A Review of Serverless Use Cases and their Characteristics

SPEC RG Cloud Working Group

Simon Eismann
Julius-Maximilian University
Würzburg
Germany
simon.eismann@uni-wuerzburg.de

Joel Scheuner
Chalmers | University of Gothenburg
Gothenburg
Sweden
scheuner@chalmers.se

Erwin van Eyk
Vrije Universiteit
Amsterdam
The Netherlands
e.vaneyk@atlarge-research.com

Maximilian Schwinger
German Aerospace Center
Oberpfaffenhofen
Germany
maximilian.schwinger@dlr.de

Johannes Grohmann
Julius-Maximilian University
Würzburg
Germany
johannes.grohmann@uni-wuerzburg.de

Nikolas Herbst
Julius-Maximilian University
Würzburg
Germany
nikolas.herbst@uni-wuerzburg.de

Cristina L. Abad
Escuela Superior Politecnica del Litoral
Guayaquil
Ecuador
cabad@fiec.espol.edu.ec

Alexandru Iosup
Vrije Universiteit
Amsterdam
The Netherlands
A.Iosup@atlarge-research.com

Contents

Executive Summary

The serverless computing paradigm promises many desirable properties for cloud applications—low-cost, fine-grained deployment, and management-free operation. Consequently, the paradigm has underwent rapid growth: there currently exist tens of serverless platforms and all global cloud providers host serverless operations. To help tune existing platforms, guide the design of new serverless approaches, and overall contribute to understanding this paradigm, in this work we present a long-term, comprehensive effort to identify, collect, and characterize serverless use cases. We survey 89 use cases, sourced from white and grey literature, and from consultations with experts in areas such as scientific computing. We study each use case using 24 characteristics, including general aspects, but also workload, application, and requirements. When the use cases employ workflows, we further analyze their characteristics. Overall, we hope our study will be useful for both academia and industry, and encourage the community to further share and communicate their use cases.

Keywords

Disclaimer

1 Introduction

Serverless computing is an emerging technology with increasing impact on our modern society, and increasing adoption by both academia and industry [IDC18, RM17, HSH+16]. The key promise of serverless computing is to make computing services more accessible, fine-grained, and affordable [vTT+18, JSS+19] by managing operational concerns [CIMS19]. Major cloud providers, such as Amazon, Microsoft, Google, and IBM already offer capable serverless platforms. However, serverless computing, and its common Function-as-a-Service (FaaS) realization, still raises many important challenges that may reduce adoption. These challenges have been recognized and discussed in fields such as software engineering, distributed systems, performance engineering [EIST17, VIA+18, HFG+19]. This work focuses on a first step to alleviate these challenges: *understanding serverless applications through a variety of use cases.*

Serverless computing enables developers to focus on implementing business logic, leaving the operational concerns to cloud providers. In turn, the providers turn to automation, which they achieve through capable serverless platforms, such as AWS Lambda, Azure Functions, or Google Cloud Functions, and IBM Cloud Functions (based on Apache OpenWhisk). Serverless platforms already support fine-grained function deployment, detailed resource allocation, and to some extent also autoscaling [CIMS19]. However, more sophisticated operational features have started to emerge such as: (a) complex function composition and even full *workflows,* (b) eventing and provider-managed messaging, (c) low-latency scheduling, (d) file storage and database setup, (e) streaming and locality-aware deployment, and (f) versioning and logging solutions. These features facilitate the serverless *application lifecyle,* and help further decreases the time-to-market for serverless applications [LWSH19].

Researchers and industry practitioners have an urgent need for serverless use cases. The variety of already existing platforms and support from the major cloud providers indicate the presence of many serverless applications. However, relatively little is known about their characteristics or behavior. For an emerging technology such as serverless computing, researchers, engineers, and platform providers could use descriptions of *use cases—which applications?, where and how was this technology already successfully applied?,* and *what are the characteristics of these use cases?—*to guide their drive for discovery and improvement *in the right direction. Researchers* can study different use cases related to the same application to extract meaningful patterns and trigger new designs. They can also identify representative use cases, which can later be used for the evaluation of novel approaches and in empirical studies. *Engineers* require descriptions in which areas serverless computing was already successfully applied, which helps to decide whether to adopt serverless computing for other projects. Additionally, existing solutions can serve as blueprints for similar use cases. *Platform providers* require knowledge of how their products are used, to optimize them and gaps in adoption can point out deficits in their current offerings.

Without access to relevant serverless use cases, the community runs the risk of missing the window-of-opportunity for this technology. The use of serverless computing, across the many application domains and types, programming languages, and complex platforms thwarts full optimization or perfect engineering. Which part of the serverless ecosystem [vIG+19] to tune, re-design, re-deploy, or otherwise change is crucial, and depends on which use cases are considered in the process.

However much needed, serverless use cases have not been studied systematically so far. For serverless computing, existing research has focused on serverless platforms and their performance properties [YBLW19]. Several studies currently exist about the features, architecture, and per-

formance properties of these platforms [vIG+19, BA18, FGZ+18, LSF18, LRC+18, WLZ+18]. Shahrad et al. [SFG+20] characterize the aggregated performance properties of the entire production FaaS workload from Microsoft Azure Functions, but do not provide details on individual use cases. A recent mixed-method empirical study investigates how developers use serverless computing, focusing on the issues (pain points) they experienced [LWSH19]. Another multivocal literature review discusses common patterns in the architecture of serverless applications [TEIPN20]. To the best of our knowledge, the only existing collection of serverless use cases is an article by Castro et al. [CIMS19], which introduces ten use cases collected from non-peer-reviewed (*grey*) literature.

In this technical report, we collect a total of 89 serverless use cases from four different sources. 32 use cases are from open-source projects, 23 from white literature, 28 from grey literature, and 6 from the area of scientific computing. Each use case is reviewed by a pair of reviewers in regard to 24 characteristics, such as execution pattern, workflow coordination, use of external services, and motivation for adopting serverless. The full dataset containing all use cases and their characteristics is publicly available as a persistent Zenodo repository [ESvE+20].

In the next section, we discuss our process for use-case collection and characterization. In Section 3, we describe the 24 characteristics we reviewed for each use case and the results of this review. Section 4 discusses threats to validity and mitigation strategies. Finally, Section 5 concludes this technical report, and discusses promising future research directions based on the finding of this study.

2 Study Design

This section summarizes our overall study process, describes the data sources to identify primary studies, the selection strategy with inclusion and exclusion criteria, the characteristics review protocol, and the discussion and consolidation phase covering inter-reviewer agreement.

2.1 Process Overview

Figure 2.1 summarizes the use case analysis process. Firstly, we compiled an extensive list of potentially relevant use cases from four different data sources (see Section 2.2), namely open source projects, white literature, grey literature, and scientific computing. Secondly, we applied our selection criteria (see Section 2.3) to classify and filter only relevant use cases in the context of this study. This resulted in 83 use cases from publicly available sources and 6 scientific use cases from internal sources, where we had access to domain experts. Thirdly, we defined a list of interesting characteristics including potential values and perform reviews to extract the actual values from available documentation (see Section 2.4). For all public sources, 2 randomly assigned researchers out of a pool of 7 available authors conducted two redundant reviews for each use case. Each scientific use case was reviewed by a single domain expert. Subsequently, we calculated the inter-reviewer agreements for all redundant reviews and resolved any conflicting values during discussion and consolidation (see Section 2.5). This resulted in a total of 89 analyzed use cases.
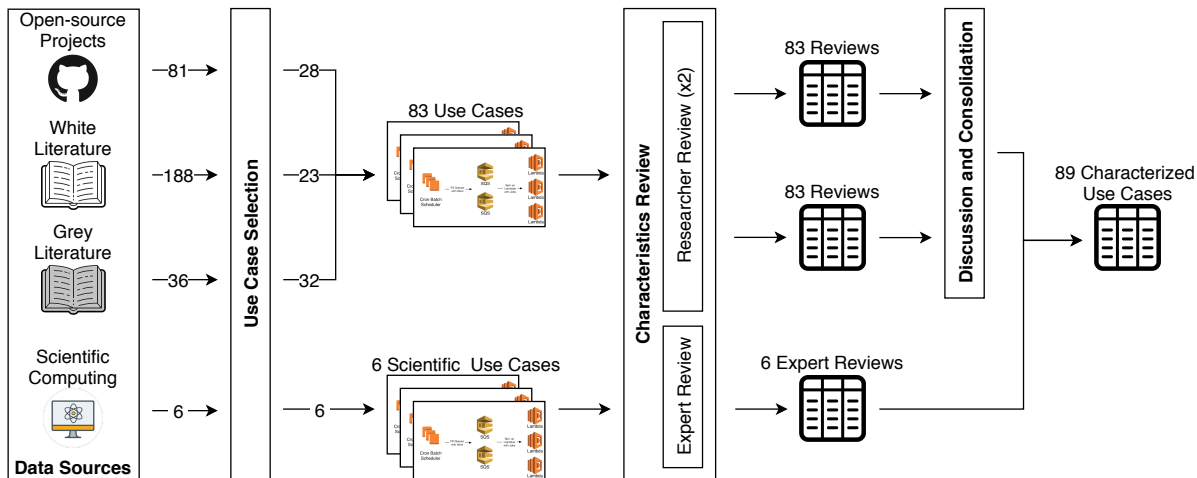
Figure 2.1: Process overview for use case analysis.

2.2   Data Sources

Reports on use cases for serverless applications appear in many different forms ranging from peer-reviewed academic papers, open-source projects, blog posts, podcasts, talks, provider-reported success stories to direct exchange with application developers. Therefore, we collect use cases from a variety of different sources. We also aim to not have a dominant source that contributes the lions share of the use cases and therefore introduces a strong selection bias. Based on this, we do not aim for an exhaustive collection of use cases, but collect use cases from the following different sources with the goal of obtaining a large varied sample:

- **Open-source projects:** Many serverless open-source projects are currently available on GitHub. As a starting point for the open-source projects, we used an existing data set from [PAM19]. This data set was scraped from GitHub using GHTorrent [Gou13], an offline mirror of the GitHub public event time line. It excludes unrelated or insignificant projects, based on a keyword search[1] and also excludes any projects that started prior to the launch of AWS Lambda (the first major serverless platform). From this data set, we removed small and inactive projects based on the number of files, commits, contributors, and watchers. As this still left us with many projects that only mention one of the keywords (e.g., "In the future, we are looking to use AWS lambda for the image resizing."), we manually filtered the resulting data set to include only projects that are deployed as serverless applications. This resulted in a total of 32 use cases from open-source projects.

- **White literature:** There is also a growing interest in serverless applications from academia, which results in a number of scientific publications, i.e., journal papers, conference papers and workshop papers describing serverless use cases. For white literature, we based our search on an existing community-curated dataset on literature for serverless computing consisting of over 180 articles from 2016 to 2019 [SAA19]. First, we filtered the articles based on title and abstract. In a second iteration, we filtered out any articles that implement only a single function for evaluation purposes or do not include sufficient detail to enable a review. As the authors were familiar with a few additional publications describing serverless applications, we contributed these articles to the community-curated dataset and included them in this study. This resulted in a total of 23 use cases from white literature.

---

[1]Keywords: *aws, aws lambda, amazon lambda, lambda functions, azure, openwhisk, serverless, google cloud functions, microsoft azure, azure functions, ibm blue mix, bluemix, oracle fn, oracle cloud fn, kubeless, ibm cloud functions, fn project*

- **Grey literature:** In software engineering, the discourse in not limited to scientific articles but extends to grey literature, such as blog posts, forum discussions and podcasts [GFM16, BJMH15]. For serverless computing, there are a number of blog posts by companies or individuals, talks at industry conferences and provider-reported success stories, as the development of serverless computing was initially mostly industry driven. We filtered the case studies reported by the major serverless providers (AWS[2], Azure[3], Google[4] and IBM[5]) and selected those that used mostly serverless solutions. We also included the ten use cases reported in a recent article on the rise of serverless computing [CIMS19], which to the best of our knowledge is the largest collection of grey literature on serverless use cases. We further extended this collection with grey literature articles describing serverless use cases that the authors were already familiar with. This process resulted in a total of 28 use cases from grey literature.

- **Scientific computing:** There is also an increasing interest in serverless solutions from the scientific computing community (e.g., by NASA [Wal19]). However, most of these use cases are still at an early stage and therefore there is little public data available for them. One of the authors of this paper is currently employed at the German Aerospace Center (DLR), which allowed us to collect information about several projects at DLR that are either currently moving to serverless solutions or are planning to do so. Additionally, a use case from the German Electron Syncrotron (DESY) could be included. This resulted in a total of 6 use cases from the area of scientific computing.

Some use cases are contained in multiple sources, e.g., a use case might have a GitHub repository that matches our keywords and is also used in the evaluation of an academic paper. For these use cases, we assign them only to a single source using the following ranking: open-source projects > grey literature > white literature. For the scientific use cases, there are no overlaps with the other use case sources.

2.3 Use Case Selection

We defined the following inclusion (I) and exclusion (E) criteria for our study:

I1 Concrete serverless use cases, as we are interested in real-world example applications.

I2 Use cases described in sufficient detail to conduct a meaningful review (i.e., excluding vague high-level case studies mainly focusing on a specific serverless platform or solution, but lacking technical detail).

E1 Serverless platforms (e.g., Apache OpenWhisk) and frameworks (e.g., Serverless Framework[6]), as these are not concrete workloads.

E2 Boilerplate code and simple technology demonstrations as often found in official serverless provider documentation, as these do not constitute full-fledged use cases.

E3 Academic papers on the same use case. For example, there are a number of academic papers that discuss serverless neural networks serving [IMS18, BCK+19, TLL18]. In this case we only include a single representative paper.

---

[2]https://aws.amazon.com/solutions/case-studies/

[3]https://azure.microsoft.com/en-in/case-studies/

[4]https://cloud.google.com/customers

[5]https://www.ibm.com/case-studies/

[6]https://www.serverless.com/

2.4 Characteristics Review

We first determined and formalized the set of investigated characteristics. In an initial round, all authors individually suggested characteristics they consider interesting. In a next round, we merged similar characteristics and kept all characteristics that at least two authors considered relevant. This process resulted in 24 characteristics, which can be divided into five groups: general characteristics, workload characteristics, application characteristics, requirement characteristics, and workflow characteristics. *General characteristics* aim to quantify the structure of our data set and include characteristics such as "Is the use case open-source?", "Is the use case currently deployed in production?", and "What domain is this use case from?". *Workload characteristics* aim to describe the traffic pattern and request properties of the use case, e.g., "Is the workload bursty?", "What is the data volume per request?", and "Is the application workload triggered by HTTP requests, cloud events, or regularly scheduled?". *Application characteristics* describe the structure and properties of the serverless application itself and focuses on characteristics such as "How many functions does the application consist of?", "What programming languages are used?", and "Which managed cloud services does the application use?". *Requirement characteristics* describe the requirements from the stakeholders, such as "Is latency relevant?", "Does the application have to run in a specific region, replicated in multiple regions or even on edge devices?", and "What is the reason for the adoption of serverless computing?". Finally, *workflow characteristics* describe the properties of workflows within the serverless applications, e.g., "Is the use case a workflow?", "How many functions does the workflow consist of?", and "How is the workflow execution coordinated?".

Based on a group discussion, we defined an exhaustive set of potential values for each characteristics. For example, for the characteristic "How are executions triggered?" we defined the potential values "HTTP request", "Cloud event', "Scheduled" and "Manual". Additionally, for every characteristic we introduced the values "Unknown" and "Not applicable". "Unknown" indicates that the documentation of the use case does not contain enough information to determine this characteristic. "Not applicable" is used when a characteristic does not make sense for a use cases, for example all workflow characteristics are only applicable to use cases that contain a workflow. For some characteristics, we were not able to define a set of potential values prior to reviewing the use cases. For these characteristics, we used text fragments during the review. Using thematic coding [CA96, GMN11], we extracted codes and treated those as the values for these characteristics. For example, for the characteristic "What is the reason for the adoption of serverless computing?" thematic coding resulted in the codes "NoOps", "Scalability", "Performance", "Maintainability" and "Simplify Development". This process enabled us to extract quantifiable results from the textual descriptions.

We randomly assigned each use case to 2 reviewers out of a pool of 7 available reviewers from the authors. We manually adjusted a few reviewer assignments to minimize the number of coinciding reviewer pairs (i.e., avoid that many use cases are reviewed by the same two reviewers). Subsequently, each reviewer individually assigned values to all characteristics of its assigned use cases.

2.5 Discussion and Consolidation

After completing the initial round of reviews, we calculate the fleiss kappa to quantify the level of agreement between the reviewers [Gwe14]. Due to the nature of the fleiss kappa, we excluded all characteristic assignments, where at least one reviewer assigned multiple values for a characteristic (e.g., if a use case execution is triggered both via HTTP requests and cloud events), the characteristics using thematic coding as well as the numeric characteristic "How

many functions does the application consist of?". As characteristics have a different number of possible values, we calculated an individual fleiss kappa value for each characteristic and then the weighted average across these individual kappa fleiss values. This results in a kappa fleiss value of 0.48, which can be interpreted as "moderate agreement" [LK77].

In the following discussion and consolidation phase, the reviewers compared their notes and tried to reach a consensus for the characteristics with conflicting assignments. In a few cases, the two reviewers had different interpretations of a characteristics. These conflicts were discussed among all authors to ensure that characteristic interpretations were consistent. However for most conflicts, the consolidation was a quick process as the most frequent type of conflict was that one reviewer found additional documentation that the other reviewer did not find. Following this process, we were able to resolve all conflicts, resulting in a collection of 89 use cases described by 24 characteristics.

For the scientific use cases, a different approach was necessary as many of them were not publicly available yet. Therefore, these use cases are reviewed by a single domain expert, which is either involved in the development of the use case or in direct contact with the development. For each of the scientific use cases there is also a textual description (see Appendix Section 1).

3   Analysis Results: On the Characteristics of Serverless Use Cases

We describe in this section the results of our characterization and analysis of serverless use cases. Overall, we cover a diverse set of characteristics, identifying the values commonly used in practice and further analyzing their impact on serverless practice.

3.1   Main Findings

Our main findings are:

1. *General Characteristics:* We find AWS as the currently dominating for platform for serverless applications (80%). The dominating application domain is web services (33%), with 40% of the analysed workloads being business-critical and at least 55% of them in production already.

2. *Application Characteristics:* 82% of all use cases consist of applications that use five or less different functions. Most (67%) of these functions are short-running, with running times in the order of milliseconds or seconds. JavaScript and Python are the most used programming languages for cloud functions (each used by 32% of the cases we studied). These applications depend on a wide variety of cloud services, with the three most used ones being cloud storage (used by 61% of the applications) and cloud database (47%); cloud API gateway (18%) and cloud pub-sub (17%) are also widely used.

3. *Requirements Characteristics:* The reduced operation cost of serverless platforms (33%), the reduced operation effort (24%), the scalability (24%), and performance gains (13%) are the main drivers of serverless adoption. In comparison, cost savings seems to be a stronger motivator than the performance benefits. At the same time, 58% of use cases have latency requirements, 2% even have real-time demands, while only 36% are latency insensitive. Locality requirements are only relevant for 21% of the total use cases.

4. *Workload Characteristics:* 81% of the analyzed use cases exhibit bursty workloads. This highlights the overall trend of serverless workloads to feature unpredictable on-demand workloads, typically triggered through lightweight (<1MB) HTTP requests.

5. *Workflow Characteristics:* Although the presence of workflows is already sizable (31% of the use cases), most workflows are of simple structure, small, and short-lived. This is likely to change, as demand follows natural trends and orchestration methods move toward (cloud-native) workflow engines.

3.2   General Characteristics

In this section, we analyse general characteristics of serverless use cases: the supported platform(s) and application types. Furthermore, we check if a serverless use case is in production yet and its availability as open source. Last, we report on the distribution across application domains for the analysed use cases.

3.2.1   Platform

Description. In November 2014, Amazon released the first commercial Function-as-a-Service platform with AWS Lambda and started the serverless trend. Two years later in 2016, Microsoft Azure, Google Cloud, and IBM Cloud released their own Function-as-a-Service platforms. There are also a number of open-source Function-as-a-Service platforms, such as Knative, OpenWhisk and OpenLambda. Selecting a deployment platform is a major decision for serverless applications, as there is a strong vendor lock-in that makes changing the deployment platform at a later point in time difficult. In this study, we grouped the deployment platforms into *AWS*, *Azure*, *IBM Cloud*, *Google Cloud*, and *Private Cloud*.
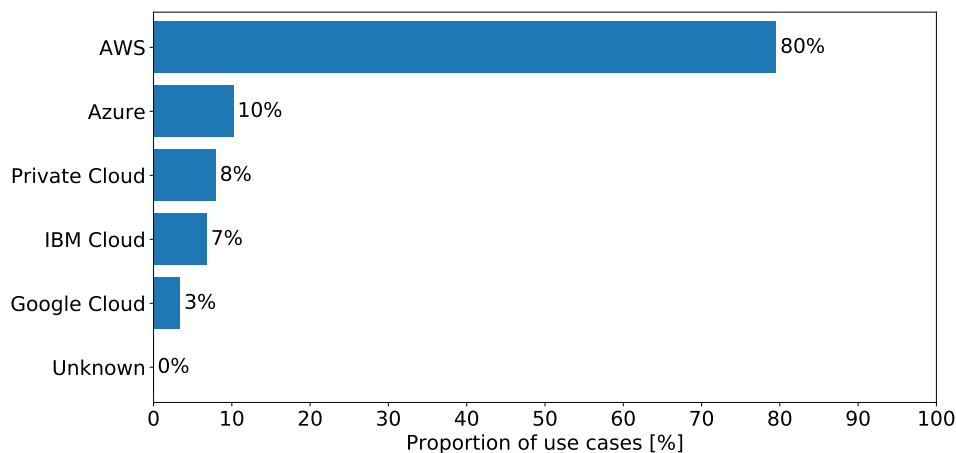

Figure 3.1: Distribution of deployment platform among the surveyed use cases. Some use cases support multiple deployment platforms.

Results. Among the use cases we surveyed, AWS is the clear choice-leader, with 80% of the use cases choosing AWS as their deployment platform. The other cloud vendors are far behind, with Azure at 10%, IBM at 7% and Google Cloud with 3%. 8% of the use cases use a private cloud, with the majority of them being scientific use cases. A total of five use cases can be deployed across multiple cloud platforms.

Discussion. That AWS is by far the most popular choice for serverless deployment among the cases we surveyed can probably be attributed to AWS having a two year head-start with offering this technology as a commercial service. A consequence of the earlier head-start is that there was more time to develop and report about serverless applications using AWS serverless

technology. Additionally, AWS has the largest market share when it comes to general cloud computing [Gar19], which gives it a larger existing user base that can move applications to serverless.

The very low adoption of private clouds outside of the scientific workflows is in strong contrast to the large number of open-source Function-as-a-Service frameworks that have been developed. A large appeal of the serverless application model is the reduction of operational concerns, so we hypothesize that the increase in operational concerns that comes with maintaining a fleet of servers and an open-source Function-as-a-Service frameworks is deterring the adoption of these frameworks. Additionally, most serverless applications make use of many managed services (storage, databases, messaging, logging, streaming, etc.) which are not available directly in a private cloud environment.

It is interesting that five of the use cases we studied can be deployed across multiple cloud platforms. This goes in contrast to the commonly reported vendor lock-in of serverless computing [AC17, Eiv17]. However, upon closer inspection, three of these use cases are computation frameworks that provide additional layers of abstraction on top of commercial Function-as-a-Service and another one is a monitoring framework that utilizes serverless technologies. Therefore, we conclude that while there are some frameworks that can operate across multiple cloud platforms, most serverless applications can only be deployed on the cloud platform they were initially developed for. This provides further evidence that vendor lock-in exists for serverless applications.

3.2.2   Application Type

Description. We categorized each use case according to their type of serverless application. The motivation behind this was to explore for what kind of tasks a serverless approach is typically employed—whether there are certain application types that are dominant or application types that are notably missing in current use cases. To evaluate this usage aspect, we added the *Application Type* metric in which we provided options of typical application types to group the use cases in:

1. *Monitoring:* Contains use cases that employ serverless to monitor or analyze the behaviour of other systems. These applications can include functions that periodically gather metrics, test the availability of a system, or process aggregated logs.

2. *Side-task:* Groups the serverless applications that perform a background task; a task that typically has very lenient performance requirements.

3. *Scientific workload:* Groups the use cases that employ serverless to execute experiments. These can include data generation workloads, such as executing benchmarks; data processing, such as processing seismic images; or data analysis.

4. *Operations:* Consists of the use cases that deploy serverless application to assist in operating other systems. Examples of such applications include automation of the test or deployment pipelines, failure mitigation or remediation, or controlling the state of running systems.

5. *Business-critical workloads:* Contains the use cases that employ serverless for workloads that are business critical [SvBI15]; applications that directly serve the critical business logic of an organization. This type of application can include, for example, serving the website of a web service, or the order management of a retailer.

6. *Other:* Serves as a catch-all group for use cases that had an application type that did not fit in any of the other options.

7. *Unknown:* Denotes the use cases that did not provide enough information about what type of serverless applications were employed.
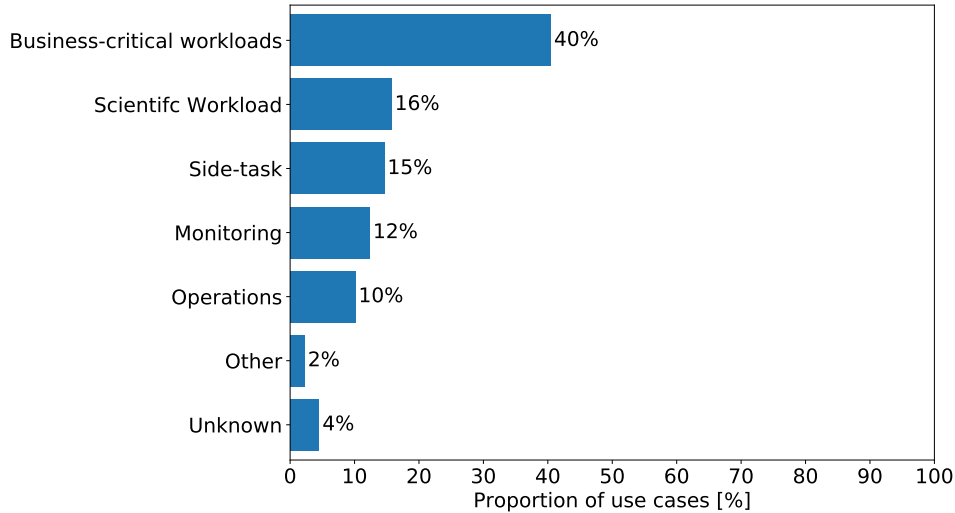

Figure 3.2: Application type distribution among the surveyed use cases.

Results. Figure 3.2 provides an overview of the collected results for the *application type* metric. In contrast with the early advice that serverless should only be used in non-critical applications, we find that nearly half (40%) of the use cases employ serverless applications that are business-critical. The other non-critical groups, Monitoring, Side-task, and Operations are used in a roughly similar number of use cases—12%, 15%, and 10%. The scientific workloads make up 16% of all serverless use cases. A small percentage of the use cases had an other application type (2%), and 4% of the use cases did not have enough information to determine the application type, and were grouped into Unknown.

Discussion. Given the results, we found that a large part of the surveyed use cases are utilizing serverless for business-critical functionality. This reliance indicate that serverless has already started to be adopted as a reliable, mature-enough technology. Next to this, we find that the application types that serverless is employed for, are widely varying, from scientific workloads, to monitoring, to operational tasks.

3.2.3 In Production

Description. Another characteristic that we analyzed is whether or not a specific use case was actually deployed in production environments. Our motivation behind this is to evaluate how reliable the given characteristics actually are, or how representative they are for the real applications running in practice. Therefore, this metric can be seen as a kind of validation of our results.
For this characteristic, we work with three possible values:

- *Yes*: There is clear evidence or statements claiming that the specific use case is already deployed in production.

- *No*: There is strong evidence that the specific use case is not deployed in production.

- *Unknown*: There can be no information found to support either "Yes" or "No".

However, in our analysis an "Unknown" has almost certainly to be seen as a "No", as if we can not find evidence supporting that a use case was not applied in production, we have to assume that it is not.
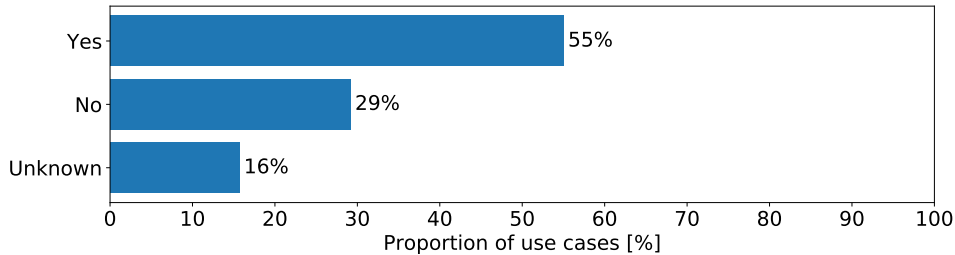

Figure 3.3: Percentage of the survey use cases that are deployed in production.

Results. The results of this characteristic are shown in Figure 3.3. We observe that more that half (55%) of analyzed use cases are actually designed for and deployed in production. Roughly a quarter (29%) is not used in production, and for the remaining 16% of use cases no clear answer could be found. However, as discussed above, we should assume that all "Unknown" use cases belong to the "No" field, leaving us with a total of 45% that do not have strong claims supporting their application in production.

Discussion. As more than half of the use cases included in this study are actually used in production, this can be seen as a strong indicator that the results of our analysis are representative and that the developed best practices have been applied in practice. Furthermore, many of the approaches not actively used in production actually originate from white literature. As most of the white literature papers just present prototypical studies and evaluation use cases, their share of non-production use-cases is significantly higher. However, this in turn increases the respective share of in-production use cases for grey literature and GitHub Projects.

3.2.4 Open Source

Description. Open source indicates whether the source code of the FaaS function or application is publicly available. Open source software is a valuable contribution for education, reuse, and testing. This is exemplified through FaaS providers sharing their own vision for high-level reference architectures[7] and fostering cataloging of example applications[8]. *Yes* indicates that a use case is open-source and *No* indicates that no source code is available. Notice that we barely check the availability of open source artifacts and cannot make any claims about completeness, maintenance levels, or use of appropriate licenses for this characteristic.

Results. Figure 3.4 shows that 53% of the use cases are open source and 47% are closed source.

Discussion. Open source software is typically hosted on GitHub and similarly common for use cases deployed in production. Interestingly, open source software is comparably widespread among the 49 use cases deployed in production with 49%. We expected a clearer tendency of use cases deployed in production to remain closed source, which is probably due to our selection strategy favoring open source use cases.
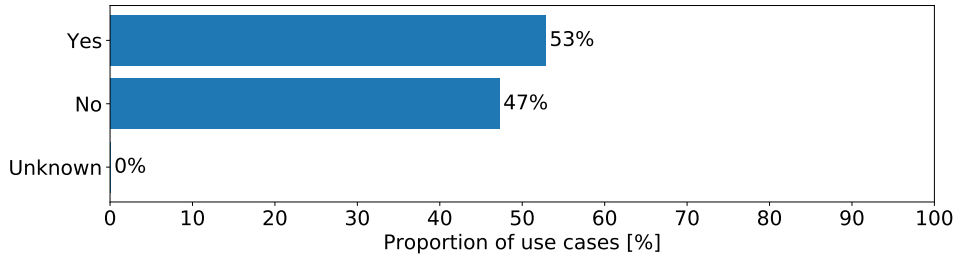
---

[7]https://aws.amazon.com/lambda/resources/reference-architectures/
[8]https://aws.amazon.com/serverless/serverlessrepo/

Figure 3.4: Percentage of the survey use cases that are open-source.

3.2.5 Domain

Description. For each use case, we classified their application domain based as one of the following: *IoT*, *entertainment*, *scientific computing*, *WebServices*, *public authority*, *university*, *FinTech*, *cross-domain*, or *other*. Cross-domain are those use cases that are generic and could be useful across more than one domain; for example, a generic image identification service which could be used in IoT, scientific computing, WebServices, public authority, and university domains.
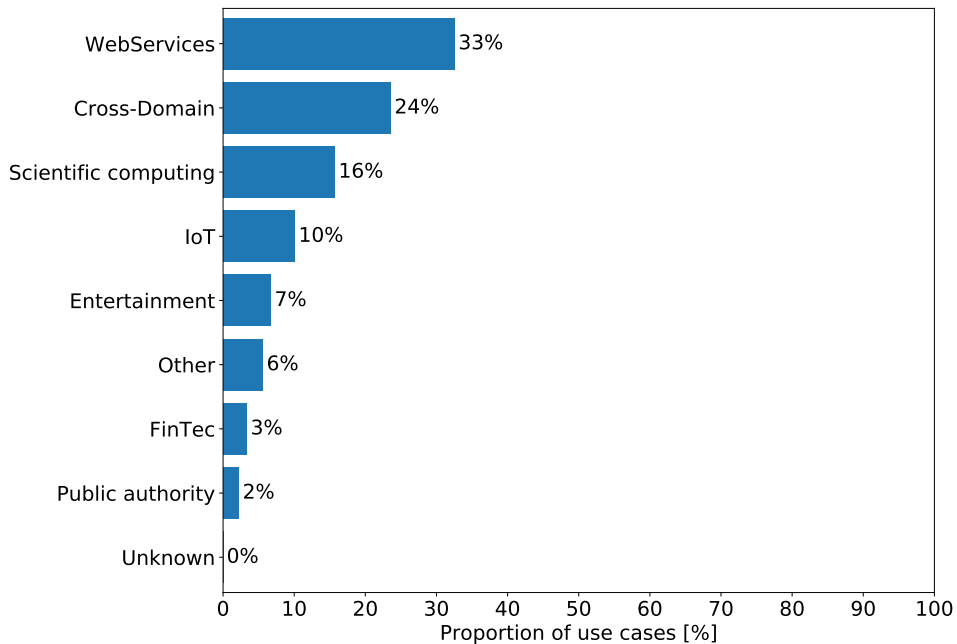

Figure 3.5: Distribution of the domain of the surveyed use cases.

Results. Unsurprisingly, WebServices is the most common application domain in our survey (33%, see Figure 3.5). This is followed by cross-domain use cases (24%), which we mostly found via our GitHub search. The other groups with high representation in our review were scientific computing (16%) and IoT (10%). The significant presence of scientific computing cases is a result of our conscientious efforts in including scientific computing use cases in our survey.

Discussion. We find that there is a wide variety of application domains represented in our survey. This is a strength of our study, as others can use our insights to make decisions about the design and implementation of serverless frameworks that are applicable to a broad variety of domains.

11

3.3 Workload Characteristics

This section characterizes the nature of workloads imposed on serverless applications through human users or technical invokers. In the following, we discuss execution patterns, burstiness, trigger types, and common data volumes of our reviewed use cases.

3.3.1 Execution Pattern

Description. Functions or workflows of functions can be triggered *on-demand* as a direct result of a user interacting with the application, or they can be *scheduled* to be run at specific times. For the on-demand workflows, we further classify them as regular *on-demand* or *high-volume on-demand*.
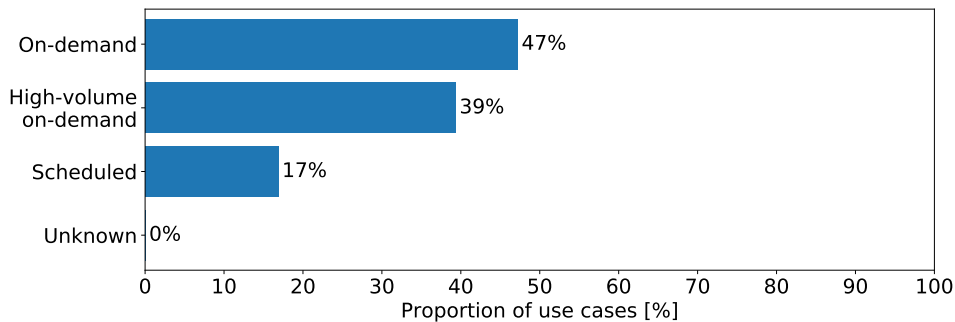

Figure 3.6: Execution pattern distribution among the surveyed use cases. Some use cases are executed both on a schedule and on-demand.

Results. Most (workflows of) functions are triggered on-demand, with scheduled triggers being used in only 17% of the use cases we analyzed (see Figure 3.6). Out of the on-demand execution patterns, close to half are high-volume, business-critical invocations.

Discussion. The high prevalence of high-volume on-demand triggers calls for special study in minimizing function start-up times, and auto-scaling mechanisms. In addition, half of the workflows that are scheduled fall into the application type category *operations* (see section 3.2.2), highlighting how the serverless model has been adopted—in many cases—to automate operations, software management, and DevOps pipelines.

3.3.2 Burstiness

Description. The workload of a function can be either bursty or non-bursty. A bursty workload follows a workload pattern that includes certain sudden and unexpected load spikes, or alternatively a significant amount of sustained noise and variation in intensity. We classify a use case as bursty (i.e., *yes* for burstiness) if its workload typically includes or can include burst patterns in some situations and as non-bursty (i.e, *no*) if the workload is almost guaranteed to never receive burst patterns (e.g., if all executions are scheduled and known in advance). If the burstiness of a use case is *unknown*, then the use case was either under-specified, or can be both bursty or non-bursty depending on the specific area of application. Note that in any scenario that involves a set of human users, we consider the workload pattern to be bursty, as user behavior can almost never the scheduled or reliably controlled, leading to a possibly bursty behavior.
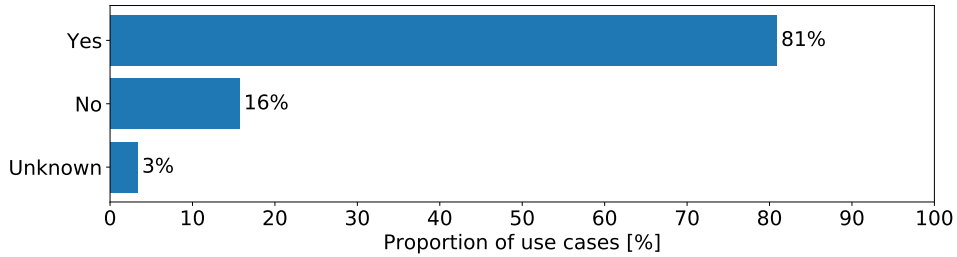
Figure 3.7: Percentage of the survey use cases that have a bursty workload.

Results. Figure 3.7 depicts that a large majority (81%) of the analyzed workload patterns are classified as bursty. Additionally, a 16% share have a clear non-bursty workload pattern, while a small minority of 3% could not be attributed to be either bursty or non-bursty.

Discussion. As one of the strengths of serverless computing is its seamless and almost infinite scalability, together with the general ease of operations it comes as no surprise that most of the use cases indeed experience bursty workload patterns. Early adopters that want to test out the new emerging paradigm are more likely to choose a use case that is optimized for the serverless offering. At the same time, engineers that have been struggling with bursty workloads and face regular performance issues are also more likely to migrate or adopt their application towards the a serverless clouds than applications that run smoothly.
An interesting comparison here would be to compare the share of bursty workloads executed on serverless platforms versus the share of bursty workloads of conventional applications. However, we can still conclude that a large majority of use cases designed for or applied to serverless platforms are experiencing bursty workloads and hence make use of the seamless elasticity that these services offer.

3.3.3   Trigger Types

Description. Trigger types refer to alternative ways of invoking a FaaS function and are closely related to external services (see Section 3.4.6). An *HTTP request* can trigger a FaaS function, which then processes the request and generates an HTTP response. This HTTP routing is often implemented through API gateways. A *cloud event* describes a state change happening in a connected cloud service, such as a file upload to cloud storage or a modified value in a cloud database. Such cloud events can be configured to trigger new function executions. A *scheduled* trigger invokes a FaaS function at a defined and potentially recurring time. The category of *manually* triggered functions refers to human-initiated executions typically executed on-demand. Notice that some use cases combine multiple trigger types and thus the sum of proportions exceeds 100%.

Results. Figure 3.8 reveals that the most common trigger types are HTTP request (46%) and cloud event (39%). Far less common are scheduled (12%) and manual (9%) execution triggers. We were unable to derive the trigger type for 3% of the use cases from their insufficient descriptions.

Discussion. We compare our results to the trigger types reported for the production workload of Microsoft Azure functions [SFG+20]. Scheduled triggers and HTTP triggers are both 16-20% more common among Azure FaaS applications in comparison to our analyzed use cases focusing on AWS (85%). The results for the remaining categories very closely (<=2%) match (after mapping some cumulative categories). We conclude that the order of categories is in
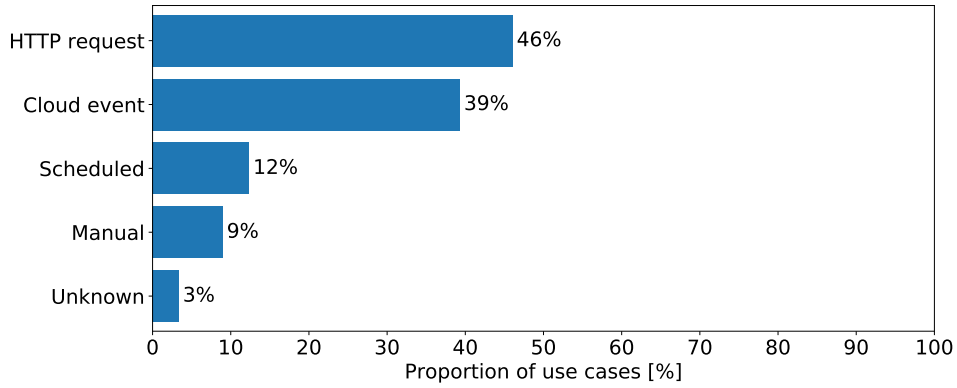
13

Figure 3.8: Trigger type distribution among the surveyed use cases. Some use cases have multiple trigger types.

line with current production workloads reported for Azure but note that some values might be higher in practice. Such an underestimation is plausible given that we derive our results from potentially incomplete sources. However, the explicit grouping of functions into applications in the Azure FaaS implementation possibly leads to different function groupings compared to our AWS-focused use cases.

3.3.4   Data Volume

Description. The data volume defines what load will be on network and storage devices. The motivation here is to analyze whether there are any clusterings of data usages or certain patterns that are generally avoided. We categorized the different use cases into five different categories: Volumes of *less than 1 MB* per execution, *less than 10 MB*, *less than 100 MB*, *less than 1 GB*, and *more than 1 GB*. Additionally, there is also the *unknown* category, if data volume could not be assessed. Note that the data volume refers to executions of the entire workflow. Furthermore, as exact numbers were seldom found in the sources, this categorization is often based on the estimate of our reviewers.
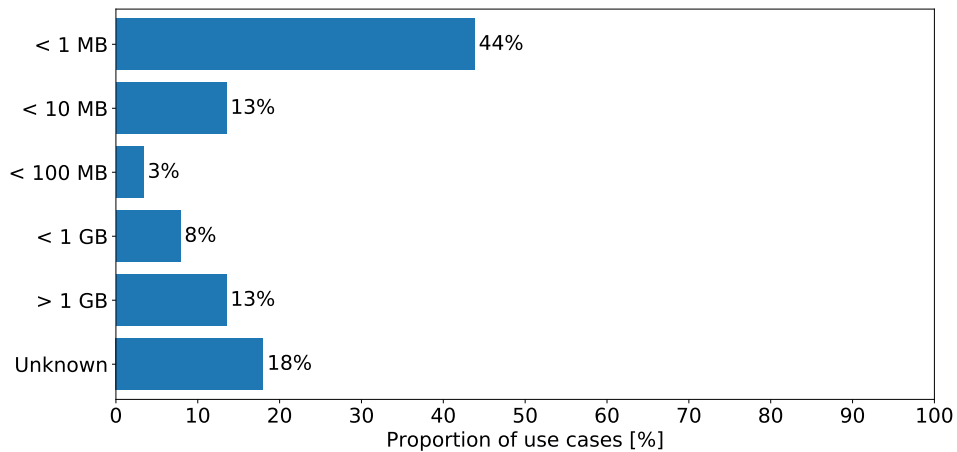

Figure 3.9: Data volume distribution among the surveyed use cases.

Results. Figure 3.9 depicts the distribution of use cases among the different classes. Almost half of the use cases (44%) fall in the smallest category of data volumes of less than 1 MB. The

second categorization transmitting more than 1 MB of data, but less than 10 MB, also make the second largest group with a fraction of 13%. Even less use cases (3%) consider a data volume between 10 and 100 MB. However, the following group between 100 MB and 1 GB increases in popularity, and finally the share of use cases transmitting 1 GB or more to the serverless platform increases to be the second-largest group (13%). Additionally, 18% of use-cases could not have a specific data volume assigned and therefore do not count into any of the enumerated groups.

Discussion. Generally, the different data volumes are relatively distributed and do not cast a clear picture. There is definitely a use case for any data volume characteristic. Therefore platforms should not strive to optimize themselves towards any specific limitation here. That said, most of the use cases transmit less than 1 MB of data per workflow execution. Note that this group also includes all use cases that might not send any data at all. Therefore, the large majority of serverless use cases that we surveyed does not work with big amounts of data. However, there is also the exact opposite group of use cases working with vast vast amounts of data of 1 GB and more per workflow execution.

3.4   Application Characteristics

This section characterizes the how the applications use cloud functions. In the following, we analyze the applications regarding: the number of distinct functions in them, the function run times, the resource bounds of the functions, the programming languages used to implement the functions, the upgrade frequency of the cloud functions, and their interactions with external cloud services.

3.4.1   Number of Distinct Functions

Description. The business logic of serverless applications is contained within serverless functions and connects to a variety of managed cloud services. Similarly to microservices, the appropriate granularity of serverless functions is a controversial topic. Opinions range from wrapping each programming function as a serverless function, each API endpoint as a serverless function to full microservices as a serverless function. In this characteristic we investigate the number of distinct functions within the use cases. As this characteristics targets the development perspective, we count a function that is executed multiple times within an application as a single function. For some use cases, the only information available is along the lines of "more than X functions", which we count as X for this analysis.

Results. About a third (32%) of the analyzed use cases consist of only a single function, as shown in Figure 3.10. Further, about one-fifth (21%) consist of two functions, a tenth (12%) of three functions, another tenth (11%) of four functions, and 5% of five functions. Larger sizes are very rare and without causing a mode in the empirical distribution: there exists in our analysis only one use case for each of the sizes 6, 7, 13, 15, 16, and, notably, 170; there exist only two use-cases for each of the sizes 8, 9, and 10. The use case with more than 170 functions is the back-end for the mobile app of a now defunct start-up. Overall, 82% of all use cases consist of five functions or less. Furthermore, 93% of the use cases that consists of ten functions or less.

Discussion. Our results determine that serverless application use a low number of serverless functions, with 82% of all use cases consisting of five or less functions and 93% of the use cases consisting of less than ten functions. There are two potential reasons for this. First, the serverless application models reduces the amount of code developers have to write, as it allows them to focus on business logic while all other concerns are taken care of by the cloud provider and
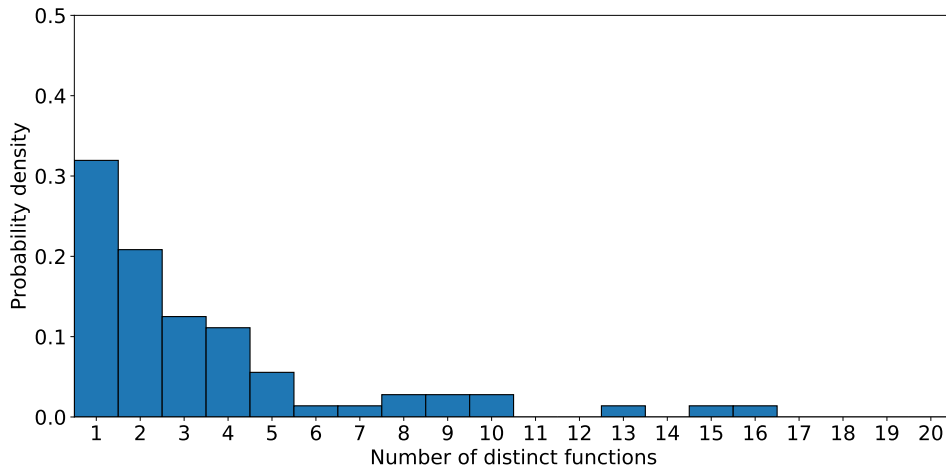
Figure 3.10: Histogram of the number of functions per use case, single outlier at 170 not shown.

managed cloud services. Secondly, this seems to indicate that developers are currently choosing a rather large granularity for the size of serverless functions. However, determining the optimal granularity for serverless functions is still an open research challenge.

3.4.2 Function Runtime

Description. The run time of the cloud functions may have important impact on optimization choices of the serverless frameworks running these functions. We classified the run time of the functions in the use cases as: *short* (order of milliseconds or seconds) and *long* (order of minutes).
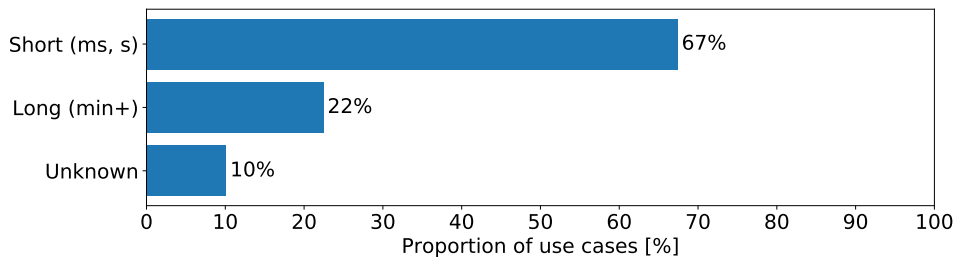

Figure 3.11: Function runtime distribution among the surveyed use cases.

Results. The majority of the functions in our survey are short (ms, s), 67%; only 22% of them have a run time in the order of minutes (see Figure 3.11). We could not assess this characteristic in 10% of the use cases we studied. All but one of the long-running functions is triggered on demand (as opposed to scheduled), with half of them falling into the scientific computing domain. The long-running functions that did not fall within the scientific computing domain, are mostly operations or side-tasks, not business critical. Finally, these long functions are not high-volume on demand APIs (only one was classified as such).

Discussion. The overhead associated with running a function is larger, in proportion, for the case of functions with short run times. This supports the large number of efforts concentrated in reducing this overhead. A limitation of our results is that, as the platforms impose a run time limit in the order of minutes, there may be a bias towards short running functions that would

not exist if there were no time limits to the function run times.

3.4.3   Resource Bounds

Description. We wanted to know if the functions' run time were limited by *I/O*, *CPU*, both (*hybrid*), the *network*, or an *external service* (e.g., cloud database). Information on the workload mix can be useful for studies regarding the scheduling of functions and the routing of execution requests.
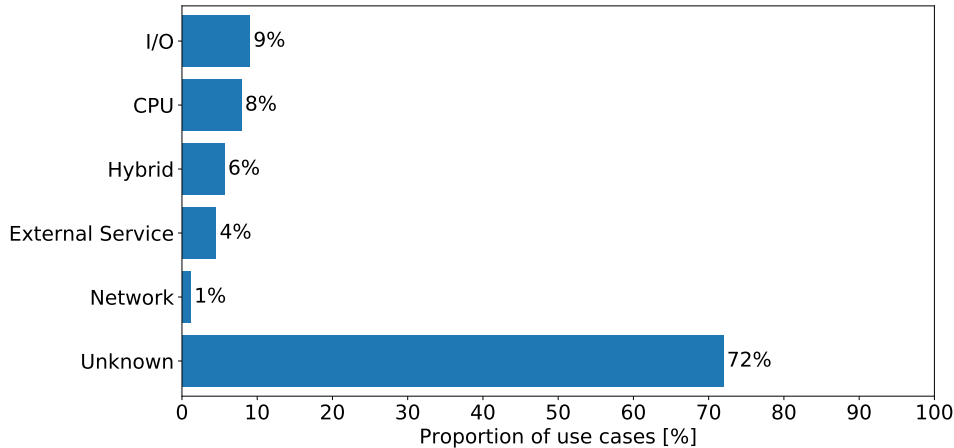

Figure 3.12: Distribution of the resource bounds among the surveyed use cases.

Results. Most use cases did not explicitly state this information (*unknown* is 72%, see Figure 3.12). For the use cases that did report this information, the I/O-bound functions and CPU-bound functions are equally represented in our survey (9% I/O, 8% CPU, 6% hybrid, 4% external service and 1% network).

Discussion. The percentage of use cases reporting this information is too small for us to derive any statistically significant analysis of the results.

3.4.4   Programming Languages

Description. This characteristic refers to the main programming languages used to write code for FaaS functions in a given application. FaaS providers typically offer a set of officially supported runtimes (e.g., Node.js for JavaScript). These execution environments of FaaS functions determine the operating system and pre-installed software libraries. Some providers support further languages through custom runtimes, often in the form of Docker images following a documented interface. Notice that the programming language might differ from the technical function runtime as so called *shims* can be used to invoke a target language through a wrapper runtime (e.g., invoking C++ through Node.js via system calls).

Results. JavaScript (32%) is the most common programming language for FaaS functions tied with Python (32%). Less common languages include Java (9%), C and C++ (8%), C# (6%), Go (3%), and Ruby (1%). We were unable to determine the language for 25% of the use cases due to lacking technical descriptions.
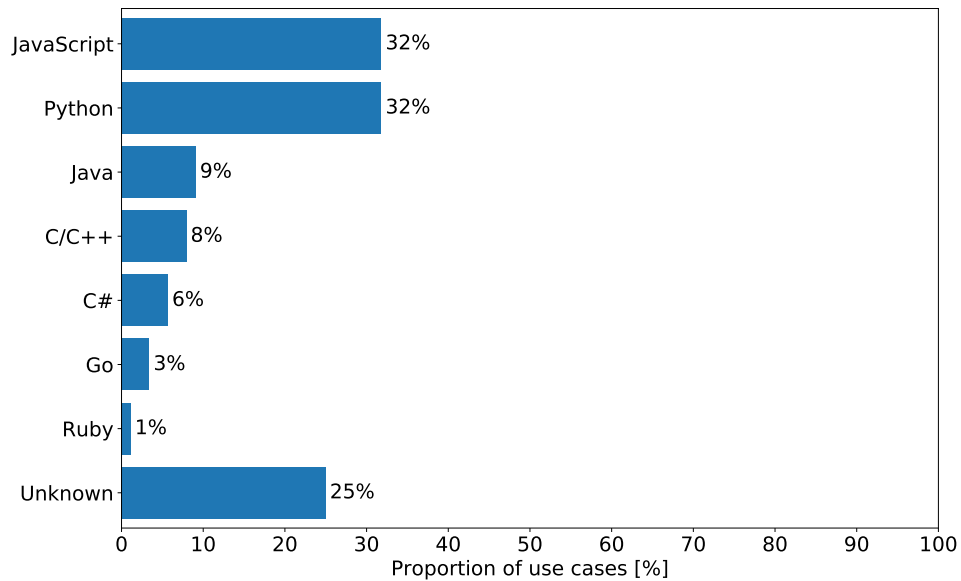
Figure 3.13: Programming language distribution among the surveyed use cases. Some use cases use multiple programming languages.

Discussion. The ranking of languages in our results follows a general trend also observed in other surveys. A study on FaaS industrial practices (N=161) [LWSH19] and initial results of the latest Serverless Community survey (N=109)[9] indicate that JavaScript is 20% more popular than Python on used languages in FaaS applications. However, they largely follow the same ranking but suggest higher popularity of Java over C#. Our results are plausible and confirm that JavaScript and Python are the most widely supported programming languages in FaaS. Further, the remaining languages (i.e., Java, C, etc.) all belong to the world's most popular languages according to the TIOBE index[10], although they are often not the primary choice for the new FaaS paradigm.

3.4.5 Function Upgrade Frequency

Description. How frequently the code of the functions is updated has implications to software engineering and to the mechanisms used by the framework to upgrade the code in the functions that are already deployed. We used two classification levels for this property: *rarely* and *often*; *unknown* indicates that this information cannot be obtained from the use case.
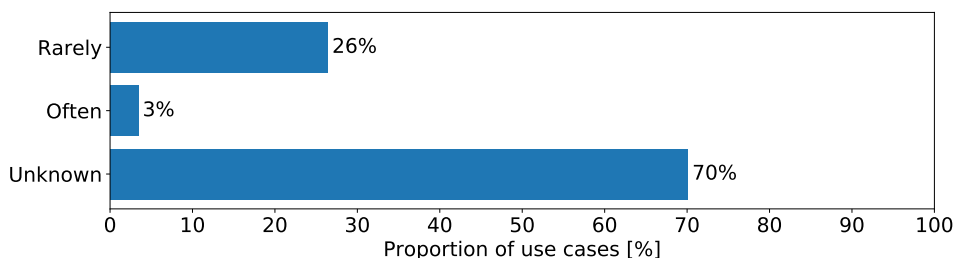

Figure 3.14: Function upgrade frequency distribution among the surveyed use cases.

---

[9]Question 25 in `https://www.nuweba.com/blog/serverless-community-survey-2020-results`
[10]`https://www.tiobe.com/tiobe-index/`

Results. Most use cases did not explicitly state this information (unknown is 70%, see Figure 3.14). For the use cases that did report this information, the functions are updated rarely (26% rarely, 3% often).

Discussion. The percentage of use cases reporting this information is too small for us to derive any statistically significant analysis of the results.

3.4.6 Use of External Services

Description. FaaS functions are often integrated into an ecosystem of serverless external services. Persistency services include cloud storage for blob data (e.g., Amazon S3 for images) and cloud database for structured data storage and querying (e.g., Amazon DynomoDB or Google Cloud SQL). A cloud API gateway exposes HTTP endpoints and can trigger FaaS functions upon incoming HTTP requests. Messaging services include cloud pub/sub for durable asynchronous messaging (e.g., Amazon SNS), cloud queue for reliable FIFO-ordered messaging (e.g., Amazon SQS), and cloud streaming for real-time data ingestion and processing (e.g., Amazon Kinesis). Cloud logging and monitoring refers to applications that explicitly process log data because we implicitly assume some essential logging infrastructure for FaaS functions (e.g., Amazon CloudWatch for AWS Lambda). Cloud ML covers machine learning services, such as Amazon Rekognition for image or video analysis. Notice that we abstracted from vendor-specific services to cross-platform terminology (e.g., AWS S3 becomes Cloud Storage).

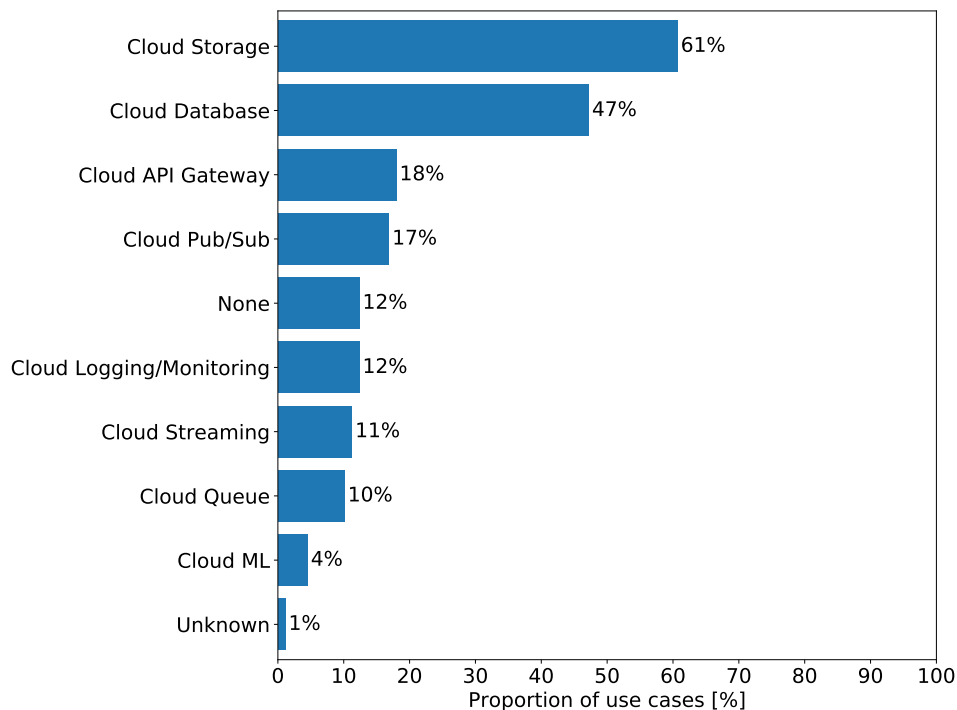Figure 3.15: Distribution of used external services among the surveyed use cases. Many use cases use multiple external services.

Results. Figure 3.15 shows that cloud storage (61%) and cloud database (47%) are the most popular external services, followed by the cloud API gateway (18%) and messaging services (10-17%). For 12% of the use cases, we could not identify any external service integration.

Discussion. Given the ephemeral nature of FaaS functions, it is unsurprising that persistency services are the most popular external services, which is consistent with other survey results [LWSH19]. However, the API gateway receives surprisingly little attention, especially when compared to the 46% of the use cases using HTTP triggers (see Section 3.3.3). We suspect the use of an API gateway is often implicitly assumed, thus not explicitly mentioned, and therefore not comprehensively captured here. Overall, we conclude that currently used external services almost exclusively focus on technical aspects (e.g., storage or messaging) and more specialized services (e.g., cloud ML) are very uncommon among our surveyed use cases.

3.5 Requirements Characteristics

In the following section, we analyze the different requirements and expectations that users have towards the serverless platforms when moving towards those. We discuss the main motivation drivers, as well as the trade-off between cost and performance, and requirements towards latency and locality of the invocations.

3.5.1 Motivation

Description. This characteristic aims at capturing the motivation of the respective engineers and therefore quantifies why they decided to host their application in a serverless environment. For this, we developed six main motivation fields and grouped each use case into one or multiple of those fields, depending on the motivation the authors gave in the description. If no conclusive motivation could be found, we put *Unknown*. The main motivations we found are:

- *Cost*: Running the application in a serverless platform significantly reduces operation cost in comparison to traditional cloud hosting.

- *NoOps*: Deploying a serverless application has the advantage of saving operation effort.

- *Scalability*: The increased scalability of serverless platforms is advantageous for the application.

- *Performance*: The performance of the application, i.e., throughputs and response times, is better when running on a serverless platform.

- *Simplify Development*: The development cycle as well as the release structure is easier using serverless applications.

- *Maintainability*: Deploying an application in a serverless cloud saves maintenance effort.

- *Scalability*: The increased scalability of serverless platforms is advantageous for the application.

Results. The results of our study can be observed in Figure 3.16. The biggest drivers for the adoption of serverless in our use cases are cost (33%), the reduced operation effort (24%), and the offered scalability (24%). Two further significant motivation behind the adoption seems to be the performance benefits (13%) and the simplified development (9%). However, the maintainability (2%) only plays a minor role. For 30% of the use cases, no specific motivation could be determined.

Discussion. As the time savings by employing the NoOps paradigm of the serverless platforms can be converted to personnel costs, we observe that saving effort and costs seems to be a bigger contributor to the adoption of serverless than the offered performance and scalability

improvements (although they are closely behind on second place). This finding is consistent with the results of the cost-performance trade-off in Section 3.5.2.
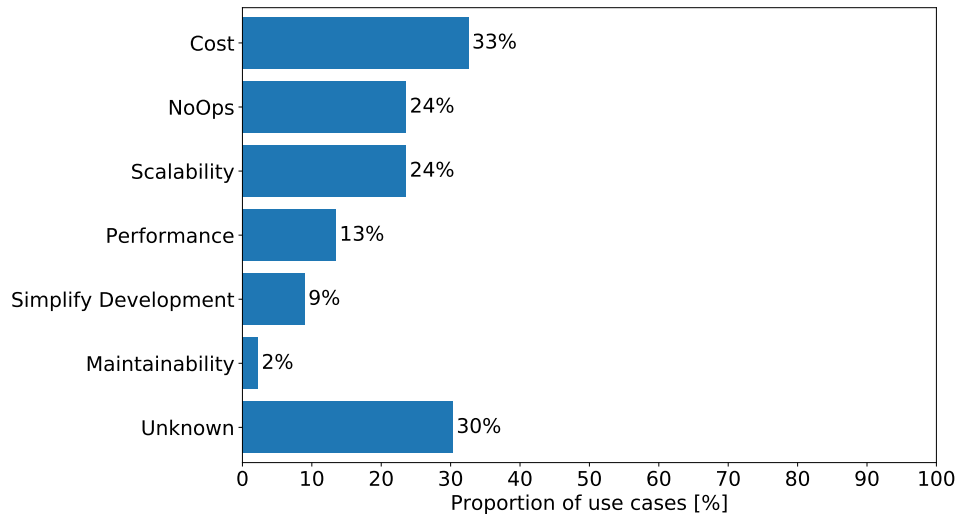

Figure 3.16: Distribution of the motivation behind adopting serverless among the surveyed use cases. Some use cases have multiple motivations.

3.5.2 Cost/Performance Trade-off

Description. The cost/performance trade-off describes whether a use case tends to focus rather on cost optimization (i.e., *cost-focused*) or rather on performance optimization (i.e., *performance-focused*). The trade-off is *undefined* if cost and performance are equally important and *unknown* if we could find no evidence towards any previously mentioned value in the provided use case description.
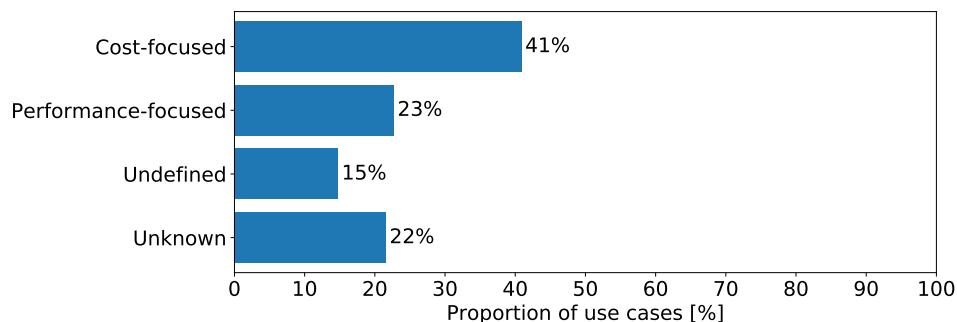
Figure 3.17: Distribution of the cost/performance trade-off among the surveyed use cases.

Results. Figure 3.17 shows that cost is generally more important than performance for 41% of the use cases. Cost-focused use cases are also twice as common compared to performance-focused use case (23%). For 15% of the use cases, cost and performance are equally important. Finally, the trade-off remains unknown for 22% of the use cases.

Discussion. The clear focus on cost optimization is plausible given that cost is a strong motivation for adopting serverless (see Section 3.5.1). Serverless solutions, such as FaaS, were also associated with lower perceived total cost in another study [LWSH19].

3.5.3 Is Latency Relevant?

Description. A diversity of use cases comes with a broad spectrum of expectations or even requirements on latency as a central performance metric. So we posed ourselves the question about the relevance of latency across the analysed serverless use cases. For this characteristic, we distinguish between four levels plus *Unknown*. The levels are:

- *Not important*: For these use cases we found evidence that latency does not play a central role. Delays and variations in latency are acceptable without disturbing the mode of operation.

- *For complete use case*: Latency plays a relevant role for the whole use case on a level of mostly unspecified expectations on latency and its variations over time, e.g., expected to exhibit a latency for convenient human user interaction.

- *For parts of the use case*: The use case includes parts where latency is irrelevant and other parts where latency is of concern following the understanding of the level above as mostly unspecified expectation.

- *Real-time*: We select the level *real-time* if evidence was found that there are soft latency requirements specified. Replies that take longer than a given upper time limit are becoming useless and are not further processed. This interpretation of *real-time* is not implying safety critical states when latency requirements are violated.
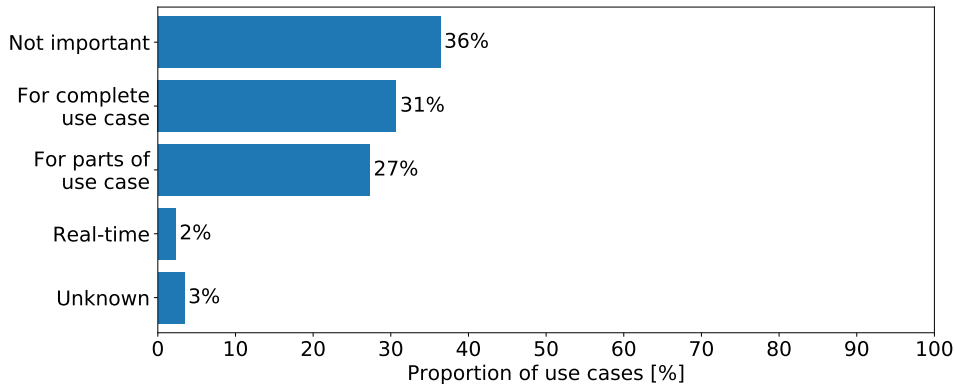

Figure 3.18: Distribution of latency importance among the surveyed use cases.

Results. Figure 3.18 shows that in more than one third (36%) of the use cases, latency does not play a role. On the other side, for 58% of the analysed use cases, latency is of relevance (joining the respective levels). In 27% this is only for parts of the use cases. The portion of use cases with real-time requirements on latency is small with only 2%. No clear assignment of was reasonable for 3% of use cases.

Discussion. Serverless computing is especially convenient for triggered or scheduled background tasks that need to run from time to time without any latency requirements. But issues around function cold-starts and limited function life time have not shown to be a showstopper for serverless uses cases that expect a certain degree of stable latency, e.g. for a smooth interaction with human users. Also, over time, we would expect more examples for serverless use cases that come with stricter soft-real-time requirements as the platforms continue to mature. We doubt that serverless computing will accommodate use cases in production with hard real-time requirements and safety critical implications in case of violations.

3.5.4 Locality Requirements

Description. Migration of any application from dedicated servers in a possibly self-owned data center to a compute infrastructure managed by a cloud provider reduces the control over the locality where the code runs and data is persisted. From the early days of cloud computing on this remains still a possible issue or even show-stopper. We analyse the serverless use cases if requirements on locality are imposed. The reasons for locality requirements can differ broadly from regulatory to performance related ones. Here, we distinguish between four levels of locality requirements plus the case "unknown":

- *None*: There is evidence that for the given use case, no locality requirements are imposed.

- *Multi-region*: The serverless application is or should be deployed in multiple regions, e.g. for improved latency or in tailored variations for specific geographic regions.

- *Specific-region*: The serverless applications is required to run in a specific region.

- *Edge*: The application or parts of it should run closer to a user or IoT device in an edge infrastructure
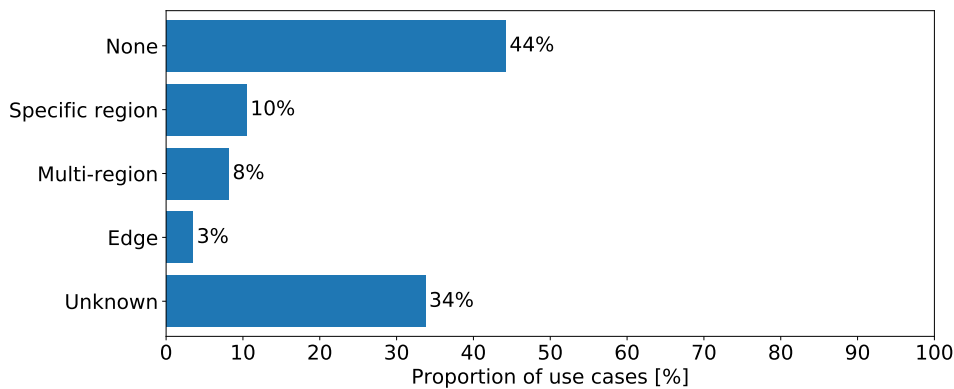
Figure 3.19: Locality requirement distribution among the surveyed use cases.

Results. While we have unclear or unspecified locality requirements for 34% of the use cases, Figure 3.19 shows that the biggest portion (44%) comes with no locality requirements. For 21% of the use cases, we found locality requirements. Out of those, 8% are deployed across regions, while 10% are run in specific regions. The remaining 3% are tailored serverless solutions for edge computing.

Discussion. As serverless technologies and applications are maturing, we expect to see more business-critical elements of serverless applications in daily operation. The portion of serverless use-cases that comes with region-specific requirements will grow respectively. Furthermore, the use of serverless technologies for Edge computing can be seen as a trend of growing importance. Thus, we think it is likely to see a growing importance of the locality requirement "Edge". At the current time, for the dominating part of use cases, locality requirements are apparently not specified or not given yet.

3.6 Workflow Characteristics

Many serverless use cases cannot use a single serverless function to meet their functional and non-functional requirements. Instead, such use cases require the execution of multiple functions,

expressed and orchestrated as *serverless workflows*. In this section, we investigate the characteristics of serverless workflows. However, not all use cases include workflows. Thus, we first investigate in Section 3.6.1 which use cases are based on serverless workflows, and from then on we only report results for the use cases that do (the *workflow use cases*).

3.6.1   Is it a Workflow?

Description. We evaluate here the prevalence of serverless workflows among the surveyed use cases. A use case is categorized as a workflow (bar *Yes* in Figure 3.20) if for a part or all of its functionality multiple serverless functions are needed. If not, the use case is not based on a workflow (*No*). Use cases where this could not be determined are assigned *Unknown*.
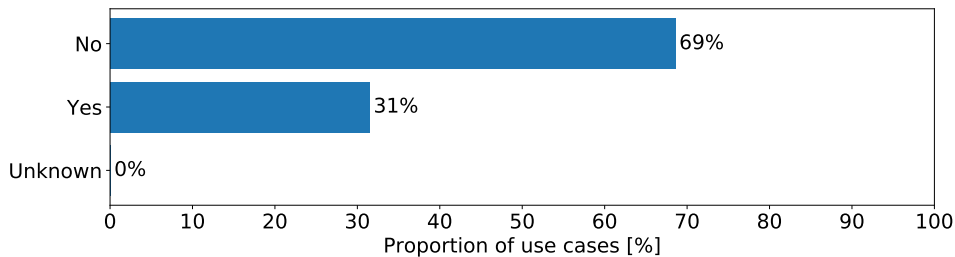

Figure 3.20: Percentage of use cases including workflows, among the surveyed use cases.

Results. As depicted in Figure 3.20, we observe that nearly a third (31%) of the use cases include serverless workflows. The other use cases (69%) are simple enough that one or a couple of independent serverless functions can fully provide the desired functionality. No use case was labeled Unknown.

Discussion. The relative prevalence of workflows in use cases is important, as it hints that serverless use cases are getting more and more complex. The evolution of use cases in fields such as grid computing and more recently cloud computing is indicative that, once workflows become acceptable practice, they become increasingly more prevalent [HTT+09, IH12, DPA+18]. Interesting too is the lack of use cases categorized as Unknown, which indicates that the presence or absence of workflows for any relevant use case is one of the clearest questions to answer.

3.6.2   Workflow Coordination

Description. The use of workflows currently does not constrain the method used for orchestration. There are various approaches to ensure that tasks—or serverless functions—are executed in a coordinated way, e.g., functions can use events to trigger the start of a new task or for other purposes, a task can act as a coordinator for a specific workflow structure, or a workflow engine can orchestrate arbitrary workflows. To evaluate which of these approaches is prevalent, we surveyed their use across workflow use cases. More formally, for this part we categorized orchestration techniques as follows:

1. *Event* groups all use cases that rely on event-driven orchestration. In this approach, workflows are constructed by configuring functions to be triggered to execute on the arrival of the completion (or failure) events of other functions. Typically, this method requires functions to explicitly listen for and publish their results or errors to a message queue—though in some cases platforms this functionality is built-in and no explicit interaction with an external message queue is needed.

2. *Local coordinator* groups all the applications which rely on programmed, user-side logic to take care of the orchestration. An application running on the user's machine, such as a GUI or the client-side JavaScript running on a web page, invokes the functions in the appropriate order, and ensures that each function is executed with the correct configuration and input data.

3. *Workflow engine* contains the use cases that delegate the coordination to a dedicated workflow management system. This workflow engine has functionality to ensure the correct orchestration, along with higher-order concerns, such as (data) provenance, monitorability, and task scheduling optimizations. The workflows typically need to be specified in a consistent format using a set of workflow primitives that are supported by the workflow engine. Compared to the local coordinator, a workflow engine can also be seen as an external, persistent coordinator.

4. *Unknown* captures the use cases where we could not determine the coordination approach, for example because of lacking or lack of documentation.
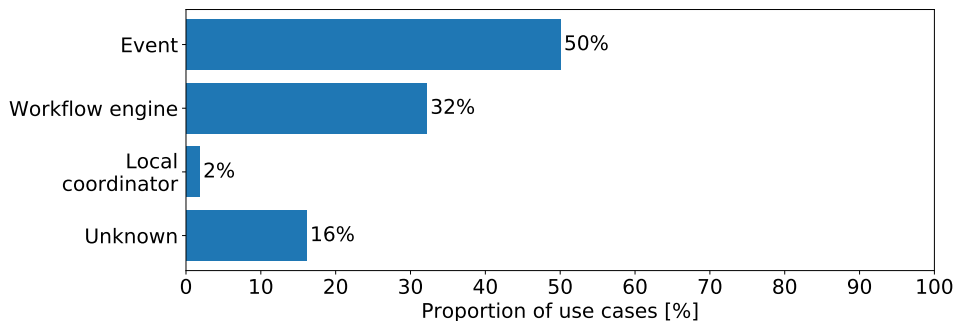

Figure 3.21: Distribution of workflow coordination approach among the surveyed use cases.

Results. As depicted in Figure 3.21, half of the serverless workflows rely on events for the coordination (50%). Slightly less prevalent, about one-third (32%) of the use cases rely on a dedicated workflow engine to ensure correct coordination. Only a few (2%) defer the coordination to a local coordinator. For 16% of the workflow-based use cases the approach could not be determined.

Discussion. Our results indicate that event-driven workflows are currently most prevalent. Inspecting the individual use cases, we find that this is in part caused by implicit workflows; use cases that do not explicitly construct workflows, but instead configure the function triggers in such a way that these form simple pipelines. While this approach can address simple workflows and especially chains of a few tasks, experience from the fields of grid and cloud computing indicates using this approach will not scale to future workflows. We further find that, as the workflows grow in complexity, workflow engines are more often used to coordinate the workflows. In contrast to cloud-side coordination techniques, the use of local coordinators is unpopular because it distributes to the user more complex logic and, in part, because it is difficult to maintain. Furthermore, such an approach could be less reliable in operation – cloud-side coordination techniques and workflow engines are carefully engineered for fault-tolerance, which significantly exceeds the typical development effort of local coordinators.

3.6.3 Workflow Structure

Description. The complexity of a workflow is mostly determined by its structure. A *bag of tasks* is a simple workflow (in mathematical terms: a degenerate workflow), which consists of a

set of tasks that can be executed in any arbitrary order. Another common workflow structure is the *sequential workflow,* where all tasks need to be executed sequentially. We further define *complex workflows* as workflows that include significantly more complex structure than the previous types, including (multi-stage) gather and scatter operations, workflows with conditional execution of (some) tasks, workflows with loops, and fully dynamic workflows.
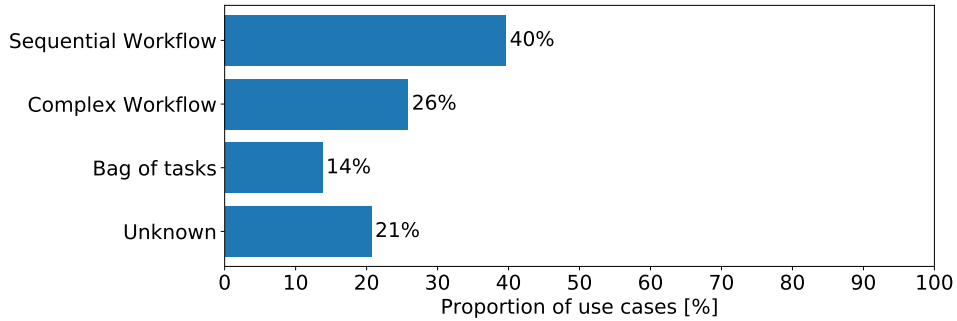

Figure 3.22: Distribution of the workflow structure among the surveyed use cases.

Results. Figure 3.22 depicts the results. Sequential workflows are the most popular workflow structure for serverless applications, with 40% of the workflow use cases including them. Including sequential workflows and bags of tasks (14%), over half (54%) of the workflow use cases only include non-complex workflows. About one-quarter (26%) of the serverless applications are complex workflows. Last, for over one-fifth (21%) of the workflow use cases we were not able to determine the workflow structure.

Discussion. For serverless applications, simple workflow structures (bag of tasks and sequential workflows) are more than twice as common as more complex workflow structures. We hypothesize that this is because serverless applications are currently mostly used for comparatively simple tasks and rarely for complex data analysis. Another possible explanation is the lack of workflow engines (Section 3.6.2); it is difficult to orchestrate arbitrarily complex workflows without such an engine.

The lack of bags of tasks can probably attributed to the fact that serverless applications come with built-in scalability when the functions can be conveniently executed in parallel. For example, resizing a collection of images can be conveniently implemented as a bag of many tasks, where each task invokes the same image-resizing function. In this case, the serverless platform has the capability to execute this case, without further need for orchestration from the user.

3.6.4   Workflow Size

Description. Next, we study workflow size, expressed as the number of tasks in the workflow. We aggregate all use cases into three groups:

1. *Small workflows*, containing 2–10 functions,

2. *Medium-size workflows*, invoking 10–1000 functions, and

3. *Large workflows*, comprised of more than 1000 function invocations.

Results. Figure 3.23 depicts the results of our analysis. The majority (59%) of analyzed workflows are small workflows. Around one fifth of use cases (19%) are medium-size, and only few
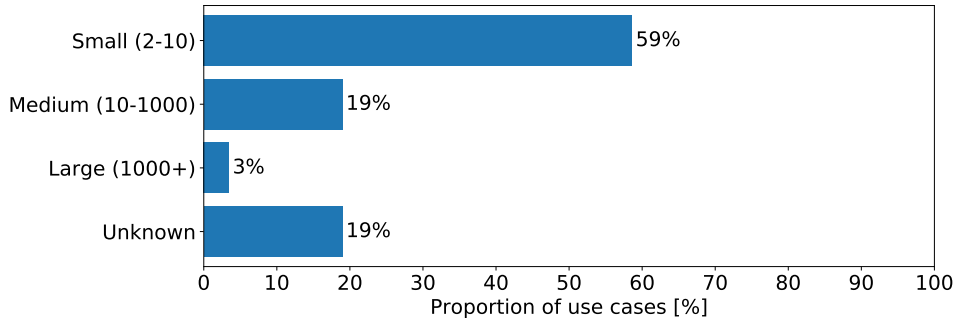
Figure 3.23: Workflow size distribution among the surveyed use cases.

(3%) qualify as large workflows. Nearly one-fifth of the workflows (19%) could not be assigned to one of the groups.

Discussion. Our results suggest that a majority of workflow executions are small; because these workflows are only composed of ten or less individual function executions, they are also relatively short-lived. This is consistent with the characteristics of early workflows in engineering and in scientific prototypes [OIP⁺08]. Only about one-fifth of the workflows are medium or large-sized. Similarly to the previous section, we hypothesize that orchestrating workflows of this size is dependent on the presence of an automated facility, such as a workflow engine. (The other hypothesis introduced in Section, that serverless workflows are currently used for relatively simpler tasks, does not limit the size of the workflow – in the earlier example, the image-resizing workflow can run 10,000s or even 100,000s of functions [BCD⁺08, DPA⁺18].)

3.6.5 Workflow Internal Parallelism

Description. For those use cases in which a workflow of serverless functions was present, we further analyzed whether they present *internal parallelism*—at least an instance of multiple functions running in parallel—or not.
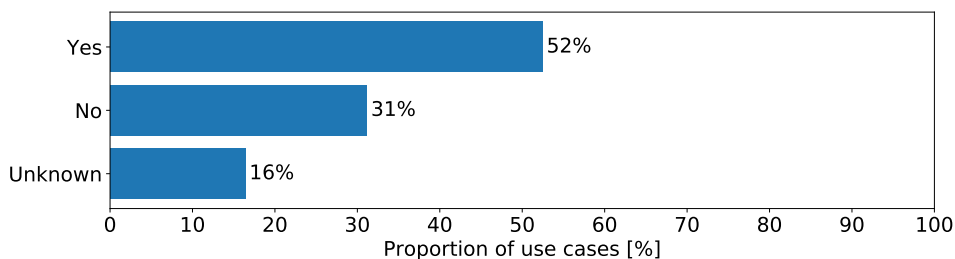
Figure 3.24: Percentage of workflows with internal parallelism among the surveyed use cases.

Results. From Figure 3.24, we observe that most workflows (52%) exhibit internal parallelism; about one-third (31%) of the workflows are simpler, exhibiting no internal parallelism. We could not obtain this information (*unknown*) for 16% of the workflows.

Discussion. The high prevalence of workflows with at least some level of internal parallelism calls for workflow managers that are native to—or well integrated with—the serverless framework, to facilitate workflow composition and management yet deliver parallelism with low overhead.

4 Threats to Validity

We discuss potential threats to validity and mitigation strategies for internal validity, construct validity, and external validity.

4.1 Internal Validity

Manual data extraction can lead to inaccurate or incomplete data. To mitigate this threat, we established and discussed a review protocol prior to reviewing, continuously discussed upcoming questions during the review process, and performed redundant reviews through multiple reviewers. In our review protocol, we established an exhaustive list of potential values for each characteristic and configured automated validation, which immediately highlighted deviations from these values. For characteristics with thematic coding, we continuously refined their values in regular meetings during the review process. To address potential individual bias, we performed two independent reviews for each use case, quantified the inter-rater agreement after an initial review round through Fleiss' Kappa, and resolved each disagreement in an extended discussion and consolidation phase.

4.2 Construct Validity

To align the goal of this study (i.e., comprehensive understanding of existing serverless use cases) with the data extraction, we compiled a list of 24 characteristics covering 5 different aspect groups. We conducted and discussed this selection process together in an international working group with authors from 5 different institutions but other researchers might consider different characteristics as relevant.

4.3 External Validity

Our study was designed to cover use cases from open source projects, white literature, and grey literature but we cannot claim generalizability to all serverless use cases. For open source projects, we filtered non-trivial projects from the most popular open source repository (i.e., GitHub) but might have missed projects published in other repositories. However, we are unaware of such other repositories and also did not discover any among our other use cases from white and grey literature. Our white literature collection is based on a curated dataset on serverless literature and complemented with articles known to the authors but we might have missed more recent articles uncovered in the dataset and unknown to all authors. Grey literature use cases mostly focus on provider-reported case studies, an existing collection of grey literature use cases, and sources known to the authors. We only partially cover corporate use cases as many of them remain unpublished and others provide insufficient details to conduct a meaningful review, which is similar to FaaS platforms [vIG$^+$19]. Our scientific computing use cases are limited to the aerospace domain originating from a national aerospace institution.

5 Conclusion and Future Work

The emergence of serverless computing has already led to a diverse design space, with tens of serverless platforms and the participation of all major cloud providers. We identify in this work the need for a systematic, comprehensive study of serverless use cases, which could help the development of serverless techniques and solutions in the fields of software engineering, distributed systems, and performance engineering.

We have proposed a systematic process to identify, collect, and characterize serverless use cases. To identify use cases, the process considers open-source software projects, peer-reviewed literature, self-published material, and domain knowledge. To collect the use cases, the process proposes a structured repository, from which reviewers take and characterize each use case alongside 24 features of interest. Each use case is covered by the following types of features: (a) *general* characteristics, such as platform, application type and domain, whether the use case was observed in production, and whether the use case provides open-source software; (b) *workload* characteristics, such as the execution pattern, burstiness, types of triggers, and data volume; (c) *application* characteristics, such as programming language(s) used to develop it, the resource bounds, whether the application depends on external services, etc.; (d) the *requirements* posed by the use case, such as locality and latency, or the performance-cost trade-off; and (e) *workflow* characteristics, including structure, size, and internal parallelism.

Using this process, we have collected and characterized a total of 89 serverless use cases from four different sources. Our systematic and comprehensive study reveals that:

1. We find a dominating portion of serverless use cases already being in production with AWS as the most popular platform and web services being the most common application domain.

2. Serverless workloads tend to exhibit on-demand execution patterns exemplified by 81% bursty workloads, which makes their load hard to predict.

3. Most cloud functions (67%) are short-running, with running times in the order of milliseconds or seconds, thus requiring serverless frameworks that impose small overheads when running functions.

4. Cost savings (both in terms of infrastructure and operation costs) are a bigger driver for the adoption of cost than the offered performance and scalability gains.

5. We observe an increasing trend toward ever-larger, ever more complex workflows, indicating the need to move toward (cloud-native) workflow engines.

Last, but not least, we see this study as a step toward a community-wide policy of sharing and discussing about use cases. Persisting beyond our effort, such use cases could stimulate a new wave of serverless designs, facilitate meaningful tuning and benchmarking, and overall prove useful for both academia and industry. We therefore extend an open invitation to prospective new collaborators in the SPEC-RG Cloud group.

References

[AC17]   G. Adzic and R. Chatley, "Serverless computing: economic and architectural impact," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 884–889.

[BA18]   T. Back and V. Andrikopoulos, "Using a microbenchmark to compare function as a service solutions," in *Service-Oriented and Cloud Computing*, K. Kritikos, P. Plebani, and F. de Paoli, Eds.   Cham: Springer International Publishing, 2018, pp. 146–160.

[BCD+08] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on.*   IEEE, 2008, pp. 1–10.

[BCK+19] A. Bhattacharjee, A. D. Chhokra, Z. Kang, H. Sun, A. Gokhale, and G. Karsai, "Barista: Efficient and scalable serverless serving system for deep learning prediction services," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*.   IEEE, 2019, pp. 23–33.

[BJMH15] T. Barik, B. Johnson, and E. Murphy-Hill, "I heart hacker news: Expanding qualitative research findings by analyzing social news websites," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015.   New York, NY, USA: Association for Computing Machinery, 2015, p. 882–885. [Online]. Available: https://doi.org/10.1145/2786805.2803200

[CA96]   A. Coffey and P. Atkinson, *Making sense of qualitative data: complementary research strategies.*   Sage Publications, Inc, 1996.

[CIMS19] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Commun. ACM*, vol. 62, no. 12, p. 44–54, Nov. 2019. [Online]. Available: https://doi.org/10.1145/3368454

[DPA+18] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. Vetter, "The future of scientific workflows," *The International Journal of High Performance Computing Applications*, vol. 32, no. 1, pp. 159–175, 2018.

[EIST17] E. V. Eyk, A. Iosup, S. Seif, and M. Thömmes, "The SPEC cloud group's research vision on faas and serverless architectures," in *Proceedings of the 2nd International Workshop on Serverless Computing, WOSC@Middleware 2017, Las Vegas, NV, USA, December 12, 2017*, 2017, pp. 1–4. [Online]. Available: https://doi.org/10.1145/3154847.3154848

[Eiv17]   A. Eivy, "Be wary of the economics of" serverless" cloud computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 6–12, 2017.

[ESvE+20] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "SPEC RG Technical Report: A Review of Serverless Use Cases and their Characteristics - Dataset," May 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3822191

[FGZ+18] K. Figiela, A. Gajek, A. Zima, B. Obrok, and M. Malawski, "Performance evaluation of heterogeneous cloud functions," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 23, p. e4792, 2018.

[Gar19] Gartner, "Worldwide IaaS Public Cloud Services Market Grew 31.3% in 2018," https://www.businesswire.com/news/home/20190729005169/en/Gartner-Worldwide-IaaS-Public-Cloud-Services-Market, Jul 2019.

[GFM16] V. Garousi, M. Felderer, and M. V. Mäntylä, "The need for multivocal literature reviews in software engineering: Complementing systematic literature reviews with grey literature," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2915970.2916008

[GMN11] G. Guest, K. M. MacQueen, and E. E. Namey, *Applied thematic analysis*. Sage Publications, 2011.

[Gou13] G. Gousios, "The ghtorrent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 233–236. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487132

[Gwe14] K. L. Gwet, *Handbook of inter-rater reliability: The definitive guide to measuring the extent of agreement among raters*. Advanced Analytics, LLC, 2014.

[HFG+19] J. M. Hellerstein, J. M. Faleiro, J. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," in *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*, 2019. [Online]. Available: http://cidrdb.org/cidr2019/papers/p119-hellerstein-cidr19.pdf

[HSH+16] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless computation with openlambda," in *Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'16. USA: USENIX Association, 2016, p. 33–39.

[HTT+09] T. Hey, S. Tansley, K. Tolle *et al.*, *The fourth paradigm: data-intensive scientific discovery*. Microsoft research Redmond, WA, 2009.

[IDC18] IDC, "FutureScape: Worldwide IT Industry 2019 Predictions," https://www.idc.com/getdoc.jsp?containerId=US44403818, Oct 2018.

[IH12] P. K. Isom and K. Holley, *Is Your Company Ready for Cloud: Choosing the Best Cloud Adoption Strategy for Your Business*. IBM Press, 2012.

[IMS18] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 257–262.

[JSS⁺19] E. Jonas, J. Schleier-Smith, V. Sreekanti, C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. J. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud programming simplified: A berkeley view on serverless computing," *CoRR*, vol. abs/1902.03383, 2019. [Online]. Available: http://arxiv.org/abs/1902.03383

[LK77] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[LRC⁺18] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 159–169.

[LSF18] H. Lee, K. Satyam, and G. Fox, "Evaluation of production serverless computing environments," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 442–450.

[LWSH19] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of function-as-a-service software development in industrial practice," *Journal of Systems and Software*, vol. 149, pp. 340–359, 2019.

[OIP⁺08] S. Ostermann, A. Iosup, R. Prodan, T. Fahringer, and D. Epema, "On the characteristics of grid workflows," in *Proc. of the CoreGRID Workshop on Integrated Research in Grid Computing (CGIW'08)*, S. Gorlatch, Ed. CoreGRID, Apr 2008, pp. 431–442.

[PAM19] I. Pavlov, S. Ali, and T. Mahmud, "Serverless development trends in open source: a mixed-research study," Bachelor Thesis, 11 2019. [Online]. Available: https://hdl.handle.net/2077/62544

[RM17] Research and Markets, "$7.72 Billion Function-as-a-Service Market 2017 - Global Forecast to 2021," https://www.businesswire.com/news/home/20170227006262/en/7.72-Billion-Function-as-a-Service-Market-2017---Global, Feb 2017.

[SAA19] J. Spillner and M. Al-Ameen, "Serverless Literature Dataset," Zenodo dataset (3rd revision) at https://doi.org/10.5281/zenodo.1175423, April 2019.

[SFG⁺20] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," 2020.

[SvBI15] S. Shen, V. van Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015*, 2015, pp. 465–474. [Online]. Available: https://doi.org/10.1109/CCGrid.2015.60

[TEIPN20] D. Taibi, N. El Ioini, C. Pahl, and J. R. S. Niederkofler, "Serverless cloud computing (function-as-a-service) patterns: A multivocal literature review," in *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER'20)*, 2020.

[TLL18] Z. Tu, M. Li, and J. Lin, "Pay-per-request deployment of neural network models using serverless architectures," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 2018, pp. 6–10.

[VIA+18] E. Van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, "A SPEC RG cloud group's vision on the performance challenges of faas cloud architectures," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018, Berlin, Germany, April 09-13, 2018*, 2018, pp. 21–24. [Online]. Available: https://doi.org/10.1145/3185768.3186308

[vIG+19] E. van Eyk, A. Iosup, J. Grohmann, S. Eismann, A. Bauer, L. Versluis, L. Toader, N. Schmitt, N. Herbst, and C. L. Abad, "The SPEC-RG reference architecture for FaaS: From microservices and containers to serverless platforms," *IEEE Internet Comput.*, vol. 23, no. 6, pp. 7–18, 2019. [Online]. Available: https://doi.org/10.1109/MIC.2019.2952061

[vTT+18] E. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uta, and A. Iosup, "Serverless is more: From paas to present cloud computing," *IEEE Internet Comput.*, vol. 22, no. 5, pp. 8–17, 2018. [Online]. Available: https://doi.org/10.1109/MIC.2018.053681358

[Wal19] J. Walter, "Systematic Data Transformation to Enable Web Coverage Services (WCS) and ArcGIS Image Services within ESDIS Cumulus Cloud," 2019. [Online]. Available: https://earthdata.nasa.gov/esds/competitive-programs/access/arcgis-cloud

[WLZ+18] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 133–146.

[YBLW19] V. Yussupov, U. Breitenbücher, F. Leymann, and M. Wurster, "A systematic mapping study on engineering function-as-a-service platforms and tools," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, ser. UCC'19. ACM, 2019, pp. 229–240.

Appendix

1 Scientific Use Cases

1.1 Copernicus Sentinel-1 for near-real time water monitoring

Contact persons: Nico Mandery, Torsten Riedlinger, Maximilian Schwinger

Floods occur frequently and across most regions around the globe. They affect lives, infrastructures, economics and local ecosystems. The economic consequences of flood damage impacts the most vulnerable members of society disproportionately. Emergency responders often request Earth Observation based crisis information for flood monitoring to target the frequently limited resources and to prioritize response actions during a disaster situation.

The European Earth Observation program COPERNICUS is providing satellite data and products suitable for a variety of environmental and security applications. The Sentinel-1 radar satellites can be used for various applications in the field of marine and land surface dynamics, e.g. for the detection of water bodies, the mapping of its seasonal dynamics and for the monitoring of flood events. On average, Sentinel-1 provides approximately 1200 scenes per day, creating a daily amount of data in the order of 800 GB.

This application facilitates the near-real time detection and monitoring of flooded areas and can therefore provide vital information for decision makers, scientists and the general public.
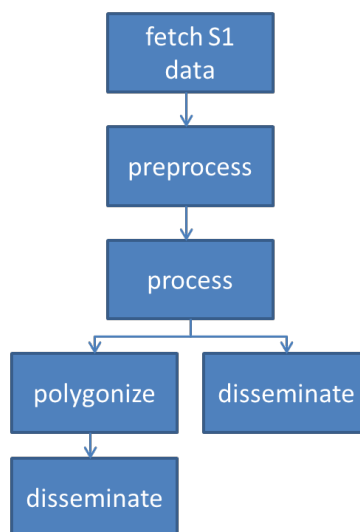

Figure 1.1: Step dependencies during extraction of information from Sentinel 1 data.

A near-real time monitoring system for the provision of Sentinel-1 based products was developed at DLR-DFD in the last years (Martinis et al. 2018, Twele et al. 2016), which is continuously improved in terms of thematic quality and processing capacity.

The radar-based processing chains make use of an automatic hierarchical tile-based thresholding approach in combination with fuzzy-logic-based post-processing for the unsupervised extraction of the flood extent (Martinis et al. 2018). The processing chain can be divided into pre-processing (internal calibration and terrain correction) and the thematic processing (derivation of water bodies, dynamic and flood extent). The results are disseminated as raster and vector files, including the provision through web-services.
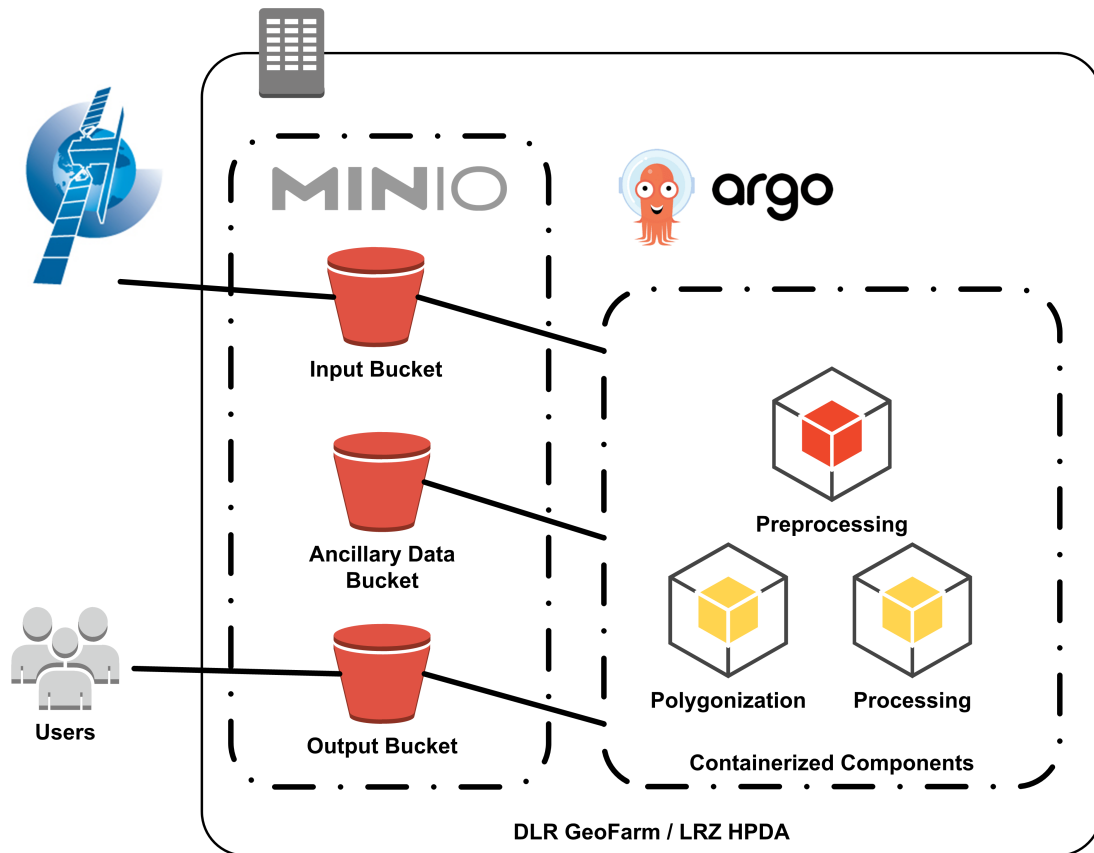
Figure 1.2: Architecture Overview of the S1 water monitoring system implemented at DLR.

The processing is performed in a cloud native environment. Argo Workflows are orchestrating docker container native workflows in a Kubernetes environment. Using Kubernetes and Argo allows an easy evolution of the processing system and the implementation and adapting of other thematic processing tasks in the future.

The physical infrastructure currently used for the Sentinel-1 application is the DLR-DFD multi-purpose processing Environment GeoFarm. The near-real time application is planned to be ported to DLRs new High performance data analytics (HPDA) -Infrastructure at the Leibnitz Rechenzentrum (LRZ) in Munich.

References:

Martinis, S., S. Plank, and K. Cwik, "The use of Sentinel-1 time-Series data to improve flood Monitoring in arid areas", Remote Sensing, vol. 10 (582), pp. 1-13, 2018

Twele, A., W. Cao, S. Plank, and S. Martinis, "Sentinel-1 based flood mapping: a fully automated processing chain," International Journal of Remote Sensing, 2016, vol. 37, no. 13, pp. 2990-3004, 2016

1.2 Terra_Byte - High Performance Data Analytic for Earth Observation

Contact person: Maximilian Schwinger

Under the cooperation agreement Terra_Byte between Europe's second largest high performance computing center LRZ (Leibnitz Super Computing Center) and the DLR (German Aerospace Center) a high performance data analytic infrastructure fit for the specific requirements of earth observation processing is procured and developed. The environment is composed from an optimized hardware layer which can answer the high I/O requirements of earth observation requirements and a software stack, which provides earth observation scientists an easy access to the available resources.

Core of the HPDA Terra_Byte is a large high performance online storage based on the LRZ's Data Science Storage (DSS) concept which provides more than 30 PByte of relevant earth observation data online for use in different applications. The identification and access of the data is one of the major tasks to be accomplished: applications need a simple way to identify relevant data in hundreds of millions of data files, each larger than 1GByte. Besides identification and access of data, usage of the significant computing infrastructure of HPDA-Terra_Byte with simple mechanisms matching the needs of earth observation scientists are required. To provide this easy access and simple scalability the HPDA infrastructures stack will provide Platform as a service (PaaS) as well as Function as a service (FaaS) capabilities.

To achieve this target an analysis and extension of available open source software will take place. For the usage within the infrastructure a set of functions will be developed taking away logistical core tasks from the scientists to provide them the freedom to focus on their core work.
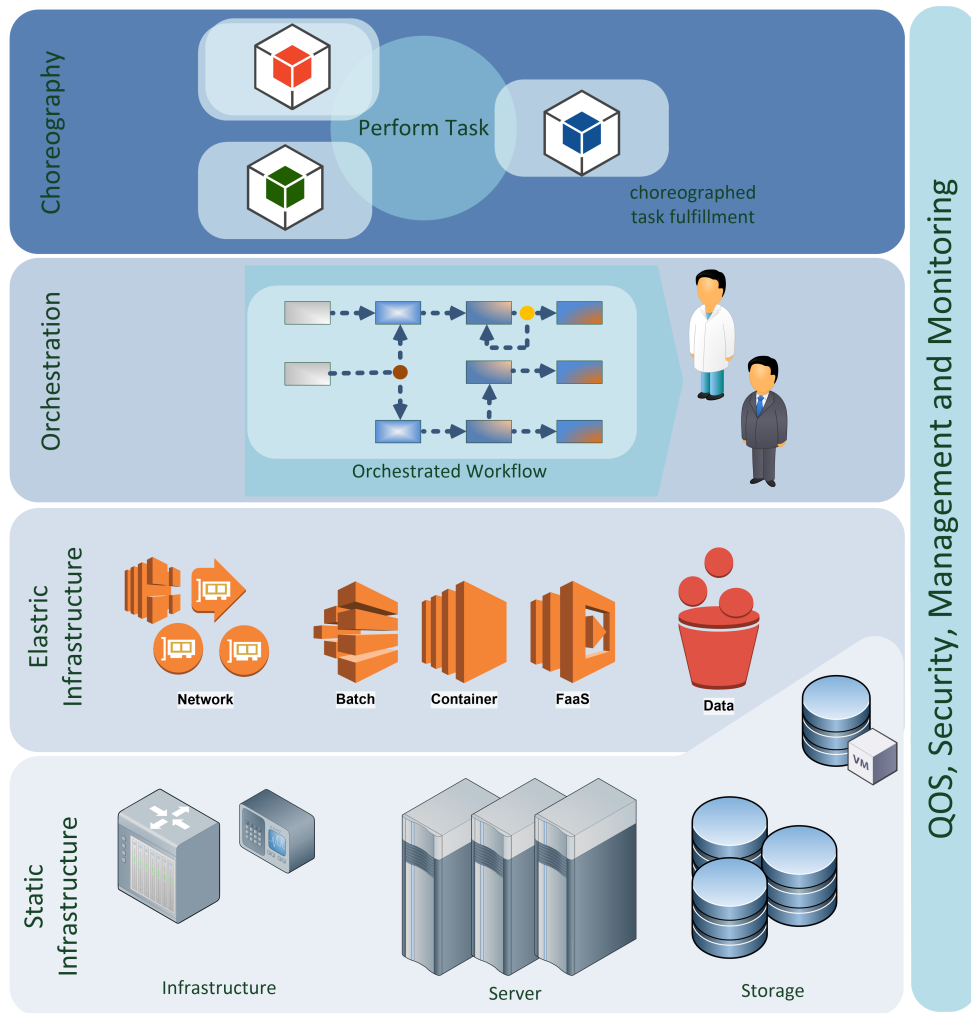
Figure 1.3: Overview of the intended implementation of HPDA Terra_Byte.

1.3   Reprocessing Sentinel 5 Precursor Data with ProsEO

Contact person: Maximilian Schwinger

The European Copernicus program provides a variety of satellite data products fit for applications in monitoring of environment and security. The Sentinel 5 precursor, carrying the hyperspectral instrument TROPOMI, continuously monitors the earth's atmospheric composition. Data is produced continuously but progress in understanding of the data as well as the earth's atmosphere leads to a continuous development in the algorithms retrieving trace gas concentration from the sensor measurements. Due to this progress a reprocessing of the whole missions data is required from time to time. The amount of input here is in a petabyte scale, the number of files to be processed is in the scale of a million files and the time available is weeks.

The reprocessing system proposed has to follow a complex dependency graph of trace gas and cope with a multitude of configuration dependencies between different processor versions and configurations. The combination of processor version and configuration defines the requirement of an input product with a specific processor version and configuration produced. The processing of a product is triggered by a simple request of a specific product produced by a version/configuration of a processor. The dependency graph is generated by prosEO and the triggering of functions on hardware is done by Kubernetes. The processors are encapsulated in
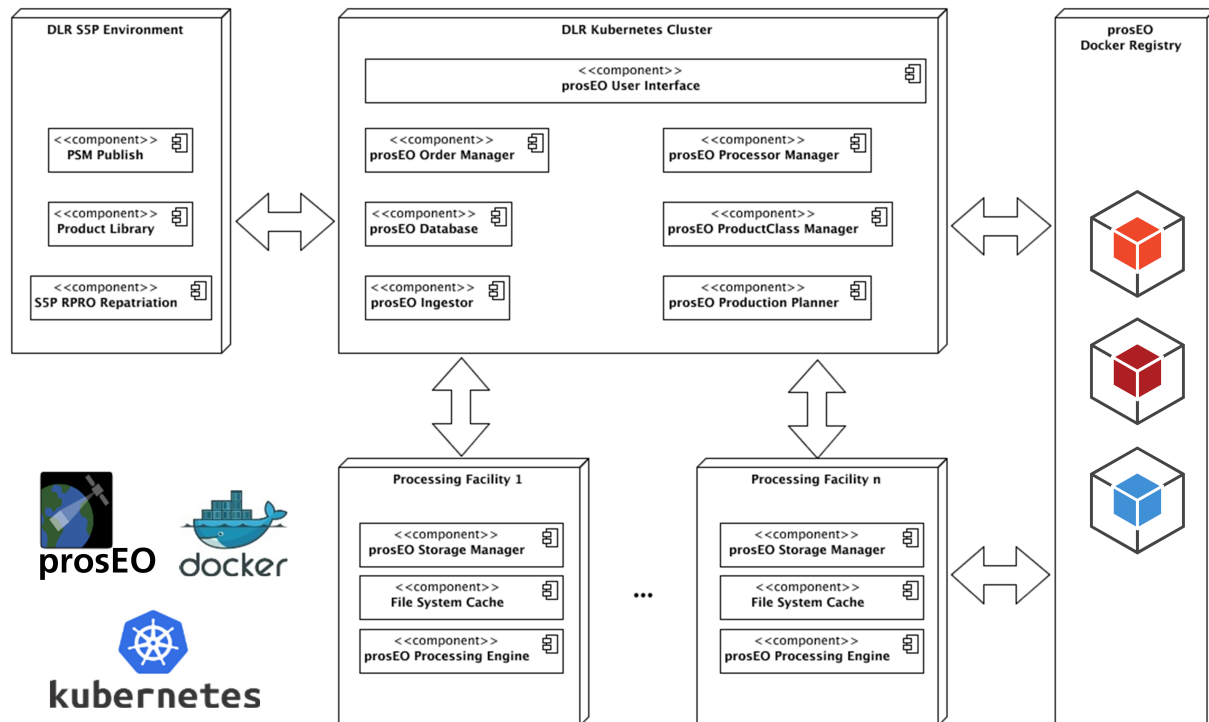
docker images.

Figure 1.4: Architectural Overview of ProsEO - a framework for generating higher level products from satellite data.

prosEO is capable of handling multiple cloud providers (processing facilities) depending on cost and data availability decisions.

1.4 Tandem-L exploitation platform

Contact persons: Wolfgang Balzer, Maximilian Schwinger

The Tandem-L mission is a German L-band Radar project proposal currently in ECSS phase B of its development cycle. Tandem-L is a constellation of two satellites capable of observing dynamic processes on the earth's surface in unpreceded quality.

Tandem-L will generate a daily amount of 8 TByte of raw acquisition data which is processed on ground to 13 interdepending products, operationally. In addition to this it is possible for scientists to implement an own processor which then can be interconnected with the operational processing streams. As no operational knowledge on processing streams, hardware allocation and interdependencies will be handed to external scientists developing processors the processor will be added to the exploitation platform as an additional function with interdependencies to other functions of the exploitation platform.

1.5 Global Urban Footprint

Contact persons: Thomas Esch, Maximilian Schwinger
Source: `https://www.dlr.de/guf`

Currently, more than half of the world's population are urban dwellers and this number is still rapidly increasing. Since settlements—and urban areas in particular—represent the centers of human activity, the environmental, economic, political, societal and cultural impacts of urbanization are far-reaching. They include negative aspects like the loss of natural habitats, biodiversity and fertile soils, climate impacts, waste, pollution, crime, social conflicts or transportation and traffic problems, making urbanization to one of the most pressing global challenges. Accordingly, a profound understanding of the global spatial distribution and evolution of human settlements constitutes a key element in envisaging strategies to assure sustainable development of urban and rural settlements.

In this framework, the objective of the "Global Urban Footprint" (GUF) project is the worldwide mapping of settlements with unprecedented spatial resolution of 0.4 arcsec (˜12 m). A total of 180 000 TerraSAR-X and TanDEM-X scenes have been processed to create the GUF. The resulting map shows the Earth in three colors only: black for "urban areas", white for "land surface" and grey for "water". This reduction emphasizes the settlement patterns and allows for the analysis of urban structures, and hence the proportion of settled areas, the regional population distribution and the arrangement of rural and urban areas. When looking at the entire GUF at once, mainly the metropolitan regions in Europe, USA East Coast and Asia stand out. When focusing on the full resolution of the GUF, one can even recognize small villages stretched along roads, single farm houses or non-built-up corridors in megacities. For a comprehensive and objective analysis of the settlement patterns, the DLR additionally developed an approach to display the spatial networks between the mapped settlements. This enables the computation of various form and centrality measures, which are used for qualitative and quantitative characterization of settlement patterns at different spatial units ranging from global to local scale.

The GUF exhibits a high potential to enhance climate modelling, risk analyses in earthquake or tsunami regions and the monitoring of human impact on ecosystems. Moreover, the it also can be employer as basis for monitoring both the historical growth of different settlements, as well as their ongoing and future development.. This will allow effective comparative analyses of urban dynamics among different regions of the world.

1.6 DESY - High Throughput Data Taking

Contact persons: Patrick Fuhrmann, Michael Schuh, Maximilian Schwinger
Source: `https://www.desy.de/about_desy/desy/index_eng.html`

DESY is one of the world's leading accelerator centres. Researchers use the large-scale facilities at DESY to explore the microcosm in all its variety—from the interactions of tiny elementary particles and the behaviour of new types of nanomaterials to biomolecular processes that are essential to life. The accelerators and detectors that DESY develops and builds are unique research tools. The facilities generate the world's most intense X-ray light, accelerate particles to record energies and open completely new windows onto the universe. That makes DESY not only a magnet for more than 3000 guest researchers from over 40 countries every year, but also a coveted partner for national and international cooperations. Committed young researchers find an exciting interdisciplinary setting at DESY. The research centre offers specialized training for a large number of professions. DESY cooperates with industry and business to promote new technologies that will benefit society and encourage innovations. This also benefits the metropolitan regions of the two DESY locations, Hamburg and Zeuthen near Berlin.