Survey of Serverless Workflows

Debarghya Saha
Vrije Universiteit Amsterdam

Sacheendra Talluri
Vrije Universiteit Amsterdam

Matthijs Jansen
Vrije Universiteit Amsterdam

Animesh Trivedi
Vrije Universiteit Amsterdam

Abstract

Serverless computing is a popular paradigm for deploying applications due to the ease of deployment and fine-grained billing offered by serverless platforms like AWS Lambda, Azure Functions and Google Cloud Functions. However, they also have limitations such as an increased latency in execution, limited memory and lack of support for specialized hardware like GPUs. Due to these limitations certain types of applications cannot benefit from the aforementioned perks of serverless architecture, for instance, interactive microservices with strict latency targets, memory and compute intensive workloads like training machine learning models, and scientific workloads that have complex task dependencies. Many workflows have been proposed in existing literature that extend the functionality of existing serverless platforms and allow them to support a wider range of applications. In this literature study we take a look at these workflows, taxonomize the state-of-the-art workflows and analyse current trends in serverless workflow research as well as put forward areas for future research in serverless workflows.

1 Introduction

The paradigm of serverless computing has emerged as a transformative force in the field of cloud computing. It is a cloud computing model that abstracts away the traditional infrastructure management, such as servers and virtual machines, from developers and delegates it to the serverless service providers. In a serverless architecture, developers focus solely on writing code in the form of functions which are event-driven and can be executed in response to specific events or triggers. For complex serverless applications a large number of functions are invoked, for example, Xtract [42] is a serverless application for processing vast collections of scientific files and automatically extracting metadata from diverse file types. These collections, or data lakes, like DataOne [30] have hundreds of thousands of files and Xtract needs to invoke multiple functions per file as well as different functions depending on specific file types. On top of this, it interacts with AWS Relational Database Service to manage state. Managing these complex processes efficiently becomes challenging, and so to facilitate the coordination and composition among functions, cloud providers have rolled out serverless workflow services. AWS offers AWS Step Functions, Azure offers Azure Durable Functions and Google Cloud offers Google Cloud Workflows to manage serverless workflows in their respective platforms enabling developers to only specify the execution logic among functions, without having to deal with complex and error-prone communication and interactions among functions.

While there has been much effort in surveying literature in serverless computing, for example Hellerstein et al. [16] surveyed the challenges and shortcomings of serverless and Eismann et al. [10] surveyed the use cases of serverless, further discussed in Section 2, there has been little work in surveying serverless workflows. Workflows make it easier for the developer to achieve specific functionalities by stringing together smaller tasks and are widely used in other areas of computer science such as scientific workflows [21, 28], multi-tier web service workflows [29], and big data processing workflows such as MapReduce [8] and Dryad [17].

Combining workflows with serverless makes it easier for developers to use serverless for building applications that weren't originally intended to be serverless, like stateful applications and data-intensive applications, while reaping the benefits of serverless such as scale to zero and freedom from provisioning and maintaining servers.

We therefore believe that surveying and developing the field of serverless workflows is important and so the goal of this literature survey is to taxonomize the state-of-the-art serverless workflows by the type of application they support, as well as to look at current and future areas of research in serverless workflows. The method used for conducting this literature survey is detailed in Section 3.

With this survey, we aim to answer the following questions:

RQ1: *Which types of applications benefit from the state-of-the-art serverless workflows?*

Answering this will enable us to construct a taxonomy of the serverless workflows grouped by type of application and give us information about which types of applications can benefit from using state-of-the-art workflows[1] when they get no benefits from using traditional workflows.

RQ2: *What are the current areas of research for different domains in serverless workflows?*
Answering this will provide insights on the trend of research in serverless workflows, as well as help gauge the interest different application domains have in serverless computing.

RQ3: *What are the potential areas for future research in serverless workflows?*
This will aid us in finding out research areas in serverless workflows that are of interest but do not have much research activity.

In line with the research questions defined, the main contributions of this literature survey are:

C1: Constructing a taxonomy of state-of-the-art serverless workflows by the type of application (Section 4, 5).

C2: Finding out current areas of research in serverless workflows by analysing co-occurrence of keywords in publications related to serverless workflows (Section 6).

C3: Exploring areas for research in serverless workflows by performing a trend analysis of keywords in publications related to serverless workflows (Section 7).

2 Related Work

Previous surveys in the field of serverless computing have focused on documenting the challenges and problems of serverless computing, surveying use-cases, or comparing the performance and offerings of various serverless providers. These are briefly outlined below.

Baldini et al. [4] compare commercial serverless platforms of AWS, Microsoft Azure and IBM OpenWhisk as well as open source serverless platforms like OpenLambda, on factors of cost, performance, programming model and supported programming language. They also discuss use cases of serverless computing, giving an idea of when the serverless approach is a good fit and when should it be avoided. Lastly, they document the challenges faced by serverless computing.

Hellerstein et al. [16] document the challenges and shortcomings of serverless computing, mainly focusing on AWS Lambda. They provide three case studies where they compare the same workloads of model training, ML prediction and distributed computing on Lambdas and EC2 instances to illustrate the performance difference between serverless and traditional VM based approaches.

[1]Throughout this survey 'state-of-the-art workflow' is used to refer to the workflows proposed in literature and 'traditional workflow' is used to refer to the workflows used in practice, for example by commercial serverless platforms like AWS.

Shahrad et al. [39] characterize and document the entire production workload of Azure functions between July 15th and July 28th, 2019 by trigger types, invocation frequencies and patterns, and resource needs. This provides valuable insight into the type of workloads a major serverless provider has to deal with, which can aid researchers in making informed decisions about resource allocation, scheduling and cold start mitigation strategies.

Serverless platforms often keep the developers in the dark regarding the underlying hardware that their functions execute on, developers can only configure the amount of RAM of the instance that the function will execute on. Kelly et al. [23] shed some light on the inner workings of AWS Lambda, Google Cloud Functions, Azure Functions and IBM Cloud Functions by identifying the hardware specifications of VMs, measuring function performance with respect to the amount of memory allocated to the function, measuring cold start latencies and amount of cold starts, measuring I/O throughput, and effect of interference on function performance due to large number of users using the platform at a time. It is useful information that is not otherwise available for developers and researchers in choosing serverless platforms.

Kuhlenkamp et al. [27] survey some common assumptions that are associated with serverless computing and contrast them with the reality, for instance it is a common assumption that with serverless computing, functions scale automatically and developers don't have to worry about scaling, however the authors find that while serverless provides autoscaling, the developers have to be careful about the standard provisioning and de-provisioning time of the serverless platform according to their application's needs. They present some challenges that arise from these common assumptions and provide some mitigation strategies for said challenges.

Wen et al. [46] taxonomize the challenges faced by serverless application developers and survey the trend and difficulty of questions about serverless application development on StackOverflow. This offers practical insights and actionable implications for developers, researchers, and cloud providers and emphasizes best practices and explores intriguing research opportunities in the adoption of serverless computing.

Wen et al. [47] also compare the workflow services of AWS Step Functions, Azure Durable Functions, Alibaba Serverless Workflow using dimensions like orchestration, data payload limit and parallelism support, and evaluate performance of these workflows by running identical workloads.

Shafiei et al. [38] survey serverless applications in literature and document the challenges faced by those applications as well as the benefits those applications got from moving to a serverless architecture. This survey offers insight into the main benefits of serverless computing that matter for different domains of applications.

Eismann et al. [10] proposed a systematic process to identify, collect, and characterize serverless use cases. They review 89 use cases of serverless computing from white and grey

literature and open source projects to perform an in-depth characterisation of them, using 24 metrics such as execution pattern, workflow coordination, use of external services, and motivation for adopting serverless. This study provides not only a high level look at serverless applications such as the serverless platform used and the application type, but also low level insights such as the number of functions invoked, type of triggers used and resource usage of these applications.

In contrast to the previous surveys, our work documents the serverless workflows that can be used to overcome the challenges faced by serverless computing that prevent it from being used in a wide variety of applications such as applications with strict latency targets or applications that need to communicate large amounts of data. It also offers insight into interesting areas of focus for future research.

3 Method of Literature Review

For this survey, three approaches were considered - unguided exploration of the material, snowballing [48] and the Systematic Literature Review (SLR) [22]. Unguided exploration involves the straightforward process of extensively reading materials on a given topic, using standard scientific literature repositories and search tools like Google Scholar. The "unguided" aspect of this method arises from the absence of predefined stop and search criteria. SLR defines guidelines for conducting a review, these guidelines are: outlining the research question(s), establishing a predetermined search strategy, and specifying inclusion and exclusion criteria in advance, which help to eliminate bias in selection of literature as these steps are carried out before actually going through the literature. For this literature review we use SLR as it eliminates the inherent bias introduced by unguided traversal, and supplement it with snowballing. The survey was conducted in three phases as defined in the guidelines for SLR by Kitchenham et al. [22] - planning, conducting and reporting, which are detailed in the proceeding sections and illustrated in Figure 1.

3.1 Planning

In this step, we analysed some initial literature on serverless computing using broad keywords like "serverless computing" and "serverless application". The goal of this step is to identify the need for a literature review. Going through the initial literature we identified several shortcomings of the serverless architecture and current serverless offerings, like cold starts and limited execution times, among others, and then used these to identify literature that proposed workflows to minimize or overcome these shortcomings. There are many literature surveys highlighting the challenges of serverless computing and applications that do not benefit from a serverless architecture due to them, as mentioned in section 2, however we did not find any surveys that covered the research

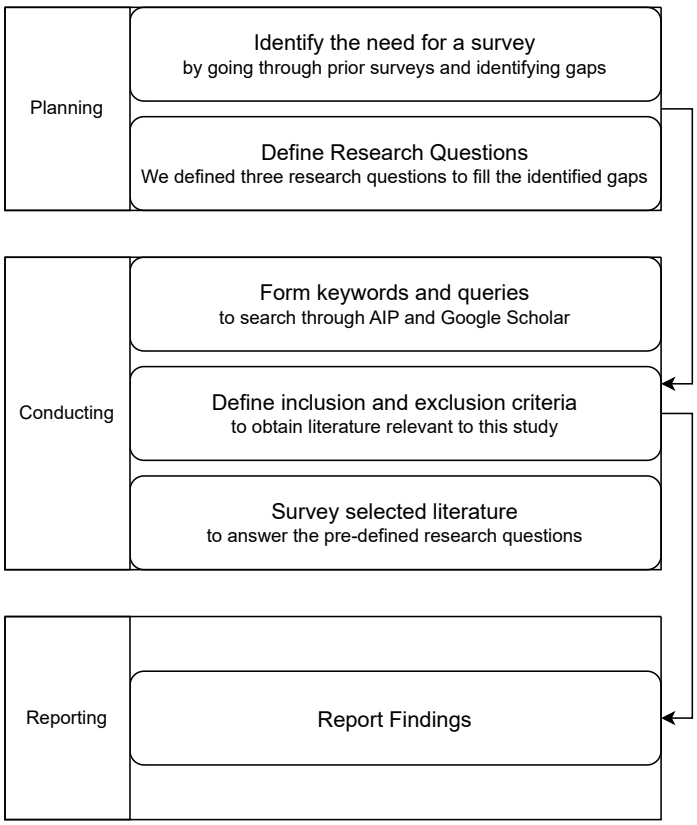

Figure 1: Steps followed for conducting this study.

in serverless workflows and the applications that benefit from these workflows. To fill this gap, the goal of this literature survey is to document the state-of-the-art in serverless workflows and construct a taxonomy of serverless workflows based on the type of application enabled by the workflow. In line with this goal, we formulated three research questions that this survey aims to answer as mentioned earlier in Section 1.

3.2 Conducting

This step involves the selection of relevant literature for this survey, including forming search queries and defining the inclusion and exclusion criteria for selection.

To obtain a selection of relevant literature for this survey, Article Information Parser (AIP) [2] and Google Scholar are used. AIP is a tool developed by AtLarge Research that combines publications from DBLP[3], AMiner[4] and Semantic Scholar[5] and provides a user-friendly way to query and filter relevant publications from this large database. The following query was used to obtain relevant publications from AIP:

[2]AIP: https://github.com/atlarge-research/AIP
[3]DBLP: https://dblp.org/
[4]AMiner: https://www.aminer.org/
[5]Semantic Scholar: https://www.semanticscholar.org/

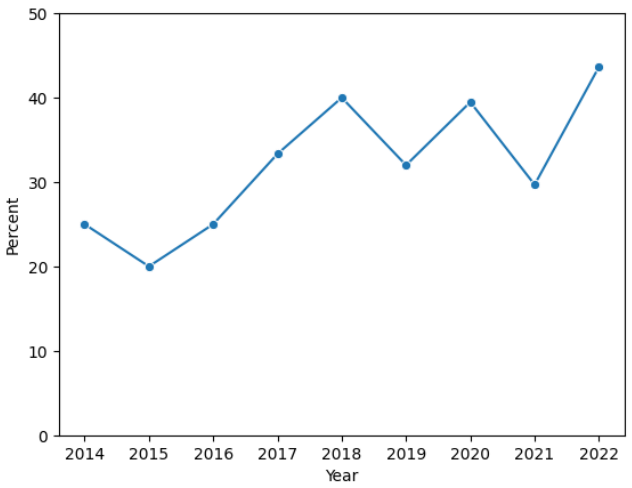

Figure 2: Percentage of papers from 2014-2022 having 'workflow' somewhere in the text among papers that have 'serverless' in the keywords. This is by no means an exhaustive analysis, but is helpful to get a general idea of the amount of papers in serverless that mention workflows.

```
SELECT * FROM publications
WHERE year >= 2014
AND (title ~* 'serverless workflow'
OR abstract ~* 'serverless workflow');
```

However, at the time of writing, the latest update to the AIP database was till February 2023. To fill this gap, Google Scholar was also used to supplement the search by using the following queries:

Q1: Serverless Computing. This gave us publications dealing with all aspects of serverless computing, as expected, and was useful to get publications that allowed us to get a general idea of the areas of research in serverless computing. However, using this query, not many publications dealing with serverless workflows in specific could be obtained. Performing a quick keyword analysis via Constellate[6] revealed that only around 35% of publications that had 'serverless' in their keyword also had the term 'workflow' somewhere in the text, as shown in Figure 2.

Q2: Serverless Workflow. This query narrowed our search space to publications specifically mentioning the term 'serverless workflow'.

Q3: "serverless", "workflow". There might be publications that do not mention the conjugated term 'serverless workflow' but instead just mention 'workflow', however searching for just workflow will result in a broad search space that deviates from serverless. So, this query was formulated to search for publications having the words 'serverless' and 'workflow' separately.

Q4: "serverless", "workflow","survey". Lastly, we wanted to obtain prior literature surveys in serverless workflows as well, so we used this query.

[6] https://constellate.org/builder/

Method	#
Search queries	132
After filtering for relevant material	51
After filtering by analysing content	11
Snowballing	4
Total Selected	**15**

Table 1: Number of papers selected.

After obtaining the queries, we set our inclusion and exclusion criteria to pick out relevant literature pertaining to this survey from the search results. The inclusion and exclusion criteria for selection was set as follows:

I1: *include* papers that propose workflows for specific types of applications. According to the goal of this survey to taxonomize serverless workflows by type of application, as motivated in Section 3.1, an inclusion criteria was set to include papers that propose workflows for specific types of applications.

I2: *include* papers that improve on existing workflows. There may be papers that improve on existing workflows to extend them to different types of applications or refine them to better support specific types of applications. Such papers were included.

I3: *include* only papers that focus on workflows in serverless. Papers that do not focus on serverless workflows and instead focus on other areas of serverless computing, or workflows in non-serverless architectures, are excluded as we are interested in serverless workflows for this study.

E1: *exclude* papers that focus on evaluating performance of existing workflows. Some papers focus on evaluating performance and comparing traditional workflows, we exclude these papers as we are only interested in state-of-the-art workflows.

E2: *exclude* papers that document implementation of applications using existing workflows. Some papers document the process of implementing applications using existing workflows, we exclude these as we are not interested in implementation specific literature.

Using these criteria 15 papers were selected, the details of the selection are shown in Table 1. These papers are discussed in detail in the following section and form the basis for the taxonomy constructed later in Section 5, however research and findings from other papers are also used to complement and contrast the design choices in these 15 papers.

Our search is confined to scientific literature, omitting any commercial applications lacking publicly available methods. Additionally, our search is limited to English literature, potentially overlooking publications not widely recognized within the English-speaking scientific community.

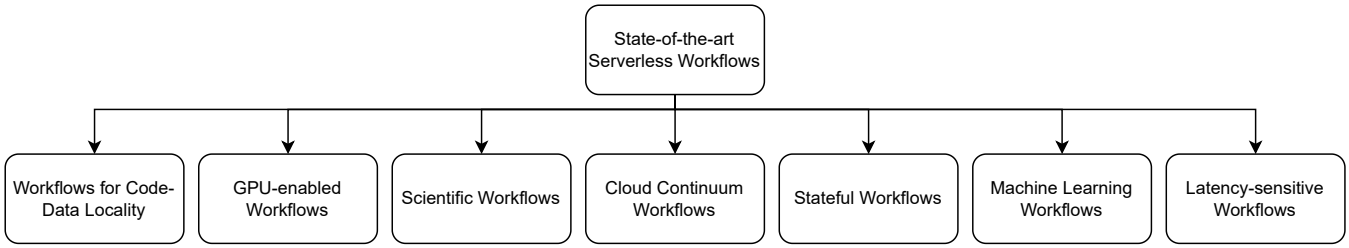

Figure 3: Taxonomy of workflows discussed in this paper.

4 State-of-the-art Serverless Workflows

This section focuses on workflows and their design decisions that improve specific shortcomings of the current serverless platforms, thereby allowing a wider array of applications and domains to use the serverless model. A short overview of all the workflows in this section can be found in Table 2. Figure 3 shows the taxonomy of the workflows that are discussed in this paper.

4.1 Workflows Focusing on Code-Data Locality

Traditionally, in serverless platforms the data required by the functions could be located at a different node than the one where the function executes. Because functions are a black box and there is no way for the serverless platform to know what data is read or written by a function beforehand, this increases latency as well as cost due to the fact that first the data has to be shipped to the code that requires it, and other cloud services (like Amazon S3) are required to mediate the data which is charged per GET/PUT request. Below are outlined some papers that focus on improving the code-data locality of serverless workflows in order to improve performance and reduce the time that is spent waiting on data.

Tang et al. [44] propose 'Lambdata', a serverless computing architecture that focuses on data caching and code-data locality. They argue that for the vast majority of serverless workloads, the input and output data can be determined before run time, so they introduce the concept of 'data intents'. For example, if we have a function that takes in images and compresses them to make thumbnails, then the input can look something like 'pic/1.jpg' and the corresponding output would be 'thumb/1.jpg', where 'pic/' and 'thumb/' are two different buckets in the data store. Data intents is a way for the developers to tell the scheduler what data is read and written by functions, and using this knowledge the scheduler can perform caching of the data on invokers and also co-locate multiple invokers if they operate on the same data. This leads to speed-ups in function execution and reduction in cloud storage calls (as the data is cached locally on the invokers), which both translate into cost reduction. An important aspect of data intents is that they affect performance and not correctness, so it is not critical for a developer to provide data intents for the functions to run correctly. In their evaluation the authors find that there is no statistically significant difference in function invocation times if data is not in the cache, however if the data is cached Lambdata gets an average speed-up of 1.50x. In terms of workflow performance, Lambdata achieves a 2.16x speed-up to finish the workflow compared to OpenWhisk.

The main design decision for Lambdata is the use of *data-aware scheduling*. Traditionally serverless platforms use a time-series based prediction approach to pick warm containers for function invocations, this approach fails to account for data locality. The advantage of data-aware scheduling is that the scheduler is aware of the data needed by the functions and therefore can schedule the functions close to the data as well as cache the data if multiple functions need it. However, the drawback of this is that it increases the overhead for scheduling and offers no benefit if the workload is not heavily data reliant, such as embarrassingly parallel workloads where each task is independent.

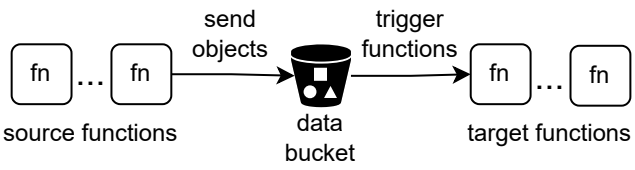

Figure 4: The basic idea behind Pheromone, letting the data drive function invocations.

Yu et al. [50] believe that function orchestration in serverless platforms is not efficient as it connects functions according to their invocation dependency. It specifies the order of function invocations but is oblivious to when and how data are exchanged between them. Without this, the serverless platform assumes that the output of a function is entirely and immediately consumed by the next function, which is not always correct, like processing dynamically accumulated data in stream analytics. They argue that function orchestration should instead follow the flow of data, where the idea is to let data consumption trigger functions and drive the workflow. To this end, they develop Pheromone, a scalable serverless platform with low-latency data-centric function orchestration. Figure 4 shows the basic idea behind pheromone.

Workflow	Area of focus	Implemented on top of	Supported Languages
Lambdata [44]	Code-data locality	OpenWhisk	All languages supported by OpenWhisk
Pheromone [50]	Code-data locality	Cloudburst	C++
GPU sharing [36]	GPU enabled workflows	KNIX (formerly called SAND)	Python, Java
GPU multimedia processing [33]	GPU enabled workflows	AWS Lambda	All languages supported by AWS Lambda
SWEEP [20]	Scientific Workflows	AWS Lambda	All languages supported by AWS Lambda
DayDream [35]	Scientific Workflows	AWS Lambda, but can be ported to any serverless platform	All languages supported the serverless platform it is implemented on
Serverless in the cloud continuum [34]	Cloud Continuum	SCAR and OSCAR, which themselves run on AWS Lambda and OpenFaaS respectively	Any language that supports a command line
OpenWolf [41]	Cloud Continuum	OpenFaaS	All languages supported by OpenFaaS
Beldi [51]	Stateful Workflows	AWS Lambda	All languages supported by AWS Lambda
FAASM [40]	Stateful Workflows	KNative	All languages that can be compiled to WebAssembly
Cirrus [6]	ML Workflows	AWS Lambda	All languages supported by AWS Lambda
FedLess [12]	ML Workflows	AWS Lambda, Azure Functions, Google Cloud Functions, IBM Cloud Functions, OpenWhisk, OpenFaaS	Python
Nightcore [19]	Latency sensitive applications	it is a serverless platform itself, not implemented on top of any existing serverless platforms	C/C++, Go, Node.js, Python
Faastlane [26]	Latency sensitive applications	OpenWhisk	All languages supported by OpenWhisk

Table 2: Workflows discussed in this paper, along with their area of focus, the serverless platform they are implemented on, and the languages supported by them.

Pheromone uses a data-centric approach to function orchestration, where the flow of data triggers function invocations. Traditionally serverless platforms specify the order of function invocations but are oblivious to when and how data are exchanged between functions, which limits the expressiveness of function invocations because it assumes that the data flows the same way as functions are invoked and that the function passes its entire output to subsequent functions, which may not be true for example in a batched stream analytics workload where functions are not immediately triggered as the data arrives but instead have to wait for a set amount of time before invocation.

Also, Pheromone makes use of two-tier distributed scheduling involving a local and a global scheduler. A workflow request first arrives at a global coordinator, which routes the request to a local scheduler on a worker node. The local scheduler invokes subsequent functions to locally execute the workflow whenever possible, thus reducing the invocation latency and incurring no network overhead. Traditionally serverless platforms, like OpenWhisk, use a centralized scheduler, which has a low overhead and results in better resource utilisation as the scheduler has a view of the entire cluster, however it does not account for data locality.

Functions exchange data within a node through a zero-copy shared-memory object store, and they can also pass data to a remote function through direct data transfer, resulting in lower latency compared to cloud storage, which is used traditionally in serverless computing. The benefits of these

design choices are reflected in the experimental results where Pheromone is compared with KNIX (formerly called SAND [2]), Cloudburst, AWS Step Functions and Azure Durable Functions and outperforms them by orders of magnitude in function invocation latency and throughput.

4.2 GPU Enabled Serverless Workflows

With the current offerings of serverless providers like Azure, AWS and Google Cloud, it is not possible to use a GPU for compute. However, compute intensive workloads like training ML models or running physics simulations benefit greatly by running their computations on a GPU instead of a CPU. For such applications, serverless offers no benefit due to the unavailability of GPU compute. The workflows discussed in this section attempt to fix this shortcoming and allow use of GPUs to run serverless functions.

Kim et al. [24] used NVIDIA-Docker, which is a tool that allows building and running docker images using NVIDIA GPUs, to enable GPU compute in IronFunctions which is a serverless service offered by iron.io. Their approach however tied an entire physical GPU to one function instance, which is not very different from the serverful approach, but at a smaller scale. This leads to resource wastage and inhibits parallelism.

Satzke et al. [36] improve on this by proposing a serverless framework allowing users to execute their applications in the form of workflows in the cloud, using heterogeneous and shared CPU and GPU cluster resources. Users can request execution of parts of their applications to be executed on GPUs, and additionally can have fine-grained control of the amount of GPU core and memory to be used for each function as part of the workflow. The authors implement this framework on top of KNIX which uses Kubernetes, and so one of the goals was that it should not be required to modify any Kubernetes code or container images for GPU sharing. An application executed with shared vGPUs should behave as if it was executed on physical GPUs.

To partition the GPU into several vGPUs for use in multiple containers, the GPU manager framework[7] is used. It supports GPU usage isolation on kernel execution, enabling Kubernetes to not only run more than one pod on the same GPU, but also gives service guarantees to each Pod. It allows configuring and limiting both GPU and memory shares for each pod to be deployed on the cluster. However, GPU Manager only works with Nvidia GPUs and Kubernetes. Some alternative approaches for sharing GPUs in containerized applications are KubeShare [49] and GaiaGPU [14].

The authors observe that the GPU-enabled workflow is on average 10x faster when compared to executing the same workloads on the CPU.

Risco et al. [33] extend the SCAR framework [31] to support event driven GPU-enabled serverless workflows for efficient data processing across diverse computing infrastructures.

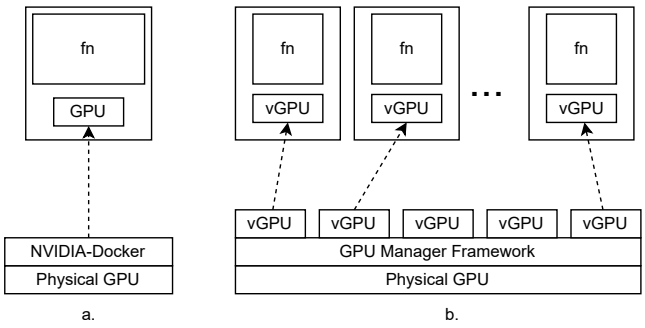

Figure 5: a. shows using of GPU for serverless execution as proposed by [24], b. shows GPU sharing for serverless proposed by [36], a physical GPU is split into multiple vGPUs that can individually be assigned to functions.

By combining the use of both AWS Lambda, for the execution of numerous short jobs, and AWS Batch, for the execution of resource-intensive GPU-enabled applications, an open-source platform has been developed to create scale-to-zero serverless workflows. AWS Batch runs Docker-based computational jobs on EC2 instances, the resource requirements of these jobs can be set by the user in terms of: an increased memory allocation, the assignment of the desired number of CPUs, the instance types to be used and the assignment of GPU devices to containers.

Though AWS Batch is a traditional batch processing system, it can scale-to-zero, terminating cluster nodes while keeping managed job queues at no extra cost. When paired with AWS Lambda's event-driven approach through SCAR, it enables automatic job submissions when new files are uploaded. Integrating AWS Batch in SCAR was done by redesigning the faas-supervisor [8], which is an open-source library to manage the execution of user scripts and containers in AWS Lambda and also in charge of managing the input and output of data on the Amazon S3 storage back-end. The redesign enables job delegation to AWS Batch using the FaaS Supervisor in AWS Lambda. This platform has three execution modes:

- *lambda*: User-defined container images run on AWS Lambda using udocker, which pulls images from a registry and executes them in Lambda's runtime.
- *batch*: In this mode, AWS Lambda serves as an event gateway, translating function invocations into AWS Batch jobs. The event details are sent as an environment variable to the job, enabling the FaaS Supervisor on the EC2 instance to handle data staging.
- *lambda-batch*: Functions initially run on AWS Lambda, but if they fail or approach a defined timeout, AWS Batch takes over. This enables Lambda to handle sudden bursts of short jobs and ensures more resource-intensive tasks are processed when Lambda limits are surpassed.

[7] GPU Manager: https://github.com/tkestack/gpu-manager

[8] faas-supervisor:https://github.com/grycap/faas-supervisor

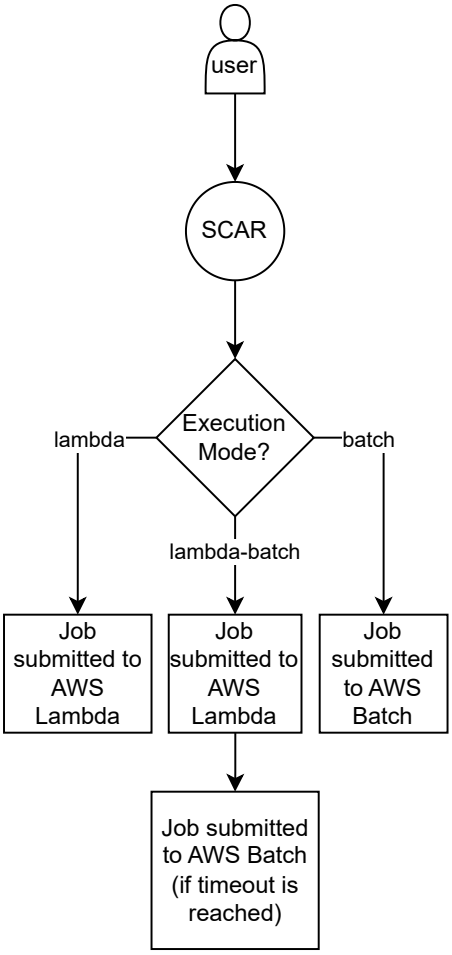

Figure 6: Workflow to integrate GPUs via AWS Batch, using a modified SCAR framework, as described in [33].

In this workflow, the GPU workloads are executed on AWS Batch. This approach is not truly serverless, as the GPU workloads are handled by AWS Batch, which is not a serverless service. The average cost of execution on AWS Batch is approximately 300x higher than AWS Lambda, this is due to the fact that unlike lambda, batch does not support per-millisecond billing and is instead billed per second. This significantly increases the cost of execution and is only beneficial when the execution time on lambdas is significantly higher than on batch. In the authors' evaluation, executing jobs on AWS Batch was cheaper than lambdas only in cases where lambdas took 50x more time compared to batch. Apart from cost, parallelism also becomes an issue with batch since it runs on EC2 instances and starting up multiple instances takes time because a new VM is provisioned for each instance, instead of containers as in lambda.

4.3 Workflows for Scientific Workloads

Scientists in diverse fields, like high-energy physics and astronomy, are creating complex workflows with numerous interconnected tasks, exemplified by projects like Montage [18], LIGO [1], and CyberShake [13]. These workflows involve a substantial number of jobs, extensive input and output data, and intricate task dependencies. To manage these workflows, scientists typically employ workflow management systems (WMS) necessitating the setup and configuration of clusters as the execution environment.

However, configuring large-scale clusters can be daunting, especially for researchers unfamiliar with high-performance computing (HPC), leading to significant resource underutilization due to complex task dependencies. The serverless paradigm provides a good solution to this problem as the job of infrastructure management is left to the provider. Below we look at some serverless workflows that are designed to handle scientific workloads.

John et al. [20] propose the Serverless Workflow Enablement and Execution Platform (SWEEP). SWEEP workflows are represented abstractly using a Directed Acyclic Graph (DAG) formalism. Here, tasks are nodes, and an edge from node X to node Y indicates that X must finish before Y can start. The main contribution of SWEEP is implementation of complex flow control such as nested scatter and multi-level gather. It offers several workflow constructs for static and dynamic control of execution. Dynamic parallelism is implemented using the task properties *scatter* and *follow*, which are described below.

The *scatter* property multiplies task Y based on a list-type output from predecessor task X. Each new copy of Y (Y_i) receives one item from the list. Descendant tasks Z of scattered Y are controlled by the *follow* property. If Z follows ancestor node U, it multiplies by the same value as U, and there's a path from U_i to Z_i for all related nodes. Essentially, Z_i aggregates output from U_i. If no follow value is specified, one copy of Z is created, and there is an edge to Z from each of its original workflow predecessors.

By combining these, custom multi-level scatter and gather behaviour can be defined, some of which are illustrated in Figure 7. Error handling behaviour can be controlled in a task-specific fashion by limiting the number of retries, and specifying whether a task failure should be ignored or should halt workflow execution.

Roy et al. [35] propose DayDream to execute scientific HPC workflows on serverless platforms with the aim of minimising execution time and cost. According to them, HPC workflows are highly dynamic and therefore often lead to overprovisioning or underprovisioning of resources when deployed traditionally using servers. Due to this, the serverless model of deployment is well suited for them.

However, there is a challenge using serverless for HPC workloads currently, that is, unlike HPC and VM clusters,

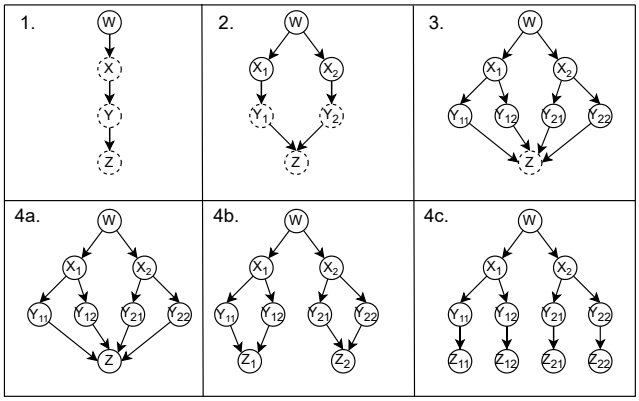

Figure 7: Multi-level scatter and gather behaviour implemented in SWEEP. Step 1 shows the original workflow definition, with dotted lines indicating dynamic tasks that have scatter and/or follow properties yet to be resolved. In Step 2, task X scatters based on output from task W. Task Y follows X. In Step 3, the tasks Yi scatter on output from Xi. Three versions of Step 4 are shown, depending on the follow-property of task Z. In 4a, task Z does not have the property specified. In 4b, task Z follows task X, with each Z_i gathering the output from an X_i. In 4c, task Z follows task Y, forming parallel chains of tasks.

serverless resources are not pre-allocated, resulting in startup delays or cold start times. These delays, often comprising 25% to 60% of execution time, have the potential to prolong workflow execution, diminishing the advantages of serverless computing. Typically, cold starts in serverless are minimized by predicting function invocations using a time-series based prediction and pre-warming containers, their experiments show that HPC DAGs, primarily dynamic and irregular in nature, exhibit varying execution paths for different inputs and operations. They lack a predictable time-series pattern, making it challenging to predict and warm up the invoked components. So, applying state-of-the-art serverless techniques yields suboptimal solutions for complex real-world scientific workflows.

DayDream utilizes a hot start mechanism, separating the runtime environment from the component (function) code. This decoupling makes a hot started function instance suitable for the execution of any component, unlike warm started instances and enables DayDream to predict the total component count rather than individual types, significantly reducing cold start overhead and minimizing keep-alive costs.

DayDream also predicts the number of instances to hot start. To calculate how many serverless instances to hot-start DayDream predicts the phase concurrency[9]. The authors noted that when examining the phase concurrency frequency histogram, a discernible trend emerges – the samples can be modelled by a statistical distribution, Weibull distribution is used by the authors in this case. During the initial run of an HPC DAG, DayDream identifies and establishes Weibull distribution parameters for phase concurrency frequency. In subsequent runs, it generates sample numbers according to this distribution for each phase. HPC workflows are typically executed multiple times with varying inputs and operations. These generated samples guide DayDream in determining how many function instances to hot start for each phase.

DayDream reduces the overall service time by more than 45% over the traditional scientific workflow manager Pegasus.

4.4 Workflows for the Cloud Continuum

Cloud Continuum is the seamless integration of various types of cloud capabilities and services, including data centre, private cloud, public cloud, hybrid cloud, and multi clouds. It consists of cloud, edge and fog tiers. This section discusses some serverless workflows for applications that span across the cloud continuum.

Risco et al. [34] want to use public clouds with their vast resource (AWS etc.), on-premise clouds with their greater privacy and federated data providers all in a single serverless workflow. It is useful when sensitive data (like medical data) cannot be on the public cloud, but processing related to the data needs the 'unlimited' resources of a large public cloud because it would be too slow otherwise. They use SCAR [31] and OSCAR [32] frameworks along with minIO [10] to create a serverless workflow where minIO is run on premise, and can communicate with the AWS S3 API to use AWS Lambdas (and other AWS services) to trigger serverless functions and Onedata [9], a global data management system that provides access to distributed storage resources for data-intensive scientific computations, issues revocable access tokens for space access, enhancing security. The on-premise cloud consists of open source platforms like Kubernetes and OpenFaaS along with a custom FDL for defining the workflow so that only one trigger can invoke the chain of functions from Kubernetes to AWS, as illustrated in Figure 8.

Sicari et al. [41] introduce OpenWolf, a versatile Serverless Workflow Management System designed to harness the FaaS paradigm for orchestrating intricate scientific workflows across the Cloud-Edge Continuum. Their approach involved federating a Continuum environment through a Kubernetes Cluster using K3S[11]. Within this environment, OpenFaaS was deployed to create architecture-agnostic functions. Additionally, they crafted a Workflow Agent, a compact microservice capable of parsing Workflow Manifest files and efficiently

[9] A *phase* refers to multiple components which can run in parallel, without any data or state dependency between them. In a phase, a component can have multiple running instances, the sum of which is *component concurrency*. The sum of all components concurrences in a phase constitutes the *phase concurrency*.

[10] MinIO: High Performance, Kubernetes Native Object Storage: https://min.io/

[11] K3S, a Kubernetes distribution for IoT & edge computing: https://K3S.io/

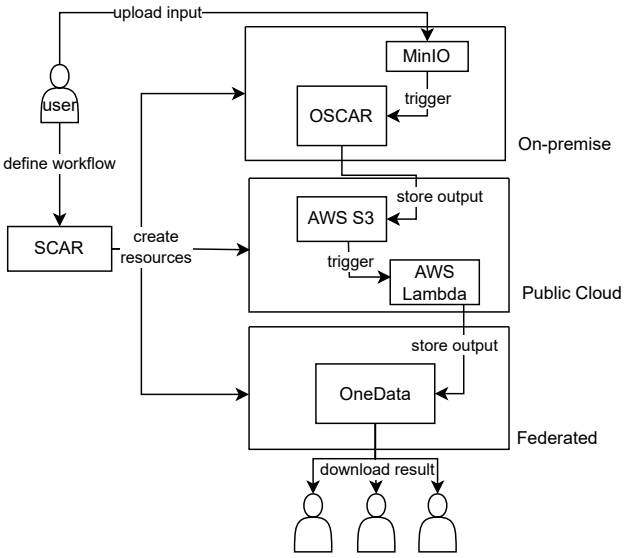

Figure 8: A workflow involving on-premise cloud, public cloud and a federated data storage, as proposed by Risco et al. [34]. SCAR manages resources across public and private clouds, OSCAR is used to deploy serverless functions on-premise using kubernetes.

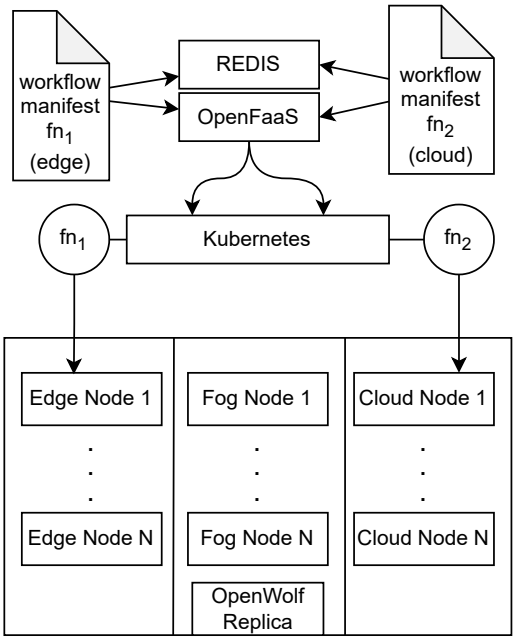

Figure 9: Architecture of OpenWolf.

routing function data in alignment with the workflow's structure. According to the authors, the shortcomings of open source serverless platforms that prevented them from being used in scientific workflows in the cloud-edge continuum was that:

- they do not natively support workflows with single/multiple trigger(s), many-to-many relationships between functions, data pre- and post-processing filters and parallel and sequential process execution, allowing only sequential composition of functions without data pre- and post-processing.
- and, serverless does not guarantee architecture transparency. It is not possible to transparently deploy functions over x64 platforms in Cloud, while ARM architectures are used in Edge.

It was required to establish a unified computing cluster that encompasses all Continuum nodes and manages them through a consistent interface facilitates the orchestration, composition, and distribution of functions across the Continuum.

For this purpose, K3S was chosen, mainly because of its ability to run in resource constrained environments while maintaining all Kubernetes features as well as support for ARM and x64 architectures. An alternative to K3S is microk8s, both of which are compared and benchmarked for serverless workloads in [25] revealing that there are no significant benefits in one over the other, however microk8s does not support ARM32 architectures which K3S does. OpenFaas was deployed over the K3S nodes to build, deploy and trigger functions. The cluster architecture includes a Redis instance dedicated to storing both the workflow manifests and execution information. Within the K3S cluster, the OpenWolf Agent serves as a central coordinator, facilitating communication and coordination among the components and the functions within the workflows.

The OpenWolf agent is a stateless microservice deployed in the same Kubernetes Cluster as the serverless functions. The architecture of OpenWolf is shown in Figure 9.

4.5 Stateful Serverless Workflows

The serverless model was initially designed for execution of stateless functions, which means that each function is ephemeral and does not persist data or share data with other functions in the workflow. However, most modern applications require exchange of data between different parts of an application. The serverless model was exploited to support these stateful applications by using databases to manage state but at the cost of performance [16, 37]. Below, we take a look at some serverless workflow technologies that were designed with state management as a priority.

Zhang et al. [51] believe that a key challenge in increasing the general applicability of serverless computing lies in correctly and efficiently composing different functions to obtain non-trivial end-to-end applications. This is fairly straightforward when functions are stateless, but becomes involved when the functions maintain their own state (e.g., modify a data structure that persists across invocations). Composing such stateful serverless functions (SSFs) requires reasoning about consistency and isolation semantics in the presence of concur-

rent requests and dealing with component failures. To this end, they present Beldi, a library and runtime system for building workflows of SSFs. Beldi runs on existing cloud providers without any modification to their infrastructure.

Beldi's goal is to guarantee exactly-once semantics to workflows in the presence of SSFs that fail at any point in their execution, and to offer synchronization primitives (in the form of locks and transactions) to prevent concurrent clients from unsafely handling state. To do so, Beldi introduces novel refinements to an existing log-based approach to fault tolerance, including new data structure and algorithms that operate on this data structure, support for invocations of other SSFs with a novel callback mechanism, and a collaborative distributed transaction protocol. With these refinements, Beldi extracts the fault tolerance already available in NoSQL databases, and extends it to workflows of SSFs at low cost with minimal effort from application developers.

To make SSFs fault-tolerant and provide concurrency control, Beldi relies on strongly consistent databases (for example, DynamoDB) to manage state. The benefit of this is that the application developers do not have to worry about concurrency control, fault tolerance, or manually making all of their functions idempotent. The tradeoff for this is that if these databases were to become unavailable, for example due to network partitions, SSFs that write to these unavailable databases would also become unavailable until the partition was resolved. An alternative to using strongly consistent databases is to use ACID databases, like Amazon Aurora, however, ACID databases are not enough to guarantee exactly-once semantics for function invocations since they provide atomicity for read and write operations, but have no support for invocations.

Shillaker et al. [40] argue that current serverless platforms use stateless containers to isolate functions, which hinders memory sharing. This necessitates redundant data duplication and serialization, resulting in performance and resource expenses. They advocate for a lightweight isolation method enabling direct memory sharing between functions, ultimately cutting down on resource overhead while sharing state across functions. To achieve this, they present FAASM, a serverless runtime that leverages the LLVM compiler toolchain to translate applications into WebAssembly. It supports functions in various languages like C/C++, Python, TypeScript, and JavaScript and can seamlessly integrate with current serverless platforms. FAASM also introduces Faaslets, a novel high-performance serverless computing concept, employing software-fault isolation (SFI) through WebAssembly to isolate function memory. Faaslets enable memory sharing in the same address space, removing the need for costly data transfers when functions run on the same machine. They employ a two-tier state architecture: a local tier for in-memory sharing and a global tier for distributed state access across hosts.

In Figure 10, Faaslets A and B access a shared region (S) from a distinct part of the common process memory (central region). Each Faaslet has a private memory region (A and B) allocated from the process memory. Functions within each Faaslet access all memory as offsets from zero, creating a unified linear address space. Faaslets map these offsets to either a private region (lower offsets) or a shared region (higher offsets). Faaslets outperform containers and VMs with a memory footprint under 200 KB and cold-start times under 10 ms. FAASM is the serverless runtime that uses Faaslets to execute distributed stateful serverless applications across a cluster. FAASM is designed with a distributed architecture where multiple runtime instances run on servers, each handling a distinct pool of Faaslets.

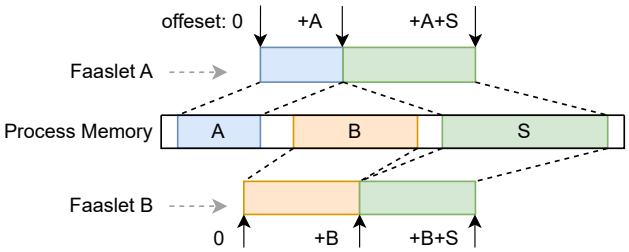

Figure 10: Faaslet shared memory region mapping.

FAASM uses faaslets, which builds on SFI, for isolation of functions while having benefits of memory safety, resource isolation, efficient state sharing and a shared file system between functions. Some alternative methods for function isolation include containers, VMs and unikernels. While containers are traditionally the most widely used method, they do not allow for efficient state sharing and are resource intensive, with a memory footprint of hundreds of megabytes. SFI has been used in several existing serverless systems like Terrarium[12] and Cloudfare Workers[13], however they don't isolate CPU or network use and rely on data shipping for accessing state. Faaslets extend SFI to enable resource isolation by using Linux cgroups and virtual network interfaces, and in-memory state sharing by adding shared memory regions to the existing WebAssembly memory model.

The main downside of faaslets is that it can only work with languages that have WebAssembly support.

4.6 Workflows for Machine Learning

Machine learning workflows are intricate, with diverse stages like preprocessing, training, and tuning, each demanding varying computational resources. This complexity often leads to over-provisioning of resources. While serverless computing offers a solution to resource management, applying it to existing ML frameworks faces challenges due to resource limitations imposed by serverless platforms, coupled with the fact

[12]Fastly Terrarium: https://wasm.fastlylabs.com/
[13]Cloudfare Workers: https://workers.cloudflare.com/

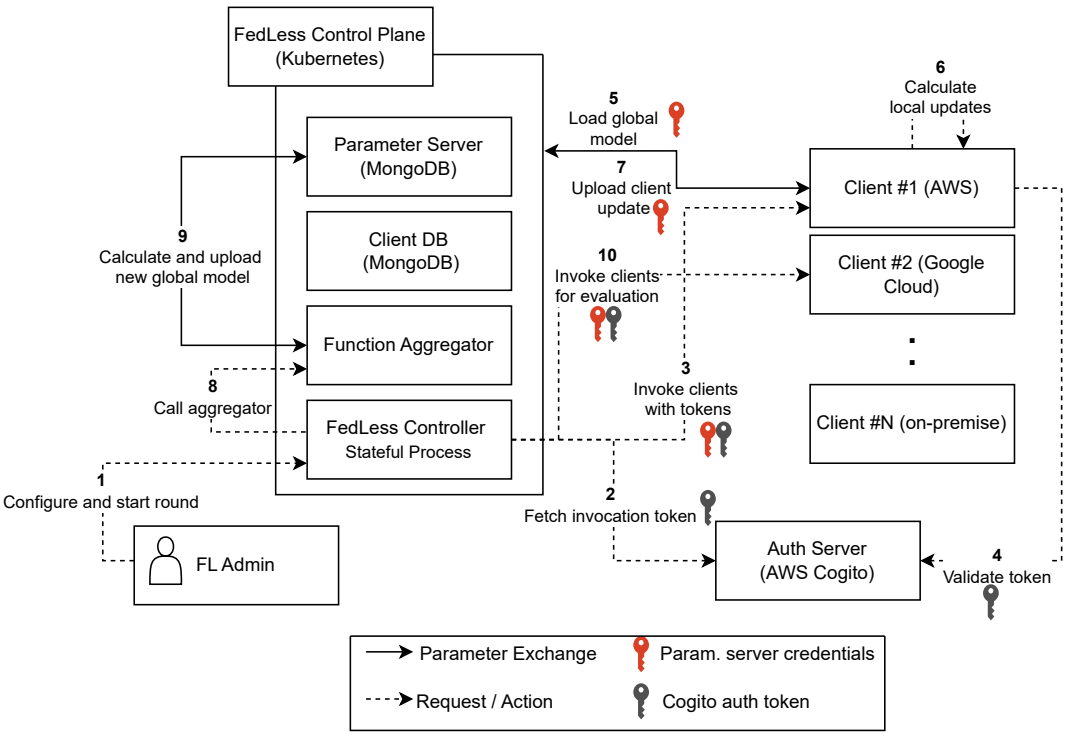

Figure 11: Workflow in Fedless.

that existing ML frameworks commonly assume abundant resources, such as memory. For instance, the ML framework Spark generally loads all training data into memory, which is not possible in serverless instances.

Training a neural network in AWS Lambda has been evaluated previously [11], revealing that there is no benefit to using serverless for training neural networks compared to the non-serverless approach. Some workflows focusing on adapting serverless computing for ML workloads have been proposed and are detailed in this section.

Carreira et al. [6] present Cirrus, a comprehensive framework designed specifically for machine learning training within serverless cloud environments, such as AWS Lambdas. It offers user-friendly tools to facilitate various aspects of machine learning workflows, including dataset preprocessing, training, and hyperparameter optimization. It has four design principles, namely, adaptive and fine-grained resource allocation to avoid over-provisioning resources, a stateless server-side backend for robust and efficient management of serverless compute resources, end-to-end serverless API for model training, dataset preprocessing, feature engineering, and parameter tuning, and high scalability.

Cirrus employs three key components to achieve these goals. First, it offers a Python frontend for ML developers, enabling an API for ML training and efficient management of large-scale computations in a serverless environment. Secondly, to address low-latency serverless storage limitations, Cirrus provides a distributed data store for shared intermediate data. Finally, it utilizes a serverless lambda-based worker runtime with efficient access to training datasets in S3 and intermediate data in the distributed data store. Since ML models can have hundreds of updates per second, it is required for the data store to be low latency and have a high throughput.

Cirrus makes use of a multithreaded server deployed on cloud VMs that distributes the workload among many cores, leading to about 30% increase in throughput, but eventually performance of the data store becomes bottlenecked by the network. To solve this, all the gradients transferred to and from the data store are compressed, and the data store only sends and receives sparse gradients and data structures. This allows the data store to achieve latencies of 300 μs compared to 10ms for AWS S3.

Cirrus's Sparse Logistic Regression was compared with two specialized VM-based ML training frameworks: TensorFlow and Bosen. Cirrus was at least 5x faster than Bosen and at least 3x faster than Tensorflow, while achieving lower loss both times. Cirrus was also compared to PyWren, which is a serverless framework providing *map* and *reduce* primitives, by implementing a SGD training algorithm for Logistic Regression. Cirrus achieved 100x more model updates per second compared to PyWren.

Grafberger et al. [12] presented a novel system and framework for serverless Federated Learning (FL) called FedLess. Federated learning is a machine learning approach that en-

ables model training across decentralized and distributed devices or servers while keeping data localized and private. In traditional machine learning, data is typically collected and centralized in a single location for training, which can raise privacy and security concerns.

Federated learning addresses these issues by allowing model training to occur locally on individual devices or servers, and only model updates are shared and aggregated. FedLess enables FL on the serverless platforms of AWS, Azure, Google Cloud and IBM Cloud as well as open source platforms OpenWhisk and OpenFaaS while providing important features such as authentication, authorization, and differential privacy.

As FL clients come from different institutions and networks and need public internet access to reach the FL server, ensuring that only authenticated and authorized entities can invoke client functions is of utmost importance.

A workflow for training multiple clients in FedLess is shown in Figure 11. First, the FL admin, that is the person who manages function deployments and holds the data, selects the model, client functions, and hyperparameters. The training begins as the FedLess controller obtains an invocation token from the Auth Server and instructs randomly chosen clients to start. Clients validate the token, load the global model, train locally, and upload their parameters. The controller waits for client completion, initiates model aggregation, and evaluates the global model, either from previous aggregation or by selecting new clients for testing. The process resumes from step 2 if the accuracy threshold isn't met.

FedLess costs on average about 50% less to reach the same accuracy as Flower while taking on average 1.7x longer.

4.7 Workflows for Latency Sensitive Applications

The average invocation latency of AWS functions is around 14ms for a warm invocation and around 100ms for a cold start, for latency sensitive applications like a user interactive application, it leads to a bad user experience or an outright unusable experience. It is amplified by the fact that in a serverless application one interaction can lead to invocation of many functions, which worsens the issue. The workflow technologies described in this section attempt to mitigate this issue.
Jia et al. [19] aim to reduce latency in serverless computing so that it could be used for applications that require microsecond latency. Although many papers focused on reducing latency by getting rid of the heavyweight docker containers and using other methods of isolation, the authors believe that the level of isolation that containers provide is essential. Instead, they reduce latency by focusing on other aspects like inter and intra process communication, optimizing the scheduling of functions by taking into account the number of invocations and the run time per function, efficient threading for I/O operations.

To this end, they present Nightcore, a serverless function runtime with microsecond-scale overheads that provides container-based isolation between functions. Nightcore follows the conventional FaaS system structure with a frontend and backend separation. The frontend serves external function requests and manages tasks like function registration through an API gateway. The backend is composed of independent worker servers, offering scalability and fault tolerance. Each worker server runs a Nightcore engine process and function containers, where each container hosts a registered serverless function, ensuring one container per function on each worker server. Nightcore's engine directly handles function containers and communicates with worker threads within them. Nightcore's architecture is illustrated in 12.

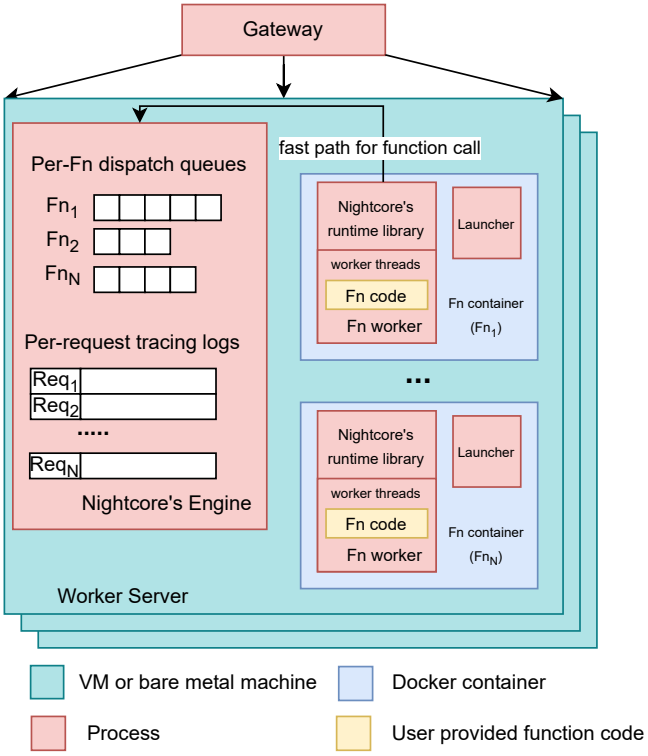

Figure 12: Architecture of Nightcore.

Nightcore optimizes local function calls within the same worker server, enhancing performance without relying on the API gateway, traditionally in serverless all function calls are routed through the gateway even if the function will be run on the same worker. If necessary, Nightcore can route these calls through the gateway when running on different worker servers, and different workflows remain logically independent and are executed on separate worker servers. However, for this Nightcore needs to know which set of functions form a single application. In practice, this knowledge comes directly from the developer, e.g., Azure Functions allow developers to organize related functions as a single *function app*. This requires extra information from the developer, the absence

of which will affect performance. Also, Nightcore's performance optimization for internal function calls assumes that an individual worker server is capable of running most function containers from a single application, which may be difficult to guarantee in a multi-tenant environment.

Nightcore offers container-level isolation between functions but doesn't ensure isolation between repeated invocations of the same function because different invocations of the same functions are run within the same process. This compromise suits microservices, given the challenge of achieving rapid isolated execution environments. When using remote procedure call (RPC) servers for microservices, Nightcore's isolation matches containerized RPC servers while also providing similar guarantees as the RPC servers.

Nightcore employs low-latency message channels for swift communication, using 1KB messages. Each message begins with a 64-byte header, including type and metadata, followed by a 960-byte payload.

Lastly, **Nightcore computes concurrency hints (τ_k) to estimate the number of instances it needs to warm up.** Nightcore's engine regulates concurrent function executions using *Little's law* which states that the ideal concurrency can be estimated as the product of the average request rate and the average processing time. The engine keeps two exponential moving averages for each function, denoted as λ_k (request rate) and t_k (processing time). Request rates are calculated as 1 divided by the interval in consecutive function invocations (Fn_k). Processing times are determined as the time between dispatch and completion timestamps, excluding queueing delays. When receiving an invocation request of Fn_k, the engine will only dispatch the request if there are fewer than τ_k concurrent executions of Fn_k . Otherwise, the request will be queued, waiting for other function executions to finish.

Nightcore achieves around 1.5x to 3x higher throughput and up to 69% reduction in tail latencies compared to a RPC server, while OpenFaaS consistently underperforms an RPC server in the authors' evaluation.

Kotni et al. [26] propose Faastlane, a serverless orchestration service developed atop Apache OpenWhisk, which minimizes function interaction latency by striving to execute functions of a workflow as threads within a single process of a container instance, which eases data sharing via simple load-/store instructions. In most serverless applications, functions work together and need to exchange data between them often, which is hindered by the fact that the functions run in separate and isolated containers. Although prior work has explored the idea of getting rid of containers and using lightweight alternatives, like a thread, process or WebAssembly runtime, as a unit of isolation [2, 3, 40], Faastlane cleverly combines execution of function containers, processes or threads depending on the requirement.

Faastlane uses a combination of containers, processes and threads for function isolation. Although using threads for isolation has benefits of low latency communication via a shared address space and low start-up overhead, interpreted languages like Python use a global interpreter lock that prevents concurrent execution of application threads. This limits the applicability of threads as isolation. Running functions as processes also presents the limitation that a single container may not have enough vCPUs to run all functions concurrently as processes in case of a massively parallel workload. Faastlane solves this by using a static workflow composer that analyses the workflow to determine if parts of the workflow should be executed in containers, threads or processes by taking into account the amount of parallelism in the workflow. Functions executed as threads share an address space and so can access data in that space without restriction, this is often undesirable for security reasons and it is required often required that sensitive data be isolated.

Faastlane uses Intel MPK to provide thread-granularity memory isolation for functions that share a virtual address space. While other techniques exist for in-process memory isolation, such as [5,7], prior work has shown that Intel MPK has a comparatively low overhead [15, 45]. The downside of using MPK is that it is only available on certain Intel CPUs.

In their evaluation, the authors find that Faastlane accelerates workflow instances by up to 15×, and reduces function interaction latency by up to 99.95% compared to OpenWhisk.

5 Taxonomy of state-of-the-art serverless workflows

This section answers **RQ1:** *Which types of applications benefit from the state- of-the-art serverless workflows?* After surveying the literature in the previous section, we now construct a taxonomy of serverless workflows, grouping them by the type of application that they support. All the serverless workflows in the literature targeted specific types or domains of applications, aiming to either allow them to be deployed serverless or to improve their performance when deployed serverless. Considering this, we decided to taxonomize the workflows by application type, as shown in Figure 3. We group the workflows into seven categories, described below.

Workflows for code-data locality. Workflows that focus on co-locating that function code and data come under this category. Applications that work with large amounts of data will benefit from these workflows as they won't have to wait for data to be shipped to the node executing the function, some examples of such applications are graph processing applications, distributed databases and data processing pipelines. The main design choices for these workflows are in the area of scheduling and storage.

GPU enabled workflows. Workflows that allow using GPU for computations in a serverless environment fall into this category. Applications that can benefit from these workflows are compute intensive applications, for example image

and video processing. The main design choices for these workflows are in the area of scheduling and GPU virtualisation.

Scientific Workflows. Workflows that enable execution of scientific applications fall into this category. Applications with intricate task dependencies requiring complex function invocation patterns can benefit from these workflows. The main design choices for these workflows deal with cold starts and function interaction patterns.

Cloud continuum workflows. Workflows that allow execution across the cloud continuum including public and private clouds as well as cloud and edge devices come under this category. Applications that need to have their computations or data spread across public and private clouds or cloud and edge devices can benefit from these workflows. The main design choices of these workflows are in the area of edge computing.

Stateful workflows. Workflows that enable efficient state management fall into this category. Applications needing low latency state management along with features like state consistency and fault tolerance can benefit from these workflows. The main design decisions for these workflows are dealing with function isolation and communication.

Machine learning workflows. Workflows that enable machine learning workloads like training, inference and hyperparameter tuning are under this category. ML applications benefit from these workflows. The main design choices affecting this type of workflow are in the area of communication.

Latency sensitive workflows. Workflows that aim to reduce end-to-end latency in serverless execution come under this category. Applications that need strict latency targets, like user-facing microservices, can benefit from these workflows. The main design choices for this type of workflow fall in the areas of function isolation, cold starts and communication.

We note that there is some overlap in the above-mentioned categories, for example, workflows for code-data locality also have the effect of reducing latency and therefore can also be considered in latency sensitive workflows. Several categories also deal with joint challenges, latency sensitive workflows and stateful workflows both deal with challenges in function isolation and communication between functions, latency sensitive workflows and scientific workflows both have to deal with cold starts, while GPU enabled workflows can also benefit applications using machine learning workflows as ML workloads benefit greatly from using GPUs.

6 Current Research Areas in Serverless Workflows

This section answers **RQ2:** *What are the current areas of research for different domains in serverless workflows?*

To get an idea of current research areas in serverless workflows, keyword analysis was performed. To obtain the data for keyword analysis, the platforms Scopus [14] and Web of Science [15] were considered, Scopus was chosen because we found it included more publications as well as provided a wider range of metrics to evaluate research impact. First, the Scopus database was queried for publications having 'serverless' and 'workflow' as keywords. This gave us an idea of the different domains that are interested in serverless workflows, as shown in Figure 14. Next, to get an understanding of which areas in serverless are of interest in these different domains, a keyword co-occurrence analysis was carried out where we check which other keywords appear alongside 'serverless' and 'workflow' in these publications. The top 6 domains, apart from computer science, and keywords in that domain that appear alongside 'serverless' and 'workflow' are discussed next. The top 6 domains were chosen as a cut-off here because after the top 5, all the other domains have less than 10 publications between 2014-2023.

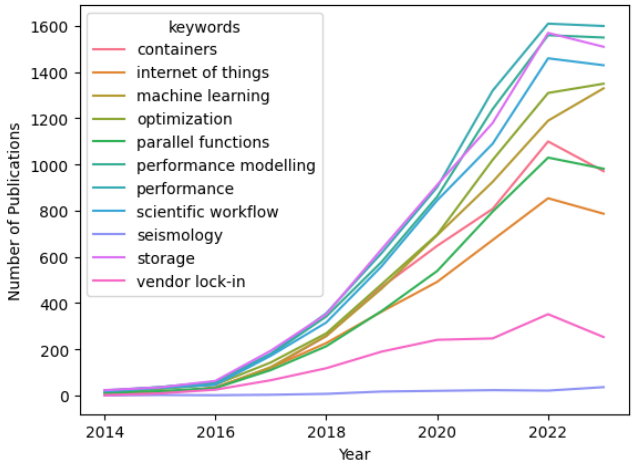

Figure 13: Number of publications having 'serverless', 'workflow' and the specified words somewhere in the text. Data collected from Google Scholar.

Computer Science. The top keywords that appear alongside 'serverless' and 'workflow' in publications in the domain of computer science are: *performance, storage, containers, scientific workflows*. This suggests that the most publications related to serverless workflows in the domain of computer science deal with improving performance, the keywords *storage* and *containers* being second the third suggest that research is also focused on finding methods of exchanging data and state efficiently as well as reducing latency in workflows.

Engineering. The top 5 keywords that appear alongside 'serverless' and 'workflow' in publications in the domain of engineering are: *performance, storage, containers, internet of things*. The keyword co-occurrence in the domain of engineering has a lot of overlap with computer science, suggesting similar research interests, the keyword 'internet of things' suggests interest in using serverless with IoT and edge devices.

[14] Scopus: https://www.scopus.com/home.uri

[15] Web of Science: https://www.webofscience.com/wos/

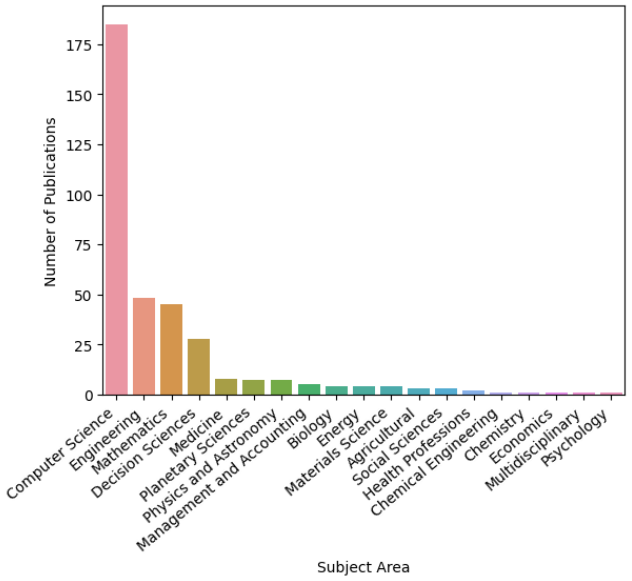

Figure 14: Number of publications having the keywords 'serverless' and 'workflow' between 2014-2023 grouped by subject area, from the Scopus dataset.

Mathematics. The top 5 keywords that appear alongside 'serverless' and 'workflow' in publications in the domain of mathematics are: *Workflow Scheduling, Scientific Workflows, Performance Modelling, Performance, Optimization*. From these keywords we can see that the research is mostly concentrated in improving, optimising and evaluating performance.

Decision Sciences. The top 5 keywords that appear alongside 'serverless' and 'workflow' in publications in the domain of decision sciences are: *Storage, Containers, Machine Learning, Scientific Workflows*. We can gather from this that for decision sciences, serverless research focuses mainly on storage and ML workflows.

Medicine. The top 5 keywords that appear alongside 'serverless' and 'workflow' in publications in the domain of medicine are: *AWS Lambda, Wireless Communication, Vendor Lock-in, Time Factors*. This suggests that for the medical domain, AWS Lambda might be widely used as well as serverless research focusing on wireless communication, vendor lock-in and time/latency.

Planetary Science. The top 5 keywords that appear alongside 'serverless' and 'workflow' in publications in the domain of planetary sciences are: *Software Testing, Web Services, Seismology, Seismic Imaging, Event-driven*, suggesting that for the field of planetary science the applications concerned with seismology are of interest in regard to serverless.

Physics and Astronomy. The top 5 keywords that appear alongside 'serverless' and 'workflow' in publications in the domain of physics and astronomy are: *Cold-start, Parallel Functions, Function Fusion, Containers*. These suggest that in the domain of physics and astronomy, research in serverless is in the area of improving latency (cold starts), parallelism of functions and function interaction patterns.

Overall, we see a good amount of overlap in co-occurrence of keywords across domains. The keywords *performance, storage, containers and scientific workflow* are common for most of the domains, suggesting a strong interest of research related to serverless workflows for these areas.

7 Areas for Future Research in Serverless Workflows

This section answers **RQ3:** *What are the potential areas for future research in serverless workflows?*. To answer this, trend analysis was performed using keywords obtained in Section 6. To perform the trend analysis, the total number of publications per year having the required keywords were obtained from Google Scholar using [43]. Figure 13 depicts the trend of co-occurrence of the specified keywords alongside keywords *serverless* and *workflow* from 2014 to 2023. We see an increasing interest in all keywords over the years, however *seismology, vendor lock-in* and *storage* are comparatively low in volume while being of interest in their specific domains, as detailed in Section 6. These areas can be useful for future research in serverless workflows.

8 Conclusion

In this literature survey, we documented the available literature for serverless workflows and taxonomized them by the type of application. These workflows extend the range of applications that can be deployed serverless by extending the functionality offered by existing serverless platforms.

We also look at the current research areas in serverless workflows by performing a keyword co-occurrence analysis from the Scopus database where we find that besides computer science other domains like engineering, mathematics, medicine and physics are also interested in serverless workflows, each focusing on different areas of research in serverless like performance, storage, scientific workflows, latency, vendor lock-in, among others. We find that some keywords are unique to certain domains, for example the publications in the field of medicine dealing with serverless workflows focus more on latency and vendor lock-in, while most keywords, like performance, storage and containers are common across most domains.

Lastly, we find areas for future research in serverless workflows by performing a trend analysis of keywords co-occurring with serverless and workflow, revealing that storage, vendor lock-in and seismology applications are of interest for future research.

References

[1] Abramovici, A., Althouse, W. E., Drever, R. W., Gürsel,

Y., KAWAMURA, S., RAAB, F. J., SHOEMAKER, D., SIEVERS, L., SPERO, R. E., THORNE, K. S., ET AL. Ligo: The laser interferometer gravitational-wave observatory. *science 256*, 5055 (1992), 325–333.

[2] AKKUS, I. E., CHEN, R., RIMAC, I., STEIN, M., SATZKE, K., BECK, A., ADITYA, P., AND HILT, V. {SAND}: Towards {High-Performance} serverless computing. In *2018 Usenix Annual Technical Conference (USENIX ATC 18)* (2018), pp. 923–935.

[3] AL-ALI, Z., GOODARZY, S., HUNTER, E., HA, S., HAN, R., KELLER, E., AND ROZNER, E. Making serverless computing more serverless. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (2018), IEEE, pp. 456–459.

[4] BALDINI, I., CASTRO, P., CHANG, K., CHENG, P., FINK, S., ISHAKIAN, V., MITCHELL, N., MUTHUSAMY, V., RABBAH, R., SLOMINSKI, A., ET AL. Serverless computing: Current trends and open problems. *Research advances in cloud computing* (2017), 1–20.

[5] BITTAU, A., MARCHENKO, P., HANDLEY, M., AND KARP, B. Wedge: Splitting applications into reduced-privilege compartments. USENIX Association.

[6] CARREIRA, J., FONSECA, P., TUMANOV, A., ZHANG, A., AND KATZ, R. Cirrus: A serverless framework for end-to-end ml workflows. In *Proceedings of the ACM Symposium on Cloud Computing* (2019), pp. 13–24.

[7] CHEN, Y., REYMONDJOHNSON, S., SUN, Z., AND LU, L. Shreds: Fine-grained execution units with private memory. In *2016 IEEE Symposium on Security and Privacy (SP)* (2016), IEEE, pp. 56–71.

[8] DEAN, J., AND GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[9] DUTKA, Ł., WRZESZCZ, M., LICHOŃ, T., SŁOTA, R., ZEMEK, K., TRZEPLA, K., OPIOŁA, Ł., SŁOTA, R., AND KITOWSKI, J. Onedata–a step forward towards globalization of data access for computing infrastructures. *Procedia Computer Science 51* (2015), 2843–2847.

[10] EISMANN, S., SCHEUNER, J., VAN EYK, E., SCHWINGER, M., GROHMANN, J., HERBST, N., ABAD, C. L., AND IOSUP, A. A review of serverless use cases and their characteristics. *arXiv preprint arXiv:2008.11110* (2020).

[11] FENG, L., KUDVA, P., DA SILVA, D., AND HU, J. Exploring serverless computing for neural network training. In *2018 IEEE 11th international conference on cloud computing (CLOUD)* (2018), IEEE, pp. 334–341.

[12] GRAFBERGER, A., CHADHA, M., JINDAL, A., GU, J., AND GERNDT, M. Fedless: Secure and scalable federated learning using serverless computing. In *2021 IEEE International Conference on Big Data (Big Data)* (2021), IEEE, pp. 164–173.

[13] GRAVES, R., JORDAN, T. H., CALLAGHAN, S., DEELMAN, E., FIELD, E., JUVE, G., KESSELMAN, C., MAECHLING, P., MEHTA, G., MILNER, K., ET AL. Cybershake: A physics-based seismic hazard model for southern california. *Pure and Applied Geophysics 168* (2011), 367–381.

[14] GU, J., SONG, S., LI, Y., AND LUO, H. Gaiagpu: Sharing gpus in container clouds. In *2018 IEEE Intl Conf 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), IEEE, pp. 469–476.

[15] HEDAYATI, M., GRAVANI, S., JOHNSON, E., CRISWELL, J., SCOTT, M. L., SHEN, K., AND MARTY, M. Hodor:{Intra-Process} isolation for {High-Throughput} data plane libraries. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)* (2019), pp. 489–504.

[16] HELLERSTEIN, J. M., FALEIRO, J., GONZALEZ, J. E., SCHLEIER-SMITH, J., SREEKANTI, V., TUMANOV, A., AND WU, C. Serverless computing: One step forward, two steps back. *arXiv preprint arXiv:1812.03651* (2018).

[17] ISARD, M., BUDIU, M., YU, Y., BIRRELL, A., AND FETTERLY, D. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (2007), pp. 59–72.

[18] JACOB, J. C., KATZ, D. S., BERRIMAN, G. B., GOOD, J. C., LAITY, A., DEELMAN, E., KESSELMAN, C., SINGH, G., SU, M.-H., PRINCE, T., ET AL. Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. *International Journal of Computational Science and Engineering 4*, 2 (2009), 73–87.

[19] JIA, Z., AND WITCHEL, E. Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2021), pp. 152–166.

[20] JOHN, A., AUSMEES, K., MUENZEN, K., KUHN, C., AND TAN, A. Sweep: accelerating scientific research through scalable serverless workflows. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion* (2019), pp. 43–50.

[21] JUVE, G., CHERVENAK, A., DEELMAN, E., BHARATHI, S., MEHTA, G., AND VAHI, K. Characterizing and profiling scientific workflows. *Future generation computer systems 29*, 3 (2013), 682–692.

[22] KEELE, S., ET AL. Guidelines for performing systematic literature reviews in software engineering, 2007.

[23] KELLY, D., GLAVIN, F., AND BARRETT, E. Serverless computing: Behind the scenes of major platforms. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)* (2020), IEEE, pp. 304–312.

[24] KIM, J., JUN, T. J., KANG, D., KIM, D., AND KIM, D. Gpu enabled serverless computing framework. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)* (2018), pp. 533–540.

[25] KJORVEZIROSKI, V., AND FILIPOSKA, S. Kubernetes distributions for the edge: serverless performance evaluation. *The Journal of Supercomputing 78*, 11 (2022), 13728–13755.

[26] KOTNI, S., NAYAK, A., GANAPATHY, V., AND BASU, A. Faastlane: Accelerating {Function-as-a-Service} workflows. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)* (2021), pp. 805–820.

[27] KUHLENKAMP, J., WERNER, S., AND TAI, S. The ifs and buts of less is more: A serverless computing reality check. In *2020 IEEE International Conference on Cloud Engineering (IC2E)* (2020), IEEE, pp. 154–161.

[28] LUDÄSCHER, B., ALTINTAS, I., BERKLEY, C., HIGGINS, D., JAEGER, E., JONES, M., LEE, E. A., TAO, J., AND ZHAO, Y. Scientific workflow management and the kepler system. *Concurrency and computation: Practice and experience 18*, 10 (2006), 1039–1065.

[29] MAO, M., AND HUMPHREY, M. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (2011), pp. 1–12.

[30] MICHENER, W., VIEGLAIS, D., VISION, T., KUNZE, J., CRUSE, P., AND JANÉE, G. Dataone: Data observation network for earth—preserving data and enabling innovation in the biological and environmental sciences. *D-Lib Magazine 17*, 1/2 (2011), 12.

[31] PÉREZ, A., MOLTÓ, G., CABALLER, M., AND CALATRAVA, A. Serverless computing for container-based architectures. *Future Generation Computer Systems 83* (2018), 50–59.

[32] PÉREZ, A., RISCO, S., NARANJO, D. M., CABALLER, M., AND MOLTÓ, G. On-premises serverless computing for event-driven data processing applications. In *2019 IEEE 12th International conference on cloud computing (CLOUD)* (2019), IEEE, pp. 414–421.

[33] RISCO, S., AND MOLTÓ, G. Gpu-enabled serverless workflows for efficient multimedia processing. *Applied Sciences 11*, 4 (2021), 1438.

[34] RISCO, S., MOLTÓ, G., NARANJO, D. M., AND BLANQUER, I. Serverless workflows for containerised applications in the cloud continuum. *Journal of Grid Computing 19* (2021), 1–18.

[35] ROY, R. B., PATEL, T., AND TIWARI, D. Daydream: executing dynamic scientific workflows on serverless platforms with hot starts. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis* (2022), IEEE, pp. 1–18.

[36] SATZKE, K., AKKUS, I. E., CHEN, R., RIMAC, I., STEIN, M., BECK, A., ADITYA, P., VANGA, M., AND HILT, V. Efficient gpu sharing for serverless workflows. In *Proceedings of the 1st workshop on high performance serverless computing* (2020), pp. 17–24.

[37] SCHLEIER-SMITH, J., SREEKANTI, V., KHANDELWAL, A., CARREIRA, J., YADWADKAR, N. J., POPA, R. A., GONZALEZ, J. E., STOICA, I., AND PATTERSON, D. A. What serverless computing is and should become: The next phase of cloud computing. *Communications of the ACM 64*, 5 (2021), 76–84.

[38] SHAFIEI, H., KHONSARI, A., AND MOUSAVI, P. Serverless computing: a survey of opportunities, challenges, and applications. *ACM Computing Surveys 54*, 11s (2022), 1–32.

[39] SHAHRAD, M., FONSECA, R., GOIRI, I., CHAUDHRY, G., BATUM, P., COOKE, J., LAUREANO, E., TRESNESS, C., RUSSINOVICH, M., AND BIANCHINI, R. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX annual technical conference (USENIX ATC 20)* (2020), pp. 205–218.

[40] SHILLAKER, S., AND PIETZUCH, P. Faasm: Lightweight isolation for efficient stateful serverless computing. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)* (2020), pp. 419–433.

[41] SICARI, C., CARNEVALE, L., GALLETTA, A., AND VILLARI, M. Openwolf: A serverless workflow engine for native cloud-edge continuum. In *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)* (2022), IEEE, pp. 1–8.

[42] SKLUZACEK, T. J., CHARD, R., WONG, R., LI, Z., BABUJI, Y. N., WARD, L., BLAISZIK, B., CHARD, K., AND FOSTER, I. Serverless workflows for indexing large scientific data. In *Proceedings of the 5th International Workshop on Serverless Computing* (2019), pp. 43–48.

[43] STROBEL, V. Pold87/academic-keyword-occurrence: First release, Apr. 2018.

[44] TANG, Y., AND YANG, J. Lambdata: Optimizing serverless computing by making data intents explicit. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)* (2020), IEEE, pp. 294–303.

[45] VAHLDIEK-OBERWAGNER, A., ELNIKETY, E., DUARTE, N. O., SAMMLER, M., DRUSCHEL, P., AND GARG, D. {ERIM}: Secure, efficient in-process isolation with protection keys ({{{{{MPK}}}}}). In *28th USENIX Security Symposium (USENIX Security 19)* (2019), pp. 1221–1238.

[46] WEN, J., AND LIU, Y. An empirical study on serverless workflow service. *arXiv preprint arXiv:2101.03513* (2021).

[47] WEN, J., AND LIU, Y. An empirical study on serverless workflow service. *arXiv preprint arXiv:2101.03513* (2021).

[48] WOHLIN, C. 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* (2014), pp. 1–10.

[49] YEH, T.-A., CHEN, H.-H., AND CHOU, J. Kubeshare: A framework to manage gpus as first-class and shared resources in container cloud. In *Proceedings of the 29th international symposium on high-performance parallel and distributed computing* (2020), pp. 173–184.

[50] YU, M., CAO, T., WANG, W., AND CHEN, R. Following the data, not the function: Rethinking function orchestration in serverless computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)* (2023), pp. 1489–1504.

[51] ZHANG, H., CARDOZA, A., CHEN, P. B., ANGEL, S., AND LIU, V. Fault-tolerant and transactional stateful serverless workflows. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (2020), pp. 1187–1204.