job to the Worker containers and gather partial results to produce the final result. It does so by generating a task graph where each node in the graph is a normal Python function and edges between nodes are normal Python objects representing the data to be transferred between two operations. After the task graphs are generated, they are executed in parallel by the Worker containers.

3.5 DaskDB Worker

During the start of execution of each sub-task corresponding to a job, the Worker containers are initiated by the relevant operation from the Driver. The worker executes the sub-tasks that are assigned to it by the Scheduler. Each Worker can execute multiple sub-tasks concurrently, depending on the available resources (CPU cores, memory, etc.). It stores and manages a portion of the intermediate result that is generated in the computations.

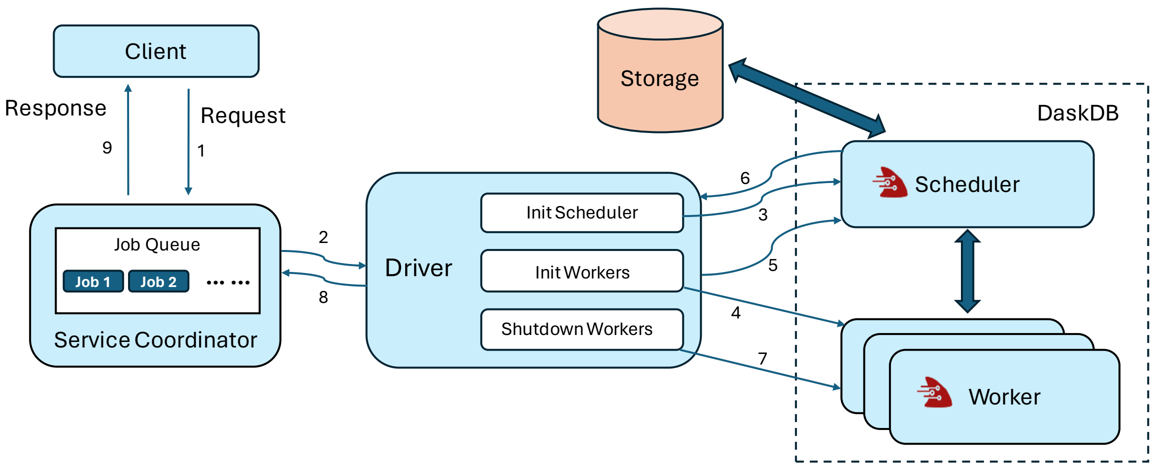

Figure 1: System architecture of *FunDa*

3.6 Flow of execution of a job

In this section, we explain in details about the steps (shown by numbers in Fig. 1) involved when a job is submitted for execution to *Fun*Da.

(1) The Client receives a query or analytical job from the user and sends it to the Service Coordinator, which places it into its Job Queue.
(2) The Service Coordinator pulls a job request from its Job Queue and sends it to the Driver.
(3) The *InitScheduler* operation within the Driver initializes the DaskDB Scheduler for the first time after receiving a job request, after which it is kept running.
(4) The DaskDB Workers are initialized by invoking the *InitWorker* operation of the Driver, with Docker Swarm facilitating the deployment and scaling of these worker nodes.
(5) The Driver submits the job to the DaskDB Scheduler for execution. The Scheduler communicates with the workers as well as the data Storage for completion of the job (shown in blue double-headed solid arrows).
(6) Once the Scheduler finishes execution of the job, the Driver is notified about the job completion status: (*SUCCESS* if the job was successfully completed, else *ERROR* with an error code).
(7) Regardless of the job completion status, the DaskDB workers are stopped using the *ShutdownWorker* operation of Driver, with the Docker Swarm ensuring a graceful shutdown of these resources.
(8) The Driver informs the Service Coordinator about the job completion status.
(9) The Service Coordinator sends the final response to the Client based on the job completion status.

4 EXPERIMENTAL EVALUATION

In this section, we present our experimental evaluation. The experiments are conducted using the TPC-H benchmark [19] and a custom UDF benchmark. The custom UDF benchmark uses slightly modified datasets from the TPC-H benchmark and includes two well-known data analytics tasks, Linear Regression (LR) and KMeans. The experiments are evaluated at three different scale factors (SF) of 1, 5 and 10. In this context, scale factor refers to the size of the raw data processed and the corresponding processing load. Scale Factor 1 (SF1) corresponds to about 1GB of data and and is used to assess the system's baseline performance. Scale Factor 5 (SF5) and Scale Factor 10 (SF10) progressively increase the dataset size by five and ten times, respectively. This allows us to evaluate how *Fun*Da scales with increasing data volumes and processing demands.

The details of the experimental setting are in Table 1. The data analytics queries that are used as part of the UDF benchmark are shown in Table 2. To run the experiments, we use a four-node cluster, where each node runs Ubuntu 20.04 LTS OS, Intel(R) Core(TM) i5-2400 CPU having 4 cores, and 8GB of RAM. We measured the

	Queries/Tasks	**Scale Factor**
TPC-H Benchmark	Q1, Q3 and Q10	1, 5 and 10
UDF-Benchmark	LR, K-Means	

Table 1: Experimental setting

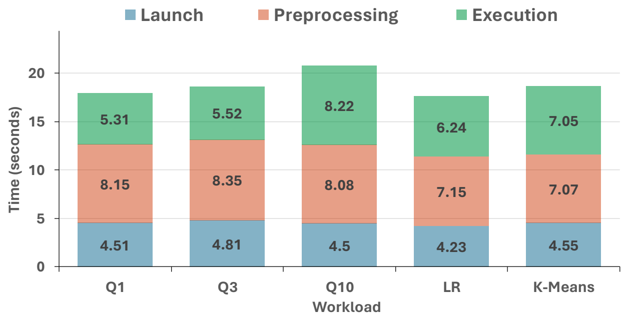

Figure 2: Scale Factor 1

Tasks	Query
LR	select **myLinearFit(l_discount, l_tax)** from orders, lineitem where l_orderkey = o_orderkey and o_orderstatus = 'F'
K-Means	select **myKMeans(l_discount, l_tax)** from orders, lineitem where l_orderkey = o_orderkey and o_orderstatus = 'F'

Table 2: UDF benchmark (queries with custom UDF)

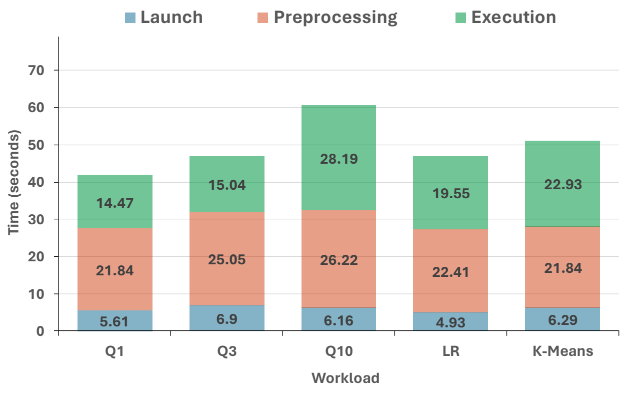

Figure 3: Scale Factor 5

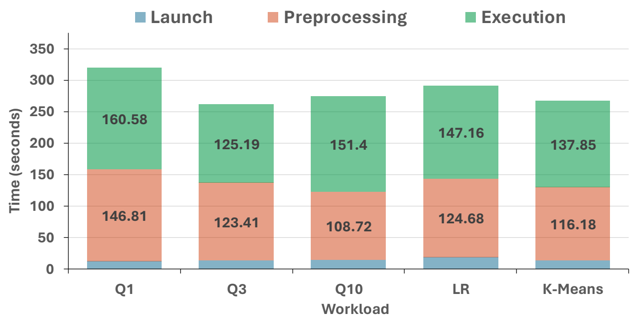

Figure 4: Scale Factor 10

launch time of the worker nodes, *pre-processing time* (which includes reading the relevant data files from the storage after a job (query/task) is issued), and the *execution time*. Every time a job was submitted all the workers were started and was then stopped once execution was over. However, the scheduler was kept up and running for the entire duration of the experiments. For all the queries, the TPC-H datasets were stored and read from a network-mounted file system (NFS). The plots corresponding to the experiments are shown in Figures 2, 3 and 4.

From Figure 2 it can be seen that with SF1, 22–26% of the overall execution time was spent in launching, 38–45% was spent in pre-processing and the remaining 30–40% was devoted to query execution. For instance, in the case of TPC-H query Q1, 25% of the time was spent for launching (4.5 seconds), 45% for pre-processing (8.2 seconds) and 30% for query execution (5.3 seconds). For LR, the values were 24%, 41% and 35% respectively.

From Figure 3 it can be seen that with SF5, 10–15% of the overall execution time was spent in launching, 43–53% was spent in query pre-processing and the remaining 32–47% was for query execution. For instance, with Q1, 13% of the time was spent for launching , 52% for pre-processing and 35% for query execution. With LR, the values were 12%, 43% and 45% respectively.

Similarly, from Figure 4 we can see that with SF10, 4–6% of the overall time was spent in launching, 40–47% was spent in query pre-processing and the remaining 48–55% was spent in query execution. Since the percentage of execution time involved in launching is small, its corresponding values were ignored from the plot as they were not properly visible. For example, in Q1 4% of the overall time was spent in launching, 46% for pre-processing and 50% for query execution. With LR, the values were 5%, 43% and 52% respectively.

Overall, it can be observed that the proportion of time spent in pre-processing and execution increases, whereas the launching time remains almost fixed within a range as it is independent of the dataset size. When evaluating the performance at each scale factor, it is evident that as the scale factor increases, entailing larger data sizes and heavier processing loads, there is a proportional increase in the preprocessing and execution times. This showcases *FunDa*'s capacity to handle increasing data volumes, which is critical for practical deployments in data-intensive scenarios.

5 DISCUSSION

While developing *FunDa*, our focus was directed towards establishing a robust architectural foundation and ensuring the framework's operational stability. Due to constraints in time and resources, a comprehensive experimental evaluation with existing serverless platforms, such as AWS Lambda, Azure Functions, and Google Cloud Functions was beyond our current scope. Recognising this limitation, our future work will involve a detailed experimental analysis comparing these platforms' capabilities and limitations against *FunDa*'s unique features. This analysis will not only help in highlighting *FunDa*'s contributions but also in identifying possible areas for future enhancements.

6 CONCLUSION AND FUTURE WORK

In this paper, we have introduced *FunDa*, our on-premises serverless data analytics and query processing framework designed to bridge the gap between existing serverless capabilities. Our evaluation demonstrates that *FunDa*, despite its nascent stage, delivers promising task execution times even in the face of significant overheads associated with launching times. These overheads, while notable, usher opportunities for future research.

The realm of serverless data analytics is still in its early days. A key challenge in this domain arises from the ephemeral nature of serverless resources, which necessitates relinquishing resources after job completion. Therefore, efficient caching and persistence mechanisms for frequently accessed data can substantially reduce query execution times. Moreover, network communication bottlenecks emerge as significant factor influencing response times within serverless architectures. Addressing these bottlenecks will offer a pathway to improve the overall architecture that may facilitate both intra-host and inter-host communications, reducing the container operational costs and resource initialization times.

The preliminary results from *FunDa* not only underscores the framework's potential to transform serverless data analytics but also highlight several promising areas for future work. These include optimizing execution for cold starts, developing serverless data caching techniques, and refining container communication protocols. Together, these avenues provide fertile ground for community collaboration and further research, pointing towards a future where serverless data analytics can be executed more efficiently, economically, and effectively.

ACKNOWLEDGMENTS

This research was supported in part by NSERC Discovery Grant RGPIN-2016-03787 (Ray) and NBIF RAI Grant (Ray).

REFERENCES

[1] Gustavo Alonso, Ana Klimovic, Tom Kuchler, and Michael Wawrzoniak. 2023. Rethinking Serverless Computing: from the Programming Model to the Platform Design. In *VLDB Workshops*.
[2] Haoqiong Bian, Tiannan Sha, and Anastasia Ailamaki. 2023. Using Cloud Functions as Accelerator for Elastic Data Analytics. *Proc. ACM Manag. Data* 1, 2, Article 161 (jun 2023), 27 pages.
[3] curl [n. d.]. curl. https://curl.se/.
[4] Suvam Kumar Das, Ronnit Peter, and Suprio Ray. 2023. Scalable Spatial Analytics and In Situ Query Processing in DaskDB. In *Proceedings of the 18th International Symposium on Spatial and Temporal Data* (, Calgary, AB, Canada,) *(SSTD '23)*. Association for Computing Machinery, New York, NY, USA, 189–193. https://doi.org/10.1145/3609956.3609978
[5] Dask [n. d.]. Dask. https://www.dask.org/.
[6] Docker [n. d.]. Docker. https://www.docker.com/.
[7] Docker Swarm [n. d.]. Docker Swarm. https://docs.docker.com/engine/swarm/.
[8] Fn Project [n. d.]. Fn Project. https://fnproject.io/.
[9] Hadoop [n. d.]. Apache Hadoop. http://hadoop.apache.org/.
[10] Eric Jonas, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the Cloud: Distributed Computing for the 99%. *CoRR* abs/1702.04024 (2017).
[11] Youngbin Kim and Jimmy Lin. 2018. Serverless Data Analytics with Flint. In *CLOUD*. IEEE Computer Society, 451–455.
[12] Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. In *SIGMOD*.
[13] Shoumik Palkar and Matei Zaharia. 2017. DIY Hosting for Online Privacy *(HotNets)*. 1–7.
[14] Matthew Perron, Raul Castro Fernandez, David J. DeWitt, and Samuel Madden. [n. d.]. Starling: A Scalable Query Engine on Cloud Functions. In *SIGMOD*. 131–141.
[15] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J. Yadwadkar, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica, and David A. Patterson. 2021. What serverless computing is and should become: the next phase of cloud computing. *Commun. ACM* 64, 5 (apr 2021), 76–84.
[16] Spark [n. d.]. Apache Spark. https://spark.apache.org/.
[17] Sacheendra Talluri, Nikolas Herbst, Cristina Abad, Tiziano De Matteis, and Alexandru Iosup. 2024. ExDe: Design space exploration of scheduler architectures and mechanisms for serverless data-processing. *Future Generation Computer Systems* 153 (2024), 84–96.
[18] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. 2009. Hive: A Warehousing Solution over a Map-reduce Framework. *Proc. VLDB Endow.* 2, 2 (Aug. 2009), 1626–1629.
[19] TPC-H [n. d.]. TPC-H. https://www.tpc.org/tpch/.
[20] Alex Watson, Suvam Kumar Das, and Suprio Ray. 2021. DaskDB: Scalable Data Science with Unified Data Analytics and In Situ Query Processing. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*.
[21] Michael Wawrzoniak, Rodrigo Bruno, Ana Klimovic, and Gustavo Alonso. 2023. Ephemeral Per-query Engines for Serverless Analytics. In *Joint Proceedings of Workshops at VLDB (CEUR Workshop Proceedings)*.
[22] Michal Wawrzoniak, Gianluca Moro, Rodrigo Bruno, Ana Klimovic, and Gustavo Alonso. 2024. Off-the-shelf Data Analytics on Serverless. In *Conference on Innovative Data Systems Research CIDR*.