OPEN ACCESS
ISSN 2356-5462
http://socj.telkomuniversity.ac.id/ijoict/

Intl. Journal on ICT
Vol. 10, No. 1, Jun 2024. pp. 116-126
doi: doi.org/10.21108/ijoict.v10i1.929

The Implementation of Titian for Data Provenance on DISC Systems Automated Debugging

Agista Hidayatillah Suparno Putri[1*], Nungki Selviandro[2], Gia Septiana Wulandari[3]

[1,2,3]*School of Computing, Telkom University*
Telekomunikasi street no. 1, Bandung, West Java, Indonesia

[1*] agistaputri20@gmail.com

Abstract

Data-Intensive Scalable Computing (DISC) systems are essential for managing large datasets with an emphasis on fault tolerance, cost-effectiveness, and user accessibility. However, input errors in processed data pose significant challenges to programmers. This research uses the Snowfall Analysis program, notorious for its anomalous data leading to forecasting inaccuracies, as a case study. To address these challenges, we employ Titian, an extended library that facilitates debugging by systematically tracing the provenance of erroneous data to its origin. Our analysis demonstrated that Titian accurately identified data errors with a precision of 100%, moreover the average of program runtime which implement Titian is only 0.505 seconds across various sizes of dataset, significantly outperforming standard manual debugging methods. These findings highlight Titian's potential to enhance data provenance in DISC systems, offering theoretical insights into debugging processes and practical applications for improving data integrity in large-scale computational environments.

Keywords: Automated-debugging, DISC, Snowfall, Titian

I. INTRODUCTION

DISC (Data-Intensive Scalable Computing) is a computational system used for collecting, computing, and managing over large-scale datasets. DISC is used to process big data to generate business decisions to scientific discoveries [1]. The characteristics of a DISC system include processing a large dataset in parallel and storing it within a shared storage system. Examples of distributed programming like this can be found in frameworks such as Apache Spark, Apache Hadoop, and MapReduce.

However, in the development of DISC systems, developers often encounter programming problems ranging from input errors, unclean data, and incorrect decision making due to the system's failure to process the data [1]. It can be challenging to debug data processing in DISC systems, since the debugging process must identify the subset of data that causes failures, exceptions, or crashes [2]. The other way to do this debugging process is trial and error debugging, in which developers repeat a section of their data processing procedures on a subset of intermediate data, resulting in an outlier or incorrect output [2]. Meanwhile, debugging is a time-consuming and irritating activity that might account for a substantial portion of the cost of software maintenance [3].

Therefore, to make developers easier for developers when debugging these systems, they can use automated debugging. Automated debugging DISC system on Apache Spark framework especially, can be done using a

Received on 14 May 2024. Revised on 28 June 2024. Accepted and Published on 30 June 2024.

data provenance library called Titian. Titian can capture data provenance in DISC systems. This library also supports tracking of problematic data input that can trigger failures and errors.

Currently, there is not much research that uses the Titian library to track data provenance in Spark. The first research was conducted by Interlandi [3], who created the Titian library and tested its efficiency through a comparison with NEWT and RAMP using the Word Count and Grep case study. It was then discovered that by utilizing Titian, the runtime overhead did not exceed 30% of the basic program's runtime. The second research was conducted by Diestelkamper [4]. The outcome of this research was that a detailed comparison could not be performed because Titian did not support data provenance for nested data and some scenarios in his study. However, he provided flat data and made scenarios which involved filter and union computations, the time overhead with Titian was 5.89%, while using Pebble resulted in 6.98%.

Therefore, due to limited literature about adopting Titian to do data provenance, there is a need to investigate the adoption of Titian library in another DISC system case study. In this context the Titian library is used in the snowfall analysis program. Climatological data on snowfall is often anomalous, resulting in data gaps [5]. Modern snowfall observations with high temporal resolution are obtained from automated weather stations. However, these stations usually do not measure snow density and its temporal changes directly; instead, these parameters are often assumed or modeled [6]. Research on extreme snowfall is limited and poorly understood, especially in terms of spatial and temporal variability and drivers [7]. This gap is due to scarce station observations, lack of consistent indices for comparison, and the complex mechanisms behind extreme snowfall, particularly in High Mountain Asia (HMA) [7]. Additionally, the failure to differentiate between types of precipitation in station observations increases uncertainty in identifying snowfall events [8]. Data mistakes in snowfall analysis, particularly in snowfall data, are caused by a variety of factors, including observer errors, instrument faults, and data representation problems as a result of snowfall diversity, which is still a concern [9].

Snowfall analysis system is filtered by the failure function which defines the impossible value of snowfall. The delta snowfall which exceeds the impossible value of snowfall is categorized as a failure data. Then, the Titian will apply to track the original input that triggers failure and show it to us immediately.

As a result, this research makes a significant contribution to the field by providing a comprehensive empirical evaluation of Titian's accuracy and efficiency in comparison with manual debugging, specifically within automated debugging for snowfall analysis systems. These aspects of accuracy and efficiency have not been investigated in previous studies. By demonstrating how Titian can streamline the debugging process, our study lays the groundwork for future research to explore root cause analysis and preventive measures.

II. LITERATURE REVIEW

A. *DISC*

Data is increasing day by day, hence the need for a high-performance computing system to generate and collect giant data. The system that is now needed is one that can perform acquisition, updating, sharing, and archiving of these data to provide advanced computing capabilities to process data. DISC is the answer to that demand, because the DISC system is inspired based on the system infrastructure developed to support Internet-based services which provide high-level computing, storage, and availability, but also considering the best cost [1].

The DISC system acts as a business decision maker and scientific discovery based on the large datasets it manages to process [2]. DISC systems have been widely implemented in various companies. They use different designs of systems but in general they follow a model called warehouse-scale computers. For example, Google has implemented a DISC system as MapReduce [10] that supports powerful parallel computation over large amounts of data. MapReduce is used by Google for doing statistical computation to generate the index structures for its search engine, also they have created algorithms called PageRank for quantifying the relative importance of various Web documents [1].

B. *Apache Spark Workflow*

In order to process DISC systems, there are several computing frameworks such as Apache Spark and Apache Hadoop. These types of frameworks are created for big data analytics since they have advanced in-memory programming and high-level libraries for data processing, machine learning, graph analysis, and streaming.

Spark abstraction is formed by RDDs (Resilient Distributed Datasets) for efficient data sharing between computation processing. RDD makes Spark program have better performance since it can be designed and implemented on various workloads with modification of its scalable data algorithms and pipelines [11]. There are several transformations that RDD offers e.g., filter, map, reduce, group-by, join. Besides that, RDD also offers action e.g., count and collect. Spark transforms RDD transformations into stages, with each stage containing subsections of changes until a shuffle step is needed (data partitioning). Common stage-breaking RDD operations, such as reduceByKey, groupBy, and join, require data re-partitioning.

Spark core uses cluster manager which objective is to manage resource sharing among the program components. Then, for the ability to access datasets, Apache Spark can access data that is stored in Cassandra, Hadoop data source, HBase, HDFS, Hive, and Alluxio.

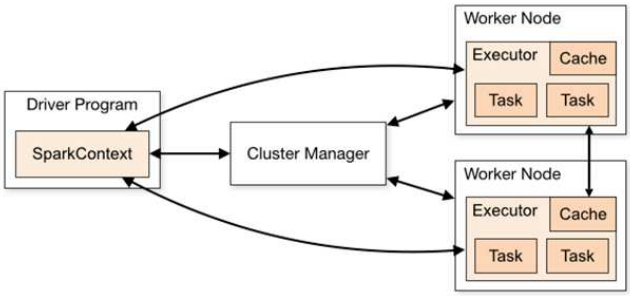

Fig. 1. Process of running Spark program

(Source: https://spark.apache.org/docs/latest/cluster-overview.html)

Figure 1 illustrates several components working on Spark program execution, which include a driver program, cluster manager, and worker node. Driver program is the process running the main function of the Spark program and creating the SparkContext. Inside of the driver program, the user should declare SparkContext to connect the driver program with the cluster manager. Once connected, Spark acquires executors on worker nodes to run tasks (units of work that will be sent to one executor) and keeps data in memory or disk storage across them. Lastly, SparkContext sends tasks to the executors to run.

C. Data Provenance in Spark

Data provenance as known as data lineage has the objective of tracking the origins of data and all processes that happened to the data starting from it arrived in a database, therefore understanding data provenance is critical to get accuracy of the data. A data item's provenance is a record of how the data item was created from other data items through a series of transformations [12].

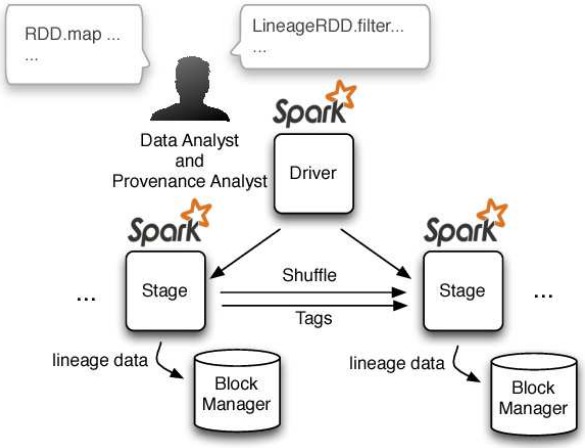

Fig. 2. Data Provenance Workflow in Spark [13]

Figure 2 demonstrates how data provenance is conducted in Spark environment. It starts on a driver program which is already set by programmer along with any transformation to dataset. Every stage data lineage will be tracked and saved in Block Manager (Spark's in-memory storage).

D. Automated Debugging using Titian Library

Titian is a Spark external library that enables data provenance. This library implements simple LineageRDD applications programming interface. LineageRDD is an extension of RDD abstraction with tracing capabilities. Titian library is able to be used in interactive Spark sessions to do exploratory data analysis, because it's incorporated into the Spark programming model and runtime [3].

To do data provenance in Spark using Titian, the first step that has to be done is create method getLineage, which is return the LineageRDD as extension of native RDD. When LineageRDD has been created, the user's program transformation will be tracked by Titian [3].

Method that enables traceback is the 'goBackAll' method, which returns all initial input. If the user only requires tracking a step backward, supported by 'goBack' method [3]. Similar to RDD abstraction, LineageRDD also has workflow by returning a new LineageRDD that corresponds to the traced location without actually assessing the trace. When a native RDD operation, such as 'count' and 'collect', is invoked, the real tracing happens.

LineageRDD also includes transformation from native RDD because LineageRDD extends it. For example, if the user needs to create native transformation, LineageRDD also returns a new native RDD that references raw data. In conclusion, the actual Spark computation can be combined with data provenance queries, therefore users can interactively design and debug the program.

III. RESEARCH METHOD

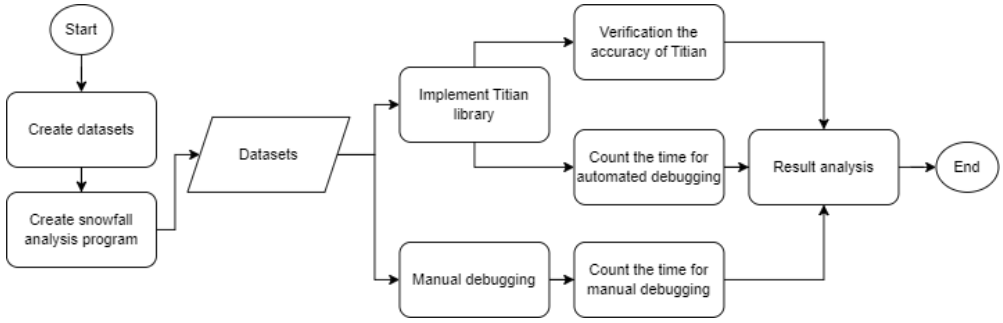

Fig. 3. Research Workflow

Figure 3 illustrates the workflow of this research. The explanation of every step in figure 3 has been detailed one by one below.

A. Create Datasets

We initially created five dummy record datasets to be implemented on the snowfall analysis program including the zip code, date, and amount of snowfall. Each line in this dataset is ordered by zip code then date. This structured dataset is sufficient to offer information on how much snowfall a location receives each day.

We developed five varied sizes of datasets to compare Titian's accuracy and efficiency on each dataset. Then we summed up each dataset to get a clear picture of Titian's accuracy and efficiency. These datasets grow bigger from first to last.

The first dataset consists of two zip codes in one month in a year. The second dataset is based on two zip codes for three months in a year. The third dataset is compiled from two zip codes over 3 months in 4 years. The fourth dataset is based on three zip codes for 3 months across a four-year period. The final dataset was created on five zip codes over a four-year period, spanning 3 months. The details of datasets were described in Table I.

TABLE I
DETAILS OF DATASETS

Dataset ID	Zip Code	Month	Year	Size
D1	74623 28359	January	2023	62 lines
D2	94649 84410	January February March	2023	180 lines
D3	13962 64242	January February March	2020 2021 2022 2023	720 lines
D4	59304 22889 91850	January February March	2020 2021 2022 2023	1080 lines
D5	71896 17321 48529 75958 52612	January February March	2020 2021 2022 2023	1800 lines

All of them are stored in HDFS (Hadoop Distributed File System). We chose HDFS because it is easy to link to the Apache Spark framework, which is still part of the Hadoop ecosystem.

B. Create Snowfall Analysis Program

Snowfall analysis is a program to discover the category of snowfall on a day in some United State areas. The category is classified from the amount of snowfall that area receives in a whole day. The original source of this code is an open-source project available on GitHub. However, authors have modified several parts of this code, thus can fit with the research's objective. The program's purpose is to divide snowfall into five groups (light, light to moderate, heavy, very heavy, extreme, and invalid data), these categories are referred to Snow and Avalanche Study Establishment (SASE) laboratory [14]. The snowfall analysis program has the ability to filter which data is incorrect according to a fact limitation. The invalid data occurs when the snowfall exceeds 1500 mm (about 4.92 ft) for each day, which is unachievable in real life [14].

Scala programming language is used to write the snowfall analysis program. Apache Spark is utilized as the framework, and it will be created in IntelliJ IDEA. There are two steps included in this program to achieve its goal: pre-processing and mapping. The diagram below in Figure 4 is the schema of this program. Program starts with input (zip code, data, and snowfall) from dataset into pre-processing section. Next, pre-processing generates output ((state, date), snowfall), which will be used in the mapping step. Once the mapping process is completed, it will generate output ((state, date), category).

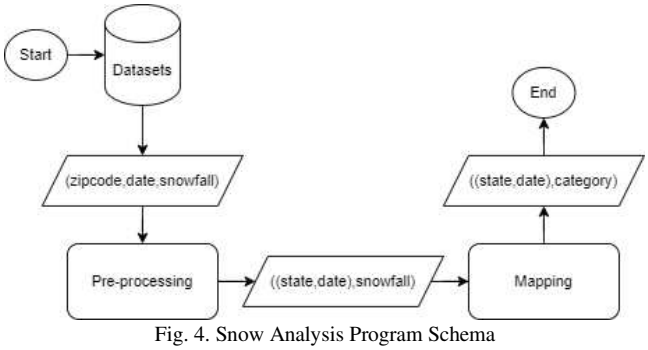

Fig. 4. Snow Analysis Program Schema

The pre-processing step includes data transformations such as reading the dataset; splitting the data separate by comma (,); converting the zip code into state name using zipToState function; converting snowfall

from feet into mm (millimeter) using convert_to_mm function; and finally mapping them into a list with ((state, date), snow) structure for each line. The flow diagram of the preprocessing step is pictured in Figure 5 below.

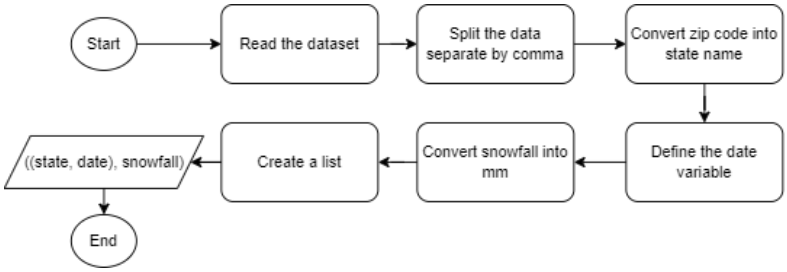

Fig. 5. Preprocessing schema on snowfall analysis program

After done with the preprocessing step, the program started to do the mapping process. The mapping process is classifying each line inside the list using conditional statement. This step categorizes the amount of snowfall each day into several categories. Each gap categories are defined by Snow and Avalanche Study Establishment (SASE) laboratory [14].

C. Implement Titian Library

Titian library is implemented on snowfall analysis program to track inputs which trigger failure. The crucial step is to do configuration on Titian jar into IntelliJ IDEA, this step used for installing all dependencies needed in this project. Once the dependencies have been installed successfully, the Titian library can be accessed in the Snowfall Analysis program to do its function.

Figure 6 below shows the schema of Titian implementation on Snowfall Analysis program. The implementation is started with filtering the output of mapping process. The most critical step in applying Titian for tracking data provenance is to filter only the failure data. The failure function handles the filtering procedure, selecting only snow data that exceeds 1500 mm. The filter function exposed only data that exceeded 1500 mm. Titian then investigated the origins of the data. Titian's results (data provenance) have the same format as the dataset. As a result, it is easy to identify where the input problem occurred.

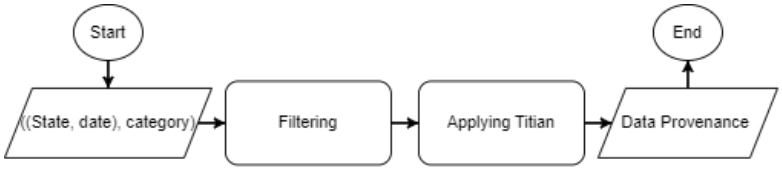

Fig. 6. Titian Schema on Snowfall Analysis Program

D. Verification The Accuracy of Titian

Accuracy is a measurement of how close Titian hitting the goal. Manual verification is used to evaluate the accuracy of Titian library. We applied confusion matrix to gain information about how correct Titian result compare with actual conditions. We made two labels in this matrix i.e., invalid, and valid data. Invalid data represent as positive classification since our goal for this research is finding the data provenance of invalid data. Then, valid data is classified as negative value. Table II below illustrates the classification in confusion matrix.

TABLE II
CONFUSSION MATRIX

		Predicted Condition	
		Positive	**Negative**
Actual Condition	**Positive**	TP	FN
	Negative	FP	TN

From table above we got 4 terminologies:

- True Positive (TP): when the actual condition is positive and predicted condition is positive.
- True Negative (TN): when the actual condition is negative, and the predicted condition is negative.
- False Positive (FP): when the actual condition is negative, but the predicted condition is positive.
- False Negative (FN): when the actual condition is positive, but the predicted condition is negative.

We applied confusion matrix to five different datasets, monitored the Titian result, and compared it with the actual dataset. To evaluate the accuracy of Titian, we implemented equation that used value from the confusion matrix classifications.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Based on (1), to get accuracy, we need to add up the TP (True Positive) and TN (True Negative) then divide it with the total of TP (True Positive), TN (True Negative), FP (False Positive), and FN (False Negative). The greater result of accuracy indicates better accuracy of Titian.

E. Manual Debugging

Manual debugging is done by the author by reading the invalid data from the state name, matching the state name with the zip code list, then search one by one in the dataset for finding the line which triggered failure input.

F. Count The Time for Automated Debugging

The first step was to count the runtime program without applying Titian. The result is noted by the author as a baseline time. The second stage was to count the runtime program, which included Titian. To get both the baseline time and automated debugging time, we tried ten times for each, and the average is reported in the result. Then we calculated the runtime overhead by reducing the runtime using Titian and base time. We used a stopwatch and recorded the result in seconds.

G. Count The Time for Manual Debugging

The time for manual debugging is started when the program has run and showed us the invalid data. For counting manual debugging time, we used a stopwatch and recorded the result in seconds.

H. Result Analysis

When analyzing the result, there are two variables that are monitored. The variables are:

- Accuracy

  To assess Titian's accuracy, we calculated using equation (1) which applied in five datasets. Furthermore, we discovered the average of Titian accuracy from those calculations in percentage. In addition, we created a table analysis to describe the program's outcomes, Titian, and actual dataset. This analysis is conducted to prove that data match for every element in program output, Titian, and actual dataset.

- Efficiency

  To measure the efficiency of Titian, we compared automated debugging using Titian to manual debugging time when performing data lineage on Snowfall Analysis program. The documentation included information about the dataset ID and the time of them in seconds.

IV. RESULTS AND DISCUSSION

This experiment assures that everything is configured the same way, including dataset type and size, JDK version, Spark and Hadoop version, time counting tool, and hardware. This may assist to mitigate the ensuing bias.

A. *Verification the Accuracy of Titian*

The verification of Titian accuracy was conducted to compare and prove the results of Titian and the actual dataset using confusion matrix.

TABLE III
CONFUSSION MATRIX ON D1, D2, D3, D4 AND D5

		Titian Result									
		Invalid Data					**Valid Data**				
		D1	**D2**	**D3**	**D4**	**D5**	**D1**	**D2**	**D3**	**D4**	**D5**
Actual Dataset	**Invalid Data**	16	42	313	298	473	0	0	0	0	0
	Valid Data	0	0	0	0	0	46	138	407	782	1327

Table III demonstrates Titian's consistent and accurate performance across various datasets. For D1, Titian identified all 16 invalid data points correctly (TP=16), with no errors in valid data (FN=0, FP=0, TN=46). For D2, Titian successfully tracked all 42 invalid data points (TP=42), again with no errors (FN=0, FP=0, TN=138). In D3, Titian accurately predicted all 313 invalid data points (TP=313) and matched the valid data perfectly (FN=0, FP=0, TN=407). Similarly, for D4, Titian identified 298 invalid data points correctly (TP=298) with no false predictions (FN=0, FP=0, TN=782). Finally, in the largest dataset, D5, Titian demonstrated excellent predictive accuracy with 473 invalid data points correctly identified (TP=473) and no errors in valid data predictions (FN=0, FP=0, TN=1327). These results underscore Titian's robust performance, accurately identifying invalid data while maintaining zero false positives and false negatives across all datasets.

Since we already have the classification from confusion matrix. The next step was calculating the accuracy of Titian using (1) which is implemented on every dataset.

TABLE IV
IMPELEMENTATION OF EQUATION 1 FOR EACH DATASET

ID Dataset	**TP**	**TN**	**FP**	**FN**	**Accuracy**
D1	16	46	0	0	1
D2	42	138	0	0	1
D3	313	407	0	0	1
D4	298	782	0	0	1
D5	473	1327	0	0	1

Table IV indicates that Titian accuracy for all datasets is 100% accurate, this is referred from accuracy result transformation to percentage. Thus, the average of Titian accuracy is 100%.

B. *Efficiency Test of Titian*

Titian efficiency test is conducted in three experiments. These experiments were conducted to compare automated debugging and manual debugging time. The first step was counting runtime program without applying Titian. This experiment is tried ten times for each dataset. The result from the first experiment is the

average and noted as a base time. The second step was counting the runtime program which involved Titian. Automated debugging time is tried ten times, and the average is written in the result. Then, we found the runtime overhead by reducing the runtime utilizing Titian and base time. The last step was counting time for manual debugging. Manual debugging is tried one time for each dataset.

TABLE V
COMPARISON OF BASE, TITIAN INVOLVED AND TITIAN OVERHEAD TIME

Dataset ID	Base Time (s)	Titian Involved (s)	Time Overhead (s)
D1	6.072	6.620	0.548
D2	6.443	6.903	0.460
D3	6.503	7.066	0.563
D4	6.642	7.065	0.423
D5	6.720	7.251	0.531

Table V above demonstrates that Titian only needs 0.505 seconds (on average) to track data provenance on snowfall analysis program.

TABLE VI
TIME SPENT ON MANUAL AND AUTOMATED DEBUGGING

Dataset ID	Manual Debugging Time (s)	Automated Debugging Time (s)
D1	130.23	0.548
D2	361.52	0.460
D3	2524.59	0.563
D4	2418.38	0.423
D5	3569.59	0.531

According to the experiment results on Table VI, manual debugging takes more time than automated debugging. Manual debugging takes longer as the number of incorrect data points in the dataset increases. Meanwhile, automated debugging takes nearly constant time even though the incorrect data and size grew bigger in the dataset.

C. Discussion

In this study, we identify a pattern in manual debugging time, which increases with the occurrence of invalid data. The experiment with datasets D3 and D4 demonstrates this: despite D3 having fewer lines than D4, the higher quantity of erroneous data in D3 results in longer debugging times.

Constrastingly, automated debugging using Titian does not exhibit a distinct pattern relative to dataset size or the volume of faulty data. Titian maintains a nearly constant runtime across varying datasets. This consistency is a key advantage, as evidenced by our findings. For instance, debugging with Titian is 260 times faster than manual methods for a mini dataset of 62 lines, 902 times faster for a dataset of 180 lines, 5000 times faster for a medium dataset of 720 lines, and 6000 times faster for a dataset of 1080 lines. Notably, for a large dataset of 1800 lines, Titian's efficiency improves by 7000 times compared to manual debugging.

These results highlight Titian's scalability and its increasing efficiency with larger datasets. This scalability suggests that Titian could potentially handle even larger datasets beyond those evaluated, maintaining or even improving its efficiency. The impact of varying data quality on Titian's performance is also significant. While manual debugging time increases with more invalid data, Titian's automated approach remains robust and

consistent, demonstrating its resilience to fluctuations in data quality. This reliability is crucial for practical applications in real-world DISC systems, where data quality can vary widely.

Additionally, while this study focuses on snowfall analysis, the efficiency gains achieved with Titian are likely generalizable to other data analysis tasks. This generalizability is rooted in Titian's fundamental design, which systematically traces data provenance regardless of the spesific application. Future work could explore Titian's application in other domains, such as financial analysis, healthcare data processing, and more.

Integration of Titian within existing data processing workflows can further enhance its practical usabillity. By embedding Titian into standard data pipelines, organizations can streamline debugging processes, reduce downtime, and improve overall data integrity. This integration can be achieved through API connections, plugin development, or incorporation into data management platforms.

The causative analysis of our results indicates that Titian's efficiency stems from its ability to automate the tracing of data provenance, significantly reducing the manual effort required to identify and correct errors. This automation not only saves time but also minimizes the risk of human error, thereby improving the reliability of the debugging process.

In summary, Titian offers a robust an scalable solution for debugging in DISC systems, with substantial efficiency gains across various dataset sizes. Its consistent performance in the face od varying data quality, potential for generalizability, and ease of integration into existing workflows make it a valuable tool for enhancing data provenance and integrity in large-scale data analysis tasks.

V. CONCLUSION

This study concludes that Titian is accurate and efficient in doing automatic debugging in the DISC system with structured dataset type especially for the snowfall analysis program. Titian successfully monitored the data provenance of program outputs with 100% accuracy across a variety of dataset sizes. Furthermore, Titian outperforms manual debugging in terms of time performance, spending nearly the same amount of time even as the dataset size increased.

While solving the root causative factors in software is undoubtedly important, our research aims to address the pressing need for more efficient debugging methods. By providing a robust and scalable automated debugging solution, we lay the groundwork for future research that can further investigate and mitigate the root causes of software errors. Titian could be used in future research to conduct more DISC study cases with unstructured datasets or other types of datasets. Alternatively, Titian can be implemented with another Spark library to provide a helpful solution for rapid decision-making that needs to save more time.

REFERENCES

[1] R. E. Bryant, "Data-intensive scalable computing for scientific applications," *Comput Sci Eng*, vol. 13, no. 6, 2011, doi: 10.1109/MCSE.2011.73.

[2] M. A. Gulzar, M. Interlandi, X. Han, M. Li, T. Condie, and M. Kim, "Automated debugging in data-intensive scalable computing," in *SoCC 2017 - Proceedings of the 2017 Symposium on Cloud Computing*, 2017. doi: 10.1145/3127479.3131624.

[3] M. Interlandi *et al.*, "Titian: Data provenance support in Spark," *Proceedings of the VLDB Endowment*, vol. 9, no. 3, 2016, doi: 10.14778/2850583.2850595.

[4] R. Diestelkämper and M. Herschel, "Tracing nested data with structural provenance for big data analytics," in *Advances in Database Technology - EDBT*, 2020. doi: 10.5441/002/edbt.2020.23.

[5] R. L. Armstrong and M. Hardman, "Monitoring global snow cover," in *Proceedings of The Western Snow Conference*, 1991.

[6] J. C. Ryan *et al.*, "Evaluation of CloudSat's Cloud-Profiling Radar for Mapping Snowfall Rates Across the Greenland Ice Sheet," *Journal of Geophysical Research: Atmospheres*, vol. 125, no. 4, 2020, doi: 10.1029/2019JD031411.

[7] F. Sun *et al.*, "Decreasing trends of mean and extreme snowfall in High Mountain Asia," *Science of the Total Environment*, vol. 921, 2024, doi: 10.1016/j.scitotenv.2024.171211.

[8] F. Sun *et al.*, "Evaluation of multiple gridded snowfall datasets using gauge observations over high mountain Asia," *J Hydrol (Amst)*, vol. 626, 2023, doi: 10.1016/j.jhydrol.2023.130346.

[9] B. Brasnett, "A global analysis of snow depth for numerical weather prediction," *Journal of Applied Meteorology*, vol. 38, no. 6, 1999, doi: 10.1175/1520-0450(1999)038<0726:AGAOSD>2.0.CO;2.

[10] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun ACM*, vol. 51, no. 1, 2008, doi: 10.1145/1327452.1327492.

[11] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on Apache Spark," *International Journal of Data Science and Analytics*, vol. 1, no. 3–4. 2016. doi: 10.1007/s41060-016-0027-9.

[12] P. Buneman, S. Khanna, and W. C. Tan, "Why and where: A characterization of data provenance?," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2001. doi: 10.1007/3-540-44503-x_20.

[13] M. Interlandi, "Supporting Data Provenance in Data-Intensive Scalable Computing Systems," 2018.

[14] A. P. Dimri and U. C. Mohanty, "Snowfall statistics of some SASE field stations in J&K," *Def Sci J*, vol. 49, no. 5, 1999, doi: 10.14429/dsj.49.3858.