An empirical study on the challenges that developers encounter when developing Apache Spark applications☆

article info

abstract

Apache Spark is one of the most popular big data frameworks that abstract the underlying distributed computation details. However, even though Spark provides various abstractions, developers may still encounter challenges related to the peculiarity of distributed computation and environment. To understand the challenges that developers encounter, and provide insight for future studies, in this paper, we conduct an empirical study on the questions that developers encounter. We manually analyze 1,000 randomly selected questions that we collected from Stack Overflow. We find that: 1) questions related to data processing (e.g., transforming data format) are the most common among the 11 types of questions that we uncovered. 2) Even though data processing questions are the most common ones, they require the least amount of time to receive an answer. Questions related to configuration and performance require the most time to receive an answer. 3) Most of the issues are caused by developers' insufficient knowledge in API usages, data conversation across frameworks, and environment-related configurations. We also discuss the implication of our findings for researchers and practitioners. In summary, our work provides insights for future research directions and highlight the need for more software engineering research in this area.

1. Introduction

The amount of available data has increased significantly in recent years. Forbes estimated that 90% of the data in the world is generated in the last few years (Forbes, 2018). The collected data contains valuable information that helps in making critical business decisions, but the data of such scale is difficult to pro-cess and analyze using a single machine. As a result, developers leverage various distributed data processing frameworks, such as Apache Hadoop (Hadoop, 2021) and Apache Spark (Spark, 2021), to analyze data across tens or even hundreds of machines.

In particular, Spark has become one of the most popular big data frameworks that companies are adopting (Agarwal, 2019). Apache Spark is a unified analytics engine for processing large-scale data. Spark can distribute data processing tasks across large clusters of computers to speed up the computation. Spark pro-vides a set of APIs that abstract the distributed computation details. Thus, developers can call Spark APIs to implement the data processing logic and can scale horizontally without modi-fying the code. Spark supports multiple programming languages (i.e., Java, Python, Scala, and R) and offers various high-level API abstractions to meet a wide range of big data development needs, such as SQL queries, machine learning, and graph analysis.

Although Spark helps developers abstract the underlying distributed data processing details, developers may still encounter various challenges when developing Spark applications. For example, while Spark's abstraction layer automatically optimizes data processing to achieve better performance, such abstrac-tion may also pose challenges in debugging data processing tasks (Wang et al., 2021). Moreover, since developers may in-tegrate Spark applications with other big data frameworks (e.g., storage engines such as HDFS), developers may also encounter integration issues. Spark provides various components, such as MLlib for machine learning and Spark Streaming for stream data processing, to perform a wide range of data processing tasks. However, there may be unique challenges associated with using various components and process different types of data.

In this paper, we conduct an in-depth qualitative study of the problems that developers encounter when developing Spark applications. Prior studies have used Stack Overflow to study the challenges that developers have in various fields, such as security and deep learning applications (Meng et al., 2018; Islam et al., 2019). Similarly, we study both the Spark-related questions and answers on Stack Overflow.

In total, we collected 12,217 posts (i.e., each post contains the question and the corresponding answers). Due to the large number, we conduct our qualitative study based on a statistically significant sample. We randomly sample 1000 posts, achieving a confidence level of 95% and a confidence interval of 3%. While prior research also aimed to study the challenges that big data developers encounter, the study was purely quantitative (Gulzar et al., 2019). In our paper, we study each post in detail and examine the code provided in the questions and answers to obtain deeper insights into the potential challenges in developing Spark applications. In particular, we seek to answer the following research questions:

RQ1: What types of questions do developers ask about Spark on Stack Overflow?
We uncover 11 types of questions that developers encounter when developing Spark applications. We find that data processing is the most common type of questions, accounting for 43% of all the studied questions. We also find that developers often encounter issues related to configuration, integration with other frameworks, and using machine learning APIs in Spark.

RQ2: Which types of questions have higher view counts and are more time-consuming to answer?
Although data processing questions or Spark basics questions have a higher number of view counts, they require less time to receive an answer (median is one hour). On the other hand, performance and configuration questions take

☆.

more time to get answers (median is eight and 14 h, respectively), while are still very popular in terms of view count. We also find that the view-count ranking of the question type remains stable even if we consider the age of the question.

RQ3: What are the root causes of Spark-related questions? By manually analyzing every question and answer in these 1000 posts, we summarize 11 root causes. We find that the Spark data abstraction and the lack of knowledge of API usage contribute to the root cause of half of the ques-tions (50%). Furthermore, complex environment configura-tions and incorrect or incompatible data format issues also contribute to many problems during Spark development (24%).

Our research uncovers the types of common problems in Spark development. In addition, we discuss the implications of our findings and provide actionable suggestions to practitioners. Our findings may inspire future research to better assist developers with big data development.

Paper Organization. Section 2 discusses the background and related work on Spark. Section 3 presents the common challenges in the development of Spark applications, their popularity and difficulty, and their root causes. Section 4 discusses the implica-tions of our findings. Section 5 discusses the threats to validity. Section 6 concludes the paper.

2. Background and related work

In this section, we introduce the background of Apache Spark and Stack Overflow, which we use to analyze Spark related ques-tions. Then, we discuss related work in the empirical study and testing of big data applications, and Stack Overflow.

2.1. Background

The Spark Ecosystem. Apache Spark is a distributed computing framework that executes the computation in parallel in a cluster. In Spark, the computation task is automatically sent from the driver node to the worker nodes so that the worker nodes can perform the computation in parallel. Spark is widely used for big data processing by large corporations, such as Amazon (Amazon, 2021) and Yahoo (Yahoo, 2021). Spark becomes one of the most popular big data frameworks and is the number one big data tech-nology that IT decision makers plan to deploy (Agarwal, 2019), and it has quickly become the largest open-source community in big data, with over 1600 contributors and 30.4k stars. Spark sup-ports various programming languages, such as Java, Scala, Python, and R, and provides three abstracted data structures for dis-tributed data processing: resilient distributed dataset (i.e., RDD), DataSet, and DataFrame. By using such an abstraction, developers would be free from the burden of implementing the underlying details on distributed computation. The three data structures are similar in functionalities, although DataSet and DataFrame have become the recommended data structure and API to use since Spark 2.0.

Since Spark is often used in distributed settings, it provides support for the integration with various data sources and frame-works. For example, Spark provides APIs that allow developers to read data from other data storage systems such as HDFS and Hive. Spark's data storage APIs also help developers deal with different data formats (e.g., JSON or CSV) that are used in the external data storage systems. Although such APIs help simplifies the integration with other frameworks, there may still be issues caused by incorrect configurations or API usages. To help ease the development of various data analysis tasks, Spark supports a number of high-level APIs including Spark SQL (SQL, 2021) (i.e., access data using SQL-like domain specific language), Spark Streaming (Streaming, 2021) for data stream processing, MLlib (MLlib, 2021) for machine learning, and GraphX (GraphX, 2021) for graph processing. All these APIs have their specific design and data format requirement. Hence, if developers are not familiar with the data format and the data storage framework that a Spark application integrates with, there may be unexpected errors during runtime. Moreover, since Spark abstracts and dis-tributes the computation across many worker nodes, there may be particular challenges when using Spark to train distributed machine learning models compared to models in a single node.

In this paper, we conduct an empirical study to unveil the challenges that developers encounter when developing Spark applications. We manually study the issues that developers have and summarize the root causes of the challenges. Our findings not only help inspire future research directions on assisting de-velopers with developing big data applications but also provide development guidelines for developers.

Stack Overflow. In this paper, we study the Spark issues that developers encounter on Stack Overflow. Stack Overflow is a well-known Q&A website for developers to discuss all kinds of questions related to software development. Users can not only ask and answer questions but also vote for the most appro-priate questions and answers. Every question and answer has attributes such as view count, vote count, and favorite count. The view count represents the number of views for a question thread, which reflects the popularity of the associated ques-tion/answer (Bajaj et al., 2014; Nadi et al., 2016; Yang et al., 2016). Each question can be upvoted or downvoted by users to reflect the vote count. A higher vote count means that the question may be more applicable to the general audience. Finally, Stack Overflow allows users to *"bookmark"* a question by marking the question as *"favorite"*. In this paper, we conduct a detailed manual

analysis on 1000 randomly sampled Spark-related questions on Stack Overflow. We also analyze the above-mentioned attributes to study which challenges are more prevalent and may require support from the research community.

2.2. Related work

Empirical Studies on Big Data Applications. There are several prior studies that aim to understand the challenges that developers have when developing big data applications. Bagherzadeh and Khatchadourian (2019) used Stack Overflow data to study what kinds of questions big data developers ask. They use topic models to find the more popular topics. However, their work focuses only on the quantitative aspects and does not provide an in-depth analysis of the challenges that developers have. Jiménez Rodríguez et al. (2018) used a LDA model to study the topics of the Spark-related questions that developers ask on Stack Overflow. They found the main libraries and topics of discussion about Spark and how the discussion of different Spark libraries change over time. While the study provides a good general overview of the challenges that Spark developers encounter, the study is totally quantitative and the categories in their study are coarse-grained, it offers few insights into the types of information that can be useful for building and debugging Spark applications. For example, compared to other Spark libraries, they found that Spark-SQL may have more problems but they did not discuss what challenge developer encounter when using Spark-SQL to do data processing tasks. Kim et al. (2018) surveyed 793 Microsoft data scientists on the common challenges that they encounter. They find that the most common challenges are related to data quality and the scale of data. Fisher et al. (2012) also interviewed 16 data analysts at Microsoft and they found that debugging in a distributed cloud environment is extremely challenging. Zhou et al. (2015) analyzed 210 issue reports from one of Microsoft's big data platforms. They find that more than 30% of the issues are related to application design and code logic.

Compared to prior studies, this paper aims to provide a more comprehensive and in-depth understanding of the problems that developers encounter and their root causes. We conduct a manual study on 1000 Spark-related questions collected from Stack Overflow. We manually categorize the questions and identify the prevalent question types (e.g., in terms of view counts). Finally, we manually derive the root causes of questions and summarize the implications of our findings.

Analysis and Testing of Big Data Applications. Gulzar et al. (2016, 2018, 2019) developed a series of techniques to assist developers in debugging and testing big data applications. Their work of BigDebug (Gulzar et al., 2016) simulates breakpoints and watchpoints to allow interactive debugging in big data applica-tions. BigSift (Gulzar et al., 2018) applies the concept of delta debugging to help developers identify the root cause of an error in Spark applications. Finally, BigTest (Gulzar et al., 2019) auto-matically generates a small and synthetic dataset for effective and efficient testing. Zhang et al. (2020) propose a novel coverage-guided fuzz testing tool for Spark applications. Wang et al. (2021) propose an approach called DPLOG to provide logging support to monitor data applications. These studies provide valuable support to developers and assist in testing and debugging Spark applications. Our work focuses on identifying problems and their root causes in developing Spark applications. Our findings provide directions for future research on improving the quality of Spark applications.

Other Empirical Studies that Leverage Stack Overflow. Stack Overflow is a widely used platform to study software engineering practices from developers' perspectives. Previous studies provide valuable insights by analyzing questions on Stack Overflow (Tahir et al., 2018; Ahmed and Bagherzadeh, 2018; Abdellatif et al., 2020; Tahaei et al., 2020; Islam et al., 2020; Meng et al., 2018; Islam et al., 2019). Meng et al. (2018) studied 503 Stack Over-flow posts to understand developers' concerns on Java secure coding. Islam et al. (2019) study over 2000 posts from Stack Overflow to find the common types and root causes of bugs in five popular deep learning libraries. Islam et al. (2020) studied 415 repairs from Stack Overflow and 555 repairs from GitHub to find the bug repair patterns and challenges in five deep learning libraries. They find that bug fix patterns of deep neural networks are different compared to traditional bug fix patterns. Zhang et al. (2019) studied obsolete answers on Stack Overflow to understand the evolution of crowdsourced knowledge. Mondal et al. (2021) studied unanswered questions on Stack Overflow to understand the difference between unanswered and answered questions. Similarly, we conduct an empirical study of Spark-related questions on Stack Overflow to understand the Spark problems that developers encounter.

Stack Overflow posts cover a wide range of topics. Barua et al. (2014) found that general programming and web-related topics are more common on Stack Overflow. To uncover the types of posts on Stack Overflow, previous studies (Beyer et al., 2020; Allamanis and Sutton, 2013) propose categories such as review, conceptual, and learning, to classify the posts. These categories provide a valuable perspective for classifying issues on Stack Overflow. Although such classification categories may be suitable for general programming-related problems, these categories are too general for categorizing the problems that developers encounter during Spark development. Spark, as a distributed data processing framework, requires certain domain-specific knowledge, such as data processing, distributed computing, and configuration management. Such knowledge often does not apply to general programming or other domain-specific (e.g., Android or web development) questions. Therefore, we propose new classification schemes to study the Spark-related problems that developers ask on Stack Overflow.

3. Study results

In this section, we first discuss our data collection process. Then, we discuss the results of our research questions (RQs). For each RQ, we discuss the motivation, approach, and results.

3.1. Study setup

Our goal is to understand common challenges that developers encounter when using Spark and to provide insights towards potential solutions to Spark challenges. To achieve the goal, we analyze the questions that developers ask on Stack Overflow — the most popular Q&A website for software development and programming questions. We collect the Stack Overflow data from Stack Exchange Data Explorer.[1] This website provides a SQL query interface where we can download Spark-related posts through SQL queries. Our collected data contains post information, such as the detailed content of each post (i.e., a question and its associated answers), the number of received votes in a post, and the view count of a question.

To study the types of questions that developers ask on Stack Overflow, our first step is to identify the posts that are related to Apache Spark. On Stack Overflow, developers are required to label at least one and at most five tags when they ask a question. These tags represent the specific topics for a question. Therefore, we use the tag *apache-spark* to select all Spark-related questions on Stack Overflow. Moreover, we follow prior studies (Wang et al.,

[1] https://data.stackexchange.com/help.

2018; Ponzanelli et al., 2014) to further select questions that have a score greater than zero. In total, we collected 14,043 questions and their associated 22,329 answers. To study the questions and their associated root causes in detail, we only consider the questions with an accepted answer and code snippets. Finally, we select the questions that were asked between 2014 and 2019. We choose the questions in this period because Spark 1.0 (the first stable version of Spark) was released in 2014 (although Spark was first open-sourced in 2010). Based on the above-mentioned crite-ria, we filter our extracted posts and collect 12,217 Spark-related posts.

3.2. RQ1: What types of questions do developers ask about Spark on Stack Overflow?

Motivation. Spark is a popular distributed big data processing framework that is widely used by many large companies around the world,[2] such as Shopify, Baidu, and TripAdvisor. One of the main advantages of using Spark is that it abstracts the data parallelization and computation for developers, which significantly reduces the software development overhead. Spark abstracts complex data computation using a functional programming model, e.g., executing operations such as filter, map, and reduce in dis-tributed and parallel settings. However, due to the peculiarity of Spark and its abstraction, there may be unique challenges in using Spark from developers' perspective. To help developers and practitioners understand the common issues when using Spark and to inspire future research, in this RQ, we analyze the Spark-related questions that developers ask on Stack Overflow.

Approach. As discussed in Section 3.1, we collected a total of 12,217 Spark-related posts (i.e., questions and their associated an-swers) that have an accepted answer and code snippet. To answer this RQ, we randomly sampled 1000 posts and conduct a qualita-tive study. The size of the random sample achieves a confidence level of 95% with a confidence interval of 3%.[3] We then study each sampled post based on the question itself and the associated answers (including both text and code snippets) to identify the challenges that developers have when using Spark. We did not use the question tags to classify the questions because the tags are user-provided and only contain high-level information. More concretely, we followed previous research process (Wang et al., 2018; Zhang et al., 2019) to manually derive categories. This process involves three phases and is performed by the first two authors (i.e., A1–A2) of this paper:

- Phase I: A1 & A2 first independently go through 300 randomly sampled questions and their corresponding comments, and suggested type for each post. A1 & A2 discussed the suggested types and merged the similar types. They discuss their opinions to unify the classification criteria, and finally generate 11 manually-derived types (shown in Table 1).
- Phase II: A1 & A2 independently categorize the question types for the remaining 700 questions using the types derived in Phase I. We assign each question to the most relevant category. Each question belongs to only one of these 11 types. A1 & A2 took notes regarding the deficiency or ambiguity of the labels for these questions.
- Phase III: A1 & A2 discussed the coding results that were obtained in Phase II to resolve any disagreements until a consensus was reached. The inter-rater agreement of this coding process has a Cohen's kappa of 0.825 (measured before starting Phase III), which indicates that the agreement level is substantial (McHugh, 2012).

Results. Table 1 shows the manually derived question types. In total, we uncover 11 types of questions that developers en-counter. Below, we discuss the uncovered types in detail. To the replication of our results, we have made the dataset publicly available[4 5]

Data Processing (43%) is the most prevalent issue that developers have. Developers often face issues related to data processing during Spark development. As a distributed big data processing framework, Spark provides users with multiple ways to process data. However, due to the vast number of APIs and approaches to process data, developers often cannot quickly find the most efficient way to process data in different situations. For example, a developer on Stack Overflow attempted to add a numeric index to every line.[6] His initial attempt used the map function, which is one of the most common functions in Spark. However, the code resulted in incorrect output. Even though the task can be achieved using the map function, the accepted answer suggested the developer use the function zipWithIndex in Spark. By design, the zipWithIndex function automatically appends the corresponding index to the element in a resilient distributed dataset (RDD).

Developers often encounter issues when configuring Spark or its integration with other frameworks (15%). Spark has a high degree of configuration flexibility and a large number of configuration parameters.[7] Therefore, developers may encounter problems when configuring Spark. In addition, as a distributed big data processing framework, Spark often needs to be configured in a cluster setting and communicates with numerous other big data frameworks. We also find that developers encounter config-uration issues related to the environment setup. For example, a developer attempted to save data to Elasticsearch using Spark, but he encountered a NoClassDefFoundError exception.[8] Although the developer tried to use Maven to manage the dependencies, the exception still occurred. In the end, an answer pointed out that, due to the distributed nature of Spark, the initial Maven setting that the developer used only applied to the driver machine but not the worker nodes. The developer needed to configure Maven to pass the dependent classes to both the worker nodes and the driver. ***Although less common, developers may also encounter issues related to configuring logging frameworks in Spark (1%)***. For example, a developer had an issue with configur-ing custom Spark logging for debugging due to the complexity of distributed systems.[9] Since logs are an important source of information for debugging large-scale systems, properly configuring Spark logging is important to assist developers with diagnosing runtime issues.

11% of the issues are related to data input and output (IO). Spark provides a series of APIs to access a variety of data sources in different formats (e.g., Parquet, JSON, and ORC) or frame-works (e.g., Elasticsearch and Hive). However, developers may encounter problems with reading and writing the data. For example, a developer tried to read some JSON data in Spark but got an error message after loading the entire 6 GB of the JSON file.[10] The reason is that Spark requires the JSON file to be in a specific format (i.e., not the regular JSON format), where each line in the file must contain a separate and self-contained valid JSON object. As a consequence, reading a regular multi-line JSON file

Table 1
Our manual classification of Spark-related posts on Stack Overflow. Percentages in the table are rounded up.

Type	Definition	Number of posts
Data processing	Spark provides a variety of different APIs and data abstraction formats to process data, such as RDD, DataFrame, and Dataset. Developers may encounter issues when they need to transform and process the data to get the desired data format.	432 (43%)
Configuration	Developers may encounter issues related to tuning the vast number of configurations in Spark, and the required configurations when integrating Spark with other frameworks.	151 (15%)
Input and Output	Spark provides a series of APIs to access a variety of data sources. Developers may have issues when they read or write data in various formats (e.g., JSON) or sources (e.g., NoSQL database) when using Spark.	114 (11%)
Spark basics	Developers may encounter issues with basic Spark usage and concepts related to big data development.	84 (8%)
MLlib	Developers may encounter issues when using MLlib, which is Spark's machine learning (ML) library.	47 (5%)
Performance	Developers may encounter performance issues when using Spark. The issues may be related to the resources in a cluster, e.g., CPU, network bandwidth, or memory.	47 (5%)
Streaming	Spark Streaming is an extension of the core Spark API that enables the processing of live data streams. Developers may encounter issues related to using Spark Streaming.	44 (4%)
Serialization	Spark requires objects to be serializable to be sent to worker nodes for computation. Some developers may encounter issues with object serialization when using Spark.	19 (2%)
Spark Bug	Developers may encounter unresolved bugs in Spark.	16 (2%)
Logging	Developers may encounter issues about the usage of the Spark logging system.	8 (1%)
Other	Issues that do not belong to the above-mentioned categories, such as questions about Spark UI monitor or Scala syntax problems.	38 (4%)

would result in an error. Such issues may be difficult to detect in advance due to the vast number of IO sources that Spark supports. Moreover, due to the size of the data that developers often work with when using Spark (e.g., 6 GB in the above-mentioned example), the issues may require a long time to debug.

In addition to regular IO, developers also encounter IO issues re-lated to streaming (4%). We find that developers often use Spark together with other stream-processing frameworks (e.g., Kafka or Flume) to read/write real-time data streams. Developers may have issues, such as delays in data reception, checkpoint issues, or inconsistent data format across frameworks.

Developers sometimes have issues understanding the basics of Spark (8%). Developers encounter some problems related to the working principle of Spark and programming language, etc., which are relatively basic concepts. For example, a developer did not know if the spark-shell mode can run in a clustered environment like spark-submit mode.[11] These are two different ways to start Spark applications (e.g., one is through a shell environment) but they can both run in a cluster environment. In particular, we find that ***2% of the issues are specifically related to object serialization, where developers either did not provide any serialization or made a mistake during the serialization process.*** Although Spark provides an abstraction for big data processing, developers may be unfamiliar with the distributed computing concept behind the abstraction, thus encountering issues with how to leverage Spark for a given task in hand.

5% of the issues are related to MLlib. MLlib is a machine learning library in Spark that supports many mainstream machine learning algorithms, such as logistic regression. One key advantage of MLlib is that it utilizes Spark's distributed computing capability so that developers train/apply the ML models using multiple worker nodes. However, due to the differences between the data representation and processing in Spark and other machine frameworks (e.g., R or scikit-learn), Spark developers may encounter problems when using MLlib. For example, a developer had an question regarding the type of input data that should be used in Spark's API for Latent Dirichlet Allocation.[12] However, due to unclear API documentation and the differences in how Spark represents the input data, the developer did not know how to properly use MLlib. In short, we find that due to the data types that are introduced to abstract big data processing (e.g., RDD and dataset), developers may have difficulties when using MLlib for machine learning in Spark.

5% of the developers encounter performance issues when using Spark. Because of the in-memory nature of most Spark computations, different settings (e.g., IO read and write, clus-ter configurations, and choice of different APIs) may affect the performance of Spark applications. Therefore, Spark developers sometimes encounter performance issues. For example, some developers discussed which API achieves better performance between `reducebykey` and `groupbykey`.[13] For different programming languages (e.g., Scala or Python), due to differences in implementation principles, the two APIs may have certain perfor-mance differences. Using `reducebykey` may bring a slight perfor-mance improvement but may also lead to an increase in code complexity.

Developers sometimes encounter unresolved bugs in Spark (2%). Since the release of Spark, it has attracted much attention from the software engineering community and is under active development. Therefore, Spark developers are constantly improv-ing and adding new features to Spark. There are over 40 Spark releases over seven years from 2014 (i.e., Spark's initial release) to 2021. Such active release practices may make it difficult for users to adopt the most recent Spark version. We find that in 2%

[11]

[12]

of the studied questions, users encounter some *resolved bugs* in Spark. The bugs were already resolved in either the latest release or in the master trunk, but some users were not aware of the issue and the fix, which increases the development overhead.

4% of the problems do not belong to the above-mentioned categories and are assigned to the Other category. We find that sometimes developers discuss the design of Spark. For example, a developer discussed that the design of the Row class in Spark needs some improvement.[14] This developer thought that in order to extract a value, one has to know the exact type, which is a bad design. Another problem discussed by a developer goes against the working principle of Spark.[15] The developer asked about how to call another RDD in the map function for an RDD, which is impossible to achieve and violates the working principle of Spark. In short, we find that even though Spark provides an abstraction of big data computation for developers, developers may still encounter various issues related to the design of abstraction or the underlying working mechanisms of Spark.

> We identify 11 different types of questions that developers encounter in developing Spark applications. Data processing is the most prevalent (43%) issue that developers encounter. Developers also encounter issues in configuring Spark or in the integration between Spark and other frameworks. Furthermore, developers encounter other diverse types of Spark challenges, such as performance issues, MLlib-related issues, and Spark bugs.

3.3. RQ2: Which types of questions have higher view counts and are more time-consuming to answer?

Motivation. In our manual analysis, we identified 11 categories of questions that developers encounter when developing Spark applications. In this RQ, we further investigate the view counts of the questions, which helps identify the questions that are most commonly encountered by users. We also analyze the time to receive the first accepted answer to questions. These metrics have been used in prior studies (Yang et al., 2016; Rosen and Shihab, 2015) as proxies to measure the popularity and difficulty of the questions in different categories. The finding may help identify which types of questions may be more popular and challeng-ing to solve, and thus require more support from the research community.

Approach. For the 1000 posts that we manually classified, we follow the steps below to compute the view counts and the time it takes to receive the first accepted answer.

- **View Counts**. Similar to prior studies (Bajaj et al., 2014; Nadi et al., 2016; Yang et al., 2016), we use view counts as a proxy to measure the popularity of the uncovered cate-gories of questions. In particular, we also collect some other metrics (such as the number of favorites and answer score). While there may be some subtle differences among all these different metrics, based on our observation, the correlation between view counts and other metrics is high (from 0.71 to 0.90). Moreover, we find that for the ranking of the types of questions, these metrics have similar trends compared to view counts. Therefore, we decided to use view counts as a proxy for popularity since we believe view counts are easier to interpret. We compute the normalized view count based on the age of the question by dividing view count by the year difference between the posting time and 2020 (our data was collected around the end of 2019). For example, for the questions that were posted in 2018, we divide the view count by two. Intuitively, a topic with a larger view count may be more popular.
- **Time to Receive an Accepted Answer.** We follow prior work (Yang et al., 2016; Rosen and Shihab, 2015) and use the needed time for a question to receive an accepted answer as a proxy for the difficulty of the questions in each category. When it takes a longer time for a question to receive an accepted answer, it may indicate that the question may be more difficult to answer or less common. We calculate the median time in hours for the questions that belong to the same category.

Table 2
The average of normalized (i.e., based on the age of the question) and raw view count, and the median time to receive an accepted answer in each category.

Classification	Average of normalized view count	Average of raw view count	Median hours to receive an answer
Spark basics	1361.0	5384.9	1.0
Data processing	1112.9	4213.6	1.0
Input and output	928.3	3710.2	4.2
Configuration	862.4	3560.1	14.4
Performance	698.6	2328.8	8.6
Other	605.4	2048.7	5.2
Streaming	377.7	1291.0	2.6
Spark bug	364.1	1307.0	30
MLlib	361.9	1385.0	4.7
Serialization	271.9	1120.3	3.4

Results. Table 2 shows the average of normalized and raw view count for the questions that belong to each category. Table 2 also sorts the categories by the average of normalized view counts. Note that we exclude one category (i.e., *Logging*) since this category has less than 10 questions and the data is highly skewed. We find that the ranking based on the normalized view counts is very similar as the ranking based on the raw view counts. The only difference is that the rankings of streaming and MLlib have swapped. The finding shows that the popularity of the question categories remain stable across the years. We find that questions related to *Spark Basics* have much larger normalized view counts (an average of 1361 views) compared to questions in other categories. We conjecture that the reason may be related to the multi-paradigm nature of Spark (i.e., Spark provides APIs in functional programming paradigm but can also be used in object-oriented languages such as Java) and its abstraction of complex distributed data processing. We find that questions related to *Data Processing* are the second most-viewed among all categories. The reason may be that most developers rely on Spark to process large-scale data, but as we found in our manual study, some developers may have difficulties knowing how to correctly use Spark to transform/process the data to the desired format. We also find that questions in the categories *Input and Output* and *Configuration* have a higher normalized view count (i.e., more than 800), which may indicate that developers encounter such problems more often. In general, developers not only ask more questions about *Data Processing* and *Input and Output*, but they also read related questions frequently.

Table 2 shows the median time to receive an accepted answer for the questions in each category. We find that, even though there are many questions related to *Spark Basics* and *Data Process-ing*, the median time for these questions to receive an accepted answer is short (i.e., within one hour). In other words, most of the issues that developers have in these two categories may be less difficult to answer, thus leading to a fast accepted answer.

On the other hand, we find that questions related to *Spark Bugs*, *Configuration*, and *Performance* require relatively more time to receive an accepted answer (the medians are 30.0, 14.4, and 8.6 h, respectively). Our findings show that whenever developers encounter an issue caused by bugs in Spark, they need to wait for a long time to receive an answer, which may indicate that developer will have to search other Q&A databases to find what they need. In addition, due to the complexity of systems based on Spark, developers may encounter issues on Spark configurations or configurations related to the integration with other frame-works. Finally, performance issues also require a longer time to resolve. For example, a developer used the window function to transform a dataframe, the developer was looking for a more efficient way to process his data. It took almost a month to get an accepted answer.[16] As we observed in our manual study, the reason may be that performance issues are difficult to diagnose in a distributed computing environment, especially when the computation involves complex data computation across multiple worker nodes.

In short, our finding highlights the potential bottlenecks and challenges that developers may encounter when using Spark for big data applications. Future studies may consider helping developers with Spark development by providing better diagnostic support and suggestion for configuration and performance related issues. Future studies may also help developers identify if the issue that they encounter is related to bugs in Spark (e.g., by automatically mining the bug reports), thus reducing the needed time to fix bugs by requesting help from Stack Overflow.

> Questions related to Configuration and Performance receive a relatively large number of average view counts and require the most time to answer. On the other hand, Spark Basics and Data Processing questions have the most view counts but are faster for developers to solve. Future studies may consider helping developers with Configu-ration and Performance related issues due to their long answering time.

3.4. RQ3: What are the root causes of Spark-related questions?

Motivation. In the previous RQs, we study the types of questions that developers encounter and how popular/time-consuming they are. However, questions of the same type can have different root causes, and different questions can share the same root cause. Hence, it is important to understand the root causes of Spark questions and provide insights into the challenges that developers have. In this RQ, we manually study and identify the root causes (e.g., incorrect API usage) of the studied Spark questions. Our findings can help practitioners avoid common issues and inspire future research directions to better support developers in improving the quality of Spark applications.

Approach. We manually analyze the same 1000 sampled posts that we studied in RQ1, with a goal to identify the root causes of each Spark question. We follow a similar open-coding pro-cess in RQ1, where two authors first study the questions and identify their root causes separately based on 300 randomly selected questions. As it is not possible for us to find and ask the questioner of each post about why this problem is caused, we speculate about the cause of their problems based on their questions, the accepted answers and the code that solves the problem. We generate a list of root causes after discussions and label the rest of the questions with our derived list of root causes. These root causes are related to developers' usage of the Spark API/configuration or Spark's abstraction. After labeling all the sampled questions, we calculate the distribution of the questions with different root causes in each question type. We filter out the question types with fewer than 20 posts, which are Serialization, Logging, Spark Bug, and the posts that belong to the Other category. We remove these posts since they are often unrelated (e.g., the ones that belong to the Other category) or are very similar (e.g., the ones that are related to Serialization). Thus, we wish to focus on identifying the root causes of the major question types.

Results. Table 3 shows our manually identified root causes of the studied Spark questions together with their corresponding percentage. In total, we identify 11 root causes about why developers encounter the question. Below, we discuss the different root causes in detail.

The root cause of 28% of the studied problems is related to Spark's data abstraction since developers are not able to see the intermediate results in data processing pipelines. Spark uses various data structures (e.g., RDD and DataFrame) to abstract distributed computation and performance optimization. In a data processing pipeline (i.e., calling a chained list of meth-ods), Spark automatically distributes the computation across the worker nodes and applies lazy evaluation for optimization. In other words, Spark would not compute the result immediately when a method is called. Instead, Spark may optimize the perfor-mance of data processing by combining several method calls into one data transformation operation. Developers may call several data transformation methods in a data processing pipeline, but developers do not know how the data is transformed/processed in each step. When developers get an unexpected result, they may not know which step in the data processing pipeline causes the issue. Debugging such data processing errors can be chal-lenging since there is no automated way of tracking how the data is transformed/processed in each step of the data process-ing pipeline. Moreover, breaking the data processing pipeline to record intermediate results will result in significant performance degradation. For example, a developer on Stack Overflow had an issue with his big data application.[17] The application always threw an exception after running for half an hour, but there was no information to show which step in the data processing pipeline caused the issue. Therefore, the developer wished to display step-by-step execution results to debug the application. Since there was no readily available solution, the suggested an-swer was to sample a subset of the data and test the application locally instead of on the cluster.

> Spark's abstraction for data processing and lazy computation causes challenges for developers to understand the intermediate data processing steps during debugging and monitoring.

The root cause of 22% of the studied problems is related to insufficient knowledge of API usage. In order for developers to gain flexibility in working with various data types, Spark provides a rich set of API functions with various configurations/options to support development. Developers can choose different API functions when working with diverse data types. However, due to a large number of API functions in Spark, it is difficult for developers to use the proper functions that apply to specific development scenarios. Developers can encounter problems by using incorrect API functions or inefficient API functions when a more performant alternative exists. For example, a developer

Table 3
Root causes of the 1000 manually studied Spark-related questions on Stack Overflow. Percentages in the table are rounded up.

Root cause	Definition	Number of questions
Not able to know the intermediate processed data due to Spark's abstraction.	Due to Spark's lazing evaluation and the nature of its data processing pipeline, developers cannot track/view how the data is transformed in the pipeline. Developers often encounter problems of not knowing how the data is transformed in each intermediate step, which can make it difficult for users to debug unexpected results, especially those cause by intermediate data processing functions.	255 (28%)
Insufficient knowledge of API usage	Spark provides a wealth of APIs to deal with various development scenarios. However, with the evolution of Spark, new APIs are constantly introduced and some of the existing APIs may also have changed significantly over time. Developers may use an outdated API that is no longer compatible with a newer version of Spark. We observe that API misuse is the root cause for a variety of questions, including data processing, performance, streaming, input/output, and MLlib.	206 (22%)
Complex environmental configuration	In order for Spark applications to run successfully, developers need to configure Spark's distributed environment and the environment of its interacting components. The configuration of a distributed environment can become complex as a Spark application is developed at a large scale. The complexity of distributed environment configuration is one of the root causes for the studied questions.	116 (13%)
Incorrect or incompatible data format	When using Spark, developers often need to convert different data types between various formats and representations, e.g., converting RDD to DataFrame, or converting input data to become compatible with MLlib. When developers are not familiar with how to convert different data types, they may encounter unexpected issues.	100 (11%)
Lack of basic knowledge of Spark or programming language	Spark may have a steep learning curve. Developers need to master some programming languages and have certain understanding of distributed systems. In some questions, the root cause of the issue is that some developers lack the basic knowledge of Spark or related concepts.	84 (9%)
Integration errors with other data sources	Spark can read data from other big data frameworks such as HDFS, Flume, Kafka, etc. Spark's input/output and streaming often involve framework integration. Developers encounter problems related to IO and streaming due to errors during framework integration.	51 (6%)
Incorrect or suboptimal configuration values	Due to the diverse configuration parameters during Spark development, developers may use suboptimal or even incorrect configuration values in their Spark applications.	45 (5%)
Diverse representation of timestamp data	Parsing and analyzing timestamps in the data due to their diverse representation (e.g., for a year-month-day timestamp only the year is needed) analyzing timestamps in the data. Many developers are unclear how to deal with data containing timestamps, such as changing the time unit.	35 (4%)
Data distributed computing	In Spark, the data is processed in a distributed manner. Developers may encounter performance issues caused by their unfamiliarity with how a computation task is distributed across worker nodes. Therefore, developers may not know how to configure the distribution system settings and process data in a distributed environment. (e.g., data partitions are too small), and result in poor performance.	11 (1%)
Lack of knowledge on memory tuning	Some developers may develop the application without making memory optimization and caused insufficient memory issues in the application.	10 (1%)
Unfamiliar with the working principle of distributed algorithms.	Some developers encounter problems during the MLlib development process because they are not clear about how to make the algorithm suitable for parallel computing, which causes them to encounter problems during the model training process.	6 (1%)

(i.e., an asker) on Stack Overflow asked how to determine if a dataframe contains a specific string and how to extract only the rows that contain the string. The asker initially attempted to initialize a user-defined function (UDF) and executed the function along with Spark's API. Nevertheless, the final result was differ-ent from what the asker expected.[18] To solve the problem, an answerer created a user-defined function (x_isin_array), while a commenter suggested a better solution that reuses a Spark built-in API function (array_contains) that the answerer was not aware of.

We also find that the names of some Spark functions may be similar to other popular data processing frameworks, but the usage can be different. For example, a developer encounters a problem when he migrated Pandas Dataframe code to Spark Dataframe.[19] While developers know how to use groupby to process data in Pandas, they encountered problems using groupby in Spark due to subtle differences between the two frameworks. In short, developers may assume that the API in Spark is the same as the one in Pandas, which results in issues in the application.

> Developers may not be familiar with some Spark APIs or may use them in an incorrect/inefficient way. Further support is needed to provide better API recommendations for various data processing tasks.

18

19

The root cause of 13% of the studied problems is related to complex environmental and inter-framework configuration. Since Spark applications are often executed in a distributed environment, developers need to properly configure the environment parameters to ensure the proper operation of the applications. However, correctly configuring the environment parameters can be a complex task, especially when developers need to configure both Spark and other interacting frameworks. This complexity may lead to configuration-related problems.

We also find that, since the issue may be related to the configuration of various frameworks/components in a distributed environment, the error message may not be detected in local mode and propagated back to the main Spark application. Namely, Spark application developers may be left in the dark when di-agnosing complex configuration issues in distributed environ-ments. For example, a developer tried to access a NoSQL database (i.e., Cassandra) in a Spark application using Spark's Cassandra-connector configured using Maven.[20] The connection was estab-lished, but no result was returned and there was no exception message in the local mode and driver node . The developer later found out that the issue was related to file path issues in the con-figuration file. Due to Spark's distributed nature, helping develop-ers analyze issues that are propagated across frameworks/nodes is important for debugging and ensuring application quality.

> Developers often encounter issues caused by configurations related to distributed settings or framework integrations. However, such issues are difficult to debug because error messages may not be propagated across frameworks or worker nodes.

11% of the studied problems are related to incorrect or incompatible data format. Spark provides different data formats (e.g., RDD and DataFrame) to abstract distributed computation for various data types. Developers may need to convert between data formats and types to leverage different APIs (e.g., to use APIs in MLlib, the machine learning library in Spark). However, data format conversions can be a challenging task for developers, espe-cially if the conversion involves multiple frameworks. For exam-ple, a developer asked how to convert a Spark's DataFrame with an array of doubles to a vector to pass the vector to a machine learning algorithm.[21] The data conversion process requires con-verting from Double in Scala to Vectors in Spark MLlib (i.e., across frameworks or libraries), where there is no standard nor an easy way for such conversation.[22] Future research is needed to assist developers with data format conversation, especially conversation across frameworks or programming languages.

> Developers often need to convert between data formats when using Spark. However, due to a rich variety of data formats from different frameworks or programming languages, developers may encounter challenges when converting between data formats.

9% of the studied problems are related to a lack of basic knowledge of Spark or programming language. Developers with different experience levels may try to use Spark for big data processing. Some developers encounter problems because they lack the basic knowledge of Spark or related programming languages. For example, a developer was not familiar with the Scala language and he did not understand the meaning of *rdd.map(_.swap)*.[23] He mentioned that he studied the Scala/Spark API but still could not find where *.swap* is defined, even though *.swap* is a method defined in Scala's Tuples. We found that these types of questions also have very high view counts on Stack Overflow (e.g., over 7K views in the above-mentioned example). The Spark community can improve the documentation to guide developers in understanding the basic knowledge of Spark. For example, tools that help developers extract solutions from the Spark official documentation would be helpful. A community mentoring program can also be launched to help developers build high-quality Spark applications.

> Developers may lack basic knowledge of Spark or re-lated programming languages. Better community sup-port is needed to guide developers in developing Spark applications.

Developers may encounter issues related to integrating Spark applications with other frameworks or data sources (6%). De-velopers commonly use and integrate a variety of data man-agement/storage systems, including HDFS, Hive, and Kafka, with Spark applications. For flexibility and ease of development, Spark provides a set of APIs that provides an abstraction to the data management systems. Namely, developers can use the same set of APIs to access different data storage systems. However, even though Spark provides an abstraction layer for the data storage systems, there are still some differences among the systems that Spark fails to abstract. For instance, a Spark developer needed to call different APIs when accessing files of different formats that are stored in HDFS.[24] The issue was the developer was not aware of the Spark API that handles a specific type of data that is stored in HDFS. The problem can be more challenging if the data type is not supported by Spark. In such cases, developers may need to manually handle the integration with other frameworks, which may be prone to maintenance issues if the frameworks are updated. Future studies should consider helping developers with better integration between Spark and other frameworks by providing support such as automated API recommendation or better data abstraction.

> While developers may know how to interact with some frameworks and data sources, a certain amount of variability (e.g., accessing data of different formats stored in an external framework) can cause problems when developing Spark applications.

The root cause of 5% of the studied problems is related to incorrect or suboptimal configuration values. Spark configuration parameters control the application runtime settings, such as memory limits and network bandwidth. Most of the param-eters that control internal settings have default values, which can be changed by developers. However, with hundreds of avail-able configuration parameters in Spark, it can be challenging for developers to configure the parameters according to the application and its environment setup. For example, a developer found there was much unused memory in his application and attempted to leverage the available memory by increasing the memory limit.[25] The developer changed the Spark configura-tion parameter, spark_worker_memory in the configuration file, but he found that the used memory limit stays the same. The

20 https://stackoverflow.com/questions/42219747/.
21 https://stackoverflow.com/questions/47543747/.
22 https://stackoverflow.com/questions/42138482/.
23 https://stackoverflow.com/questions/34670957/.
24 https://stackoverflow.com/questions/51994323/.
25 https://stackoverflow.com/questions/24242060/.

reason is that in order to increase the memory, the developer should modify the spark.executor.memory parameter rather than spark_worker_memory parameter. Future research is needed to assist developers in better managing Spark configurations, espe-cially in a distributed environment.

> Although Spark provides developers with a variety of configuration options, developers often encounter configuration problems due to the inability to find the correct or optimal configuration parameters.

4% of the studied problems are related to the diversity of timestamps representations. Timestamps have many different representations, such as year and month format (yyyy-MM-dd) or Unix time. Developers often need to do some processing on the timestamps in the original data based on their needs. Spark provides some API functions to help developers process timestamp data. However, the support from Spark is limited and does not cover the wide variety of timestamp formats. For example, Spark provides APIs to calculate timestamp differences in days but not for other granular levels (e.g., minutes or seconds). A developer knew that Spark provides the datediff function to get the number of days between two timestamps but he did not know how to get the minute difference.[26] The accepted answer is to convert the different times into the long data type and then manually calculate the difference in minutes. Due to the variety of timestamp representations, we found that many developers encounter issues in converting timestamp formats or calculating time differences. Thus, more support is needed to help developers avoid mistakes and assist in processing timestamp data.

> Due to different representations of timestamps and the uniqueness of the time difference calculation, developers can encounter problems when manually processing data that contains timestamps.

1% of the studied problems are related to the specifics of distributed data computing. Unlike stand-alone computing, Spark acts as a distributed data processing engine, allowing developers to maximize the advantages offered by distributed computing. Developers should consider how to maximize the use of limited resources to process data concurrently when running their applications. Spark automatically sets the number of ‘‘map’’ tasks to run on each file according to its size (though it can be controlled through setting optional parameters in SparkContext.textFile). For distributed ‘‘reduce’’ operations, Spark uses the largest specified number of partitions. Clusters will not be fully utilized unless developers set the proper level of parallelism for each operation. Some developers ignored the mechanism of distributed data com-puting and set an unreasonable partition number. For example, a developer needed to read all the images into memory as RDD, and the target image was saved in a directory hosted on HDFS.[27] This developer found that the process is time-consuming and attempted to understand if there was any better way to load large image data set into Spark. An answerer suspected that it may because there were a large number of small files on HDFS that cause the problem. A large number of small files causes Spark to generate too many tasks and affects performance. This answerer gave a suggestion which is to set the number of partitions to a reasonable number: at least 2x the number of cores in the cluster. This can increase the degree of parallelism and make full use of cluster resources to process data. Future research is needed to provide automated support to help developers utilize the computing resources in clusters.

> Developers may not utilize the resources in a distributed environment, which can cause performance issues. Future research is needed to help developers automatically configure Spark applications to utilize the cluster settings and computing resources.

1% of the studied problems are related to a lack of knowledge in Spark’s memory allocation. There are two main types of memory in Spark: execution and storage. The execution memory is mainly used to store the temporary data when running a data processing task. The storage memory is mainly used to store the cached data and broadcast variable values across nodes in the cluster. The allocation of the memories has a significant impact on the performance of Spark programs. However, developers may lack knowledge in Spark’s memory allocation, which generally requires a certain level of understanding of the inner workings of Spark. For example, a developer wished to know why the cached memory was improved when using order by in a Spark SQL query.[28] An answerer explained that Spark SQL scans the required columns and automatically optimizes memory usage through data compression. Therefore, it is important for developers to know how to choose a proper data structure and storage strategy to utilize memory usage in Spark applications.

> Different data processing tasks may have different memory allocation and optimization strategies. Therefore, developers may need to know the inner workings of Spark to utilize the memory allocation in different data processing tasks.

The root cause of 1% of the studied problems is related to being unfamiliar with the working principles of distributed algorithms. We find that in some cases, developers encounter challenges in the development process because they do not understand the distributed algorithms used in Spark for machine learning operations. For example, a developer got different results for multiple runs on the same input matrix when he computed SVD (Singular Value Decomposition).[29] An answer explained that even though the SVD computation should be deterministic (i.e., does not rely on random numbers), the final results in Spark are sometimes different. The reason is that every change in Spark may be regarded as non-deterministic and Spark can merge partial results of upstream tasks in any order, which can help applications to get optimal performance in distributed computing. If developers are not familiar with such optimization and apply the algorithm regularly, there may be unexpected results. There may be a need to provide better documentation on helping developers know, from a high-level perspective, the difference between the distributed and non-distributed algorithms when applying machine learning models.

> Due to the distributed nature, some machine learn-ing algorithms may be optimized and have slightly different behavior. Developers need to be aware of such differences to avoid issues when developing Spark applications.

[26] .

[28] .

Table 4
The distribution of the root causes of Spark-related questions among the seven uncovered question types. Percentages in the table are rounded up.

Question type	Root cause	Number of post
Data processing	Not able to know the intermediate processed data due to Spark's abstraction.	243 (56%)
	Insufficient knowledge of API usage	103 (24%)
	Incorrect or incompatible data format	51 (12%)
	Diversity of the representation of timestamps	35 (8%)
Spark configuration	Complex environmental configuration	106 (70%)
	Incorrect or suboptimal configuration values	45 (30%)
Input and Output	Integration errors with other data sources	39 (34%)
	Insufficient knowledge of API usage	33 (29%)
	Incorrect or incompatible data format	32 (28%)
	Complex environmental configuration	10 (9%)
MLlib	Insufficient knowledge of API usage	24 (51%)
	Incorrect or incompatible data format	17 (36%)
	Unfamiliar with the working principle of distributed algorithms	6 (13%)
Spark performance	Insufficient knowledge of API usage	26 (55%)
	Data distributed computing	11 (23%)
	Lack of knowledge about memory tunning	10 (21%)
Streaming	Insufficient knowledge of API usage	20 (46%)
	Not able to know the intermediate processed data due to Spark's abstraction.	12 (27%)
	Integration errors with other data sources	12 (27%)
Spark basics	Lack Basics knowledge of Spark or programming language	84 (100%)

4. Discussion and implications of our findings

In this section, we discuss the implications of our findings. We discuss actionable implications and future work for two groups of audiences: researchers and practitioners.

4.1. Discussion and implications for researchers

Developers often encounter issues with using or choosing the correct API in various tasks when developing Spark applica-tions. Future studies should provide better API recommen-dations and usage support. Table 4 shows the root causes of the different types of problems and their distribution. *Insufficient knowledge of API usage* is a common root cause for questions of various types, including Data Processing, Input and Output, MLlib, Spark Performance, and Streaming. As we found in RQ3, choosing the correct API can be a challenging task, especially given the complex setup of Spark applications (e.g., integration with many other frameworks). Therefore, future studies should consider proposing techniques to help developers identify the correct APIs to use based on the given data processing tasks, data types, and the used frameworks. Moreover, once the API is recommended, it would be important to provide support on choosing the correct API options/parameters for optimal usage.

Environmental and runtime configuration tuning is essential in the deployment of Spark applications. Future studies are needed to automatically help developers detect or debug configuration issues. Configuration problems are prevalent and difficult to resolve (i.e., require more time to answer, as shown in RQ2). As discussed in RQ3 and shown in Table 4, configuration issues are common and may occur when developers need to leverage other frameworks for data storage. Prior studies (Vitui and Chen, 2021; Xu et al., 2016; Chen et al., 2016) proposed approaches to tune configurations or detect configuration errors. However, prior studies often do not consider the complexity of the composition of big data applications, where the applications are integrated with other frameworks in a distributed environ-ment. Furthermore, applications based on Spark can depend on other software systems in the big data ecosystem, leading to the challenge in optimizing the configuration. Hence, an automated tool that helps developers choose and optimize the configuration of Spark applications according to the product requirements and cluster setting can effectively assist developers with application deployment.

Even though Spark provides APIs for interacting with other frameworks, developers often encounter issues related to cross-framework access such as data format conversation. Automated approaches that help developers better handle framework integration are needed. When developing Spark applications, developers often integrate the applications with other data storage frameworks such as HDFS. Therefore, accessing data of different types across frameworks is a challenging task. As shown in Table 4, developers often encounter issues related to data format conversion when reading/storing data across frameworks or system components. Although Spark provides a schema to standardize data reading/writing, developers still encounter data format-related issues. In many cases, developers need to convert data format manually, which may be error-prone and increase maintenance effort. Moreover, some APIs (e.g., MLlib) may even have specific data format requirements. Hence, ap-proaches that automatically handle data format conversation and abstraction can better help developers handle data from various sources.

Developers spend more time solving the performance issues in Spark. Future research should not only help developers automatically detect performance problems but also automatically analyze the causes of performance problems for developers. Although performance issues in Spark development are not among the major issue types, as shown in RQ2, such issues require more time to fix. In our manual study, we found that performance problems are usually solved by tuning parameter configurations or code optimization. Prior studies (Ren et al., 2018; Alnafessah and Casale, 2020) often focus on the identifica-tion and prediction of performance problems. However, once the issue is found, it may take a long time for developers to resolve performance issues. Future research is needed to help developers diagnose the root causes of performance problems in production environments and provide automated optimization suggestions based on factors such as deployment settings.

Developers often encounter issues when pre-processing data or running machine learning models in distributed settings. Future research should help developers better migrate machine learning tasks to distributed environments. The machine learning library (i.e., MLlib) is an important part of Spark. Al-though Spark aims to abstract the distributed computing de-tails, developers still encounter various issues. The model training/inference process in MLlib may be different from other traditional machine learning libraries (e.g., scikit-learn) due to Spark's distributed nature and integration with other data storage frameworks. As we found in RQ3, developers may not be aware that

some algorithms may become non-deterministic when running on Spark, which may affect the model output. Also, some APIs in MLlib may have specific data format requirement that is not needed if developers want to train the model in other frame-works. Future studies are needed to provide developers with additional information (e.g., API hints Wang et al., 2021) to high-light the data requirement and behavior of certain algorithms in distributed settings.

4.2. Implication for Spark application developers

Developers should be aware of the potential risks in using long method chains and develop good unit testing habits. We observe that developers commonly make mistakes during data transformation since the data processing is abstracted away from developers. It can be difficult for developers to visualize how a set of data is processed in each step of the data processing pipeline. Wang et al. (2021) propose an approach to log samples of the intermediate data processing results. However, developers should still avoid using long method chaining in a data process-ing pipeline (e.g., dataframe.filter().groupby().join().distinct()) for better debugging and maintenance.

Developers should carefully study the official Spark programming and API documentation before developing Spark applications. Although Spark aims to abstract the distributed data computation from developers, such abstraction is not always perfect. Some errors may occur if developers use some Spark APIs without knowing the underlying details. As we found in RQ1 and RQ3, developers often use Spark APIs incorrectly or inefficiently, assuming the API usages may be the same as their single-node counterpart (e.g., Python's Pandas library). Although the official Spark programming and API documentation already highlighted the correct usage, developers may not be aware of them. Therefore, before using a Spark API, developers should check the usage examples in the official documentation to avoid misuse and be aware of the differences with other single-node data processing frameworks such as Pandas. To support developers in developing Spark applications, in RQ3 we analyze the root causes of Spark-related questions and provide guidelines for developers to mitigate potential issues.

4.3. Implication for Spark framework developers

Spark framework developers can improve the Spark official documentation and provide more targeted tutorials based on developers' needs. During our manual study, we found that the answers for many problems on Stack Overflow can be found in the Spark official documentation. The official Spark documenta-tion is a comprehensive document with detailed explanations and descriptions of each part of Spark development. However, some Spark components may require more experience and technical skills. In our paper, we uncover the most difficult (in terms of time to receive an answer) and common issues for developers. Spark framework developers may leverage the findings and doc-ument explicitly the common issues that we find as common pitfalls to avoid. Our findings also highlight the challenges that developers have, and Spark framework developers may consider creating more targeted tutorials to help developers avoid such issues. For example, developers spend the most time on solving configuration issues. Although Spark introduces almost all con-figuration parameters in the official documentation, the Spark community can provide more explanations and descriptions of the commonly used configurations commonly and how to resolve common configuration issues.

5. Threats to validity

Internal Validity. Threats to internal validity are related to ex-periment errors or biases. In our study, we rely on studying the Stack Overflow posts to understand the problems that developers have when developing Spark applications. To ensure that the questions are representative, we chose to study the questions with a score higher than zero by following prior studies (Wang et al., 2018; Ponzanelli et al., 2014), which indicates that other de-velopers also upvoted the questions. We study the questions with accepted answers, which may inadvertently exclude a certain types questions, e.g., questions specific to the asker's environ-ment. However, without an accepted answer, we cannot know the real cause and solution to the problem. We also exclude the questions without code snippets, since we want to study the code snippets to better understand the root causes and fixes. We manually studied 1000 Spark-related posts. To reduce biases in our manual study process, two authors independently categorize the posts. Any difference is discussed until a consensus is reached. We computed the Cohen's Kappa and found the agreement value is high (0.825). To measure the popularity and difficulty of the studied Spark questions, we need to extract some representative metrics. To minimize the threat that the metrics may not be representative, we follow previous studies (Yang et al., 2016; Rosen and Shihab, 2015; Bajaj et al., 2014; Nadi et al., 2016) and use the same metrics to measure popularity and difficulty. We also normalize the metrics based on the year that a question was first posted, since elder questions may receive more votes or view counts by nature. We find that, the ranking of the categories of the questions remain stable before and after the normalization.

External Validity. Threats to external validity are related to the generalization of our results. Since there are over 12,000 Spark-related posts after the filtering process, it is manually infeasible to study all of them. Therefore, we chose to study a statistically significant sample. To increase the generalization of our sampled data, we conduct the study on 1000 randomly selected posts, re-sulting in a statistically significant sample using a 95% confidence level and 3% confidence interval. Although we have a relatively large sample, it is difficult to guarantee that our samples cover all types of problems. Some problems with very small percentage in our sample data are also hard to guarantee that there is the same percentage on the entire dataset. There may also exist a certain degree of subjectiveness and ambiguities during the manual study process. To reduce this threat, the two authors independently studied the posts and discuss the categorization result.

6. Conclusion

Due to the increased data size, developers start to leverage big data frameworks such as Apache Spark for data processing and analysis. Although Spark abstracts the underlying distributed data computation details, there may still be issues caused by the abstraction and other challenges associated with big data development. To understand the challenges that developers en-counter and help inspire future research direction, in this paper, we conduct an in-depth study on the issues that developers encounter when developing Spark applications. We sample 1000 Q&A posts from Stack Overflow and conduct a detailed man-ual analysis on each post. We classified the problems into 11 categories and found that data processing is the most common problem that developers. We also found that developers often encounter issues related to framework integration (e.g., integrate Spark applications with HDFS) and configuration. Then, we com-puted the view counts and the time it takes to receive the first accepted answer to study the popularity and difficulty of the

questions in different categories. We found while data process-ing and basic Spark questions receive more view counts, they tend to be solved within an hour. On the other hand, questions related to configuration and performance require more time to answer. Finally, we manually derived 11 root causes based on the sampled posts. We found that incorrect/inefficient API usage is a common challenge across various types of questions, including framework integration, data processing, configuration, etc. Our findings summarize the challenges that developers encounter and we discussed the implication of our findings and future research direction in assisting developers with big data development.

Spark SQL, 2021. Spark SQL. https://spark.apache.org/sql/.
Spark Streaming, 2021. Spark streaming. https://spark.apache.org/streaming/. Tahaei, M., Vaniea, K., Saphra, N., 2020. Understanding privacy-related questions on Stack Overflow. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. pp. 1–14.
Tahir, A., Yamashita, A., Licorish, S., Dietrich, J., Counsell, S., 2018. Can you tell me if it smells? a study on how developers discuss code smells and anti-patterns in stack overflow. In: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018. pp. 68–78.
Vitui, A., Chen, T.-H., 2021. MLASP: Machine learning assisted capacity planning. Empir. Softw. Eng. 26 (87).
Wang, S., Chen, T.-H.P., Hassan, A.E., 2018. How do users revise answers on technical Q&A websites? A case study on Stack Overflow. IEEE Trans. Softw. Eng..
Wang, Z., Zhang, H., Chen, T.-H.P., Wang, S., 2021. Would you like a quick peek? Providing logging support to monitor data processing in big data applications. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. In: ESEC/FSE 2021, pp. 1–11.
Xu, T., Jin, X., Huang, P., Zhou, Y., Lu, S., Jin, L., Pasupathy, S., 2016. Early detection of configuration errors to reduce failure damage. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation. In: OSDI'16, pp. 619–634.
Yahoo, 2021. Yahoo. https://www.yahoo.com/.
Yang, X.-L., Lo, D., Xia, X., Wan, Z., Sun, J.-L., 2016. What security questions do developers ask? A large-scale study of stack overflow posts. J. Comput. Sci. Tech. 31, 910–924.
Zhang, H., Wang, S., Chen, T.-H.P., Zou, Y., Hassan, A.E., 2019. An empirical study of obsolete answers on stack overflow. IEEE Trans. Softw. Eng..
Zhang, Q., Wang, J., Gulzar, M.A., Padhye, R., Kim, M., 2020. BigFuzz: Efficient fuzz testing for data analytics using framework abstraction. In: 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 722–733.
Zhou, H., Lou, J.-G., Zhang, H., Lin, H., Lin, H., Qin, T., 2015. An empirical study on quality issues of production big data platform. In: Proceedings of the 37th International Conference on Software Engineering - Volume 2. In: ICSE '15, pp. 17–26.

Zehao Wang is a Ph.D. student in the Department of Computer Science and Software Engineering at Concor-dia University, Montreal, Canada. His research interests include empirical software engineering, data analysis, and software performance. His work has been pub-lished in conferences, such as ICSE, and FSE. He served as a program shadow committee member for MSR. He obtained his B.Sc. from Xidian University and M.Sc. from Concordia University. More information at: https: //zehaowang00.github.io/.

Tse-Hsun (Peter) Chen is an Associate Professor in the Department of Computer Science and Software Engineering at Concordia University, Montreal, Canada. He leads the Software PErformance, Analysis, and Re-liability (SPEAR) Lab, which focuses on conducting research on performance engineering, program anal-ysis, log analysis, production debugging, and mining software repositories. His work has been published in flagship conferences and journals such as ICSE, FSE, TSE, EMSE, and MSR. He serves regularly as a program committee member for international conferences such as ASE, ICSE, ICSME, SANER, and ICPC, and he is a regular reviewer for software engineering journals such as EMSE and TSE. Dr. Chen obtained his B.Sc. from the University of British Columbia, and M.Sc. and Ph.D. from Queen's University. Besides his academic career, Dr. Chen also worked as a software performance engineer at BlackBerry for over four years. Early tools developed by Dr. Chen were integrated into industrial practice for ensuring the quality of large-scale enterprise systems. More information at: http://petertsehsun.github.io/.

Haoxiang Zhang is a researcher at the Centre for Software Excellence at Huawei Canada. His research in-terests include empirical software engineering, mining software repositories, and intelligent software analyt-ics. He received a Ph.D. in Computer Science from Queen's University, Canada. He received a Ph.D. in Physics and M.Sc. in Electrical Engineering from Lehigh University, and obtained his B.Sc. in Physics from the University of Science and Technology of China. Contact haoxiang.zhang@acm.org. More information at: https: //haoxianghz.github.io/.

Shaowei Wang is an assistant professor in the Depart-ment of Computer Science at University of Manitoba. He leads lead the Software Management, Maintenance, and Reliability Lab (Mamba). His research interests include software engineering, machine learning, data analytics for software engineering, automated debug-ging, and secure software development. His work has been published in flagship conferences and journals such as FSE, ASE, TSE, TOSEM, and EMSE. He serves regularly as a program committee member of interna-tional conferences in the field of software engineering, such as ICSE, ASE, ICSME, SANER, and ICPC, and he is a regular reviewer for software engineering journals such as JSS, EMSE, and TSE. He received several distinguished reviewer awards, such as the Springer EMSE (SE's highest impact journal) and MSR. He obtained his Ph.D. from Singapore Management University and his B.Sc. from Zhejiang University. More information at: https://sites.google.com/site/wswshaoweiwang/.