IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 12, Special Issue 2, April 2023

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)

|| e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | SJIF: 8.423|| Volume 12, Special Issue 2, April 2023 ||

8th National Conference on Advanced Computing Technologies (NCACT'23)

Organized by

Department of CSE, Velammal Engineering College, Chennnai, India

Debugging of Big Data Analytics made interactive and Automated

Midhuna Mohanraj

U.G. Student, Department of Computer Engineering, Velammal Engineering College, Chennai, Tamil Nadu, India

ABSTRACT: The availability of vast amounts of data in various fields such as science, engineering, healthcare, and business has given rise to the field of Big Data Analytics, which utilizes cloud computing environments and Data-Intensive Scalable Computing (DISC) systems like Hadoop, Spark, and Google's MapReduce to process large datasets. However, developers currently face difficulties in debugging DISC applications as cloud computing makes application development feel like batch jobs, resulting in post-mortem debugging. Developers typically test their code on a small sample dataset and hope that it works in the production cloud. However, when a job fails, data scientists may want to identify the source of errors by investigating a subset of corresponding input records. The vision of this work aims to provide interactive, real-time, and automated debugging services for big data processing programs with minimum performance impact. It focuses on exploring necessary debugging primitives, scalable fault localization algorithms, and ways to improve testing efficiency during iterative development of DISC applications by reasoning the semantics of dataflow operators and user-defined functions. The work synthesizes and innovates ideas from software engineering, big data systems, and program analysis, coordinating innovations across the software stack from the user-facing API to the systems infrastructure.

KEYWORDS:Automated debugging, minimizing tests, debugging and testing, localizing faults, data provenance, data intensive scalable computing (DISC), big data and data cleaning.

I. INTRODUCTION

The availability of vast amounts of data in various fields such as science, engineering, healthcare, and business has given rise to the field of Big Data Analytics, which utilizes cloud computing environments and Data-Intensive Scalable Computing (DISC) systems like Hadoop, Spark, and Google's MapReduce to process large datasets. However, developers currently face difficulties in debugging DISC applications as cloud computing makes application development feel like batch jobs, resulting in post-mortem debugging. Developers typically test their code on a small sample dataset and hope that it works in the production cloud. However, when a job fails, data scientists may want to identify the source of errors by investigating a subset of corresponding input records. The imaginative and prescient of this paintings ambitions to offer interactive, real-time, and automatic debugging offerings for massive statistics processing applications with minimal overall performance impact. It focuses on exploring necessary debugging primitives, scalable fault localization algorithms, and ways to improve testing efficiency during iterative development of DISC applications by reasoning the semantics of dataflow operators and user-defined functions. The work synthesizes and innovates ideas from software engineering, big data systems, and program

II. RELATED WORK

Fisher et al conducted a study by interviewing 16 data analysts at Microsoft to identify the pain points of big data analytics tools. They found that cloud-based computing solutions make debugging more difficult and data analysts often have to sift through trace files distributed across multiple virtual machines (VMs). Similarly, Zhou et al.

Manually categorized 210 failures of a big data platform at Microsoft and found that job failures and slowdowns are common in DISC applications, and many in-field failures are caused by logical and design errors. These findings served as motivation for the development of BIGDEBUG.

Various approaches have been proposed to help developers debug DISC applications by collecting and analyzing execution logs. However, unlike BIGDEBUG, none of these approaches aid in real-time debugging, as they conduct post-mortem log analysis. Inspector Gadget proposes monitoring and debugging data flow programs in Apache Pig but leaves it to others to implement the proposed APIs. Arthur is a post-hoc instrumentation debugger for Spark, requiring the user to write a custom query for post-hoc instrumentation and supporting post-mortem analysis only.

|| e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | SJIF: 8.423|| Volume 12, Special Issue 2, April 2023 ||

8th National Conference on Advanced Computing Technologies (NCACT'23)

Organized by

Department of CSE, Velammal Engineering College, Chennnai, India

Graft is a post-hoc instrumentation debugger for a graph-based DISC computing framework, Apache Giraph . Daphne enables a user to attach a debugger to a remote process on the cluster, but it only works for DISC systems that materialize intermediate results and not for in-memory, pipelined DISC systems like Spark.

III. INTERACTIVE DEBUGGING FOR BIG DATA APPLICATIONS

When developing data-intensive scalable computing (DISC) applications, developers often receive notification of runtime failures or incorrect outputs after wasting many hours of computing cycles on the cloud. While DISC systems like Spark provide execution logs, these logs only show the physical view of Big Data processing and do not offer a logical view of program execution. It's difficult to determine which inputs are causing incorrect results or delays, and crashes can cause previously computed stages to be thrown away, wasting valuable partial results. Finding the intermediate data records responsible for a failure or delay can be challenging, as it requires sifting through millions, if not billions, of records. Therefore, investigating the probable cause of delay or finding straggler records is crucial for improving the runtime of DISC applications.BigDebug [5] is a tool designed to tackle the debugging challenges faced by developers in big data processing using Apache Spark. It provides a set of interactive and real-time debugging primitives, including simulated breakpoints, that allow users to inspect program state in distributed worker nodes and resume relevant sub-computations without interrupting the entire program. Traditional methods for handling anomalies in intermediate data involve terminating the job and rewriting the program to handle the outliers, resulting in high re-run costs. BigDebug addresses this by allowing users to replace any code in the succeeding RDDs after the breakpoint, minimizing the need for costly re-runs. BigDebug offers a solution for users to inspect large volumes of data passing through a dataparallel pipeline. This is achieved through on demand guarded watchpoints, which retrieve only the records that match a user-defined guard predicate that can be dynamically updated in real-time. Fine-grained forward and backward tracing is made possible at the level of individual records, thanks to the integration with Titian. In the event of a job crash, BigDebug provides a real-time quick fix and resume feature, allowing users to modify code or data at runtime without restarting the job from scratch. Additionally, BigDebug offers fine-grained latency monitoring, notifying users of records that take longer to process. The Spark UI is extended by BigDebug to provide a live stream of debugging information that is presented in an interactive and user-friendly manner. We conducted an evaluation of BigDebug's performance overhead, scalability, time-saving, and crash localizability improvement using three Spark benchmark programs that involved up to one terabyte of data. When BigDebug was enabled with the maximum instrumentation setting, which included record-level tracing, crash culprit determination, and latency profiling, as well as breakpoints and watchpoints for every operation at every step, it took 2.5 times longer than the baseline Spark (see Figure 1). However, when we disabled the most expensive record-level latency profiling, BigDebug introduced an average overhead of less than 34%, as shown in Table 1. With BigDebug's quick fix and resume feature, users can avoid having to re-run a program from scratch, resulting in up to 100% time savings. On average, BigDebug exhibited less than 24% overhead for record-level tracing, 19% overhead for crash monitoring, and 9% for on-demand watchpoints.

BENCHMARK	DATASET(GB)	OVERHEAD MAX	OVERHEAD W/O LATENCY
PigMix L1	1, 10,50,100,150,200	1.38X	1.29X
Grep	20,30, 40,…….90	1.76X	1.07X
Word Count	0.5 to 1000(increment)	2.5X	1.34X

Table 1: **Evaluation of program performance for specific subjects**

AUTOMATED DEBUGGING OF DISC WORKFLOWS

In the field of big data analytics, diagnosing errors can be a challenging task. When a program experiences a failure, data scientists may need to investigate a particular subset of the input data that caused the issue. To identify the root cause of the problem, data provenance, which maps the input and output records generated by individual distributed

worker nodes, can be used. However, our previous research has shown that using backward tracing with data provenance can still result in a subset of millions of input records, making it difficult for developers to manually

|| e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | SJIF: 8.423|| Volume 12, Special Issue 2, April 2023 ||

8th National Conference on Advanced Computing Technologies (NCACT'23)

Organized by

Department of CSE, Velammal Engineering College, Chennnai, India

analyze. To address this issue, Delta Debugging is a commonly used algorithm that executes the same program with different subsets of input records. However, applying Delta Debugging in a straightforward manner to big data analytics is not feasible as it is a generic, black-box approach that does not take into account the key-value mapping created by individual dataflow operators.BigSift is a tool we created to help identify the root cause of failures in DISC (Data-Intensive Scalable Computing) workflows with high precision. By combining delta debugging (DD) with data provenance and implementing a unique set of system optimizations geared towards repetitive DISC debugging workloads, we have brought DD closer to reality in DISC environments. Our approach involves redefining data provenance for debugging purposes by leveraging the semantics of data transformation operators. BigSift prunes out input records that are irrelevant to the given faulty output records, reducing the initial scope of failure-inducing records before applying DD. We have also developed a set of optimization and prioritization techniques that specifically benefit the iterative nature of DD workloads. Although our current implementation targets Apache Spark, our approach can be adapted to any data processing system that supports data provenance.

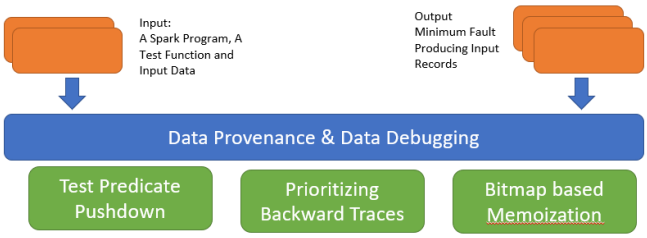

Figure 1: BigSifts Overall Architecture

BigSift is a tool that automates the process of finding the minimum set of fault-inducing input records responsible for a faulty output in three phases. In the first phase, BigSift uses test-driven data provenance to remove input records that are not relevant for identifying the fault(s) in the initial scope of fault localization. This is achieved by redefining the notion of data provenance and pushing down a test oracle function from the final stage to an earlier stage, testing partial results instead of final results. In the second phase, BigSift prioritizes the backward traces by implementing trace overlapping, based on the insight that faulty outputs are rarely independent. The smallest backward traces are prioritized first to explain as many faulty output records as possible within a given time limit. In the third phase, BigSift performs optimized delta debugging while leveraging bitmap-based memoization to reuse the test results of previously tried sub-configurations when possible. BigSift also augments the current Spark UI and provides a live streamof debugging information. Given a DISC program, an input dataset, and a user-defined test function that distinguishes the faulty outputs from the correct ones, BigSift automates the process of finding the root cause of failures in a more efficient and effective manner.

Running Time (s)		Debugging Time (s)		
Program	Original Job	DD	BigSift	Improvement
Movie Histogram	56.2	232.8	17.3	13.5X
Inverted Index	107.7	584.2	13.4	43.6X
Rating Histogram	40.3	263.4	16.6	15.9X
Sequence Count	356.0	13772.1	208.8	66.0X
Rating Frequency	77.5	437.9	14.9	29.5X
College Students	53.1	235.2	31.8	7.4X
Weather Analysis	238.5	999.1	89.9	11.1X
Transit Analysis	45.5	375.8	20.2	18.6X

Table 2: Enhancing fault localization speed with Big Shift

|| e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | SJIF: 8.423|| Volume 12, Special Issue 2, April 2023 ||

8th National Conference on Advanced Computing Technologies (NCACT'23)

Organized by

Department of CSE, Velammal Engineering College, Chennnai, India

Compared to the use of DP alone, BigSiftis able to achieve a more concise set of fault-inducing input records, thereby improving the accuracy of fault localization by several orders of magnitude. Typically, data provenance can only identify fault-inducing records up to a certain size, ranging from about 10^3 to 10^7 records, which still remains impractical for developers to manually investigate. In comparison to using DD alone, BigSift accelerates fault localization time (up to 66×) by eliminating irrelevant input records. In addition, our trace overlapping heuristic reduces the total debugging time by 14%, while our test memoization optimization results in up to 26% decrease in debugging time. In fact, the overall time spent on debugging using BigSift is often 62% less than the time required for a single faulty output to complete its original job run. This is a significant improvement since, in the software engineering literature, debugging time is generally much longer than the original running time of a program [1, 2, 9]

WHITE-BOX TEST DATA SAMPLING AND GENERATION IN DISC

Testing big data applications can be a challenging and resource-intensive process due to the sheer volume of input data that is required. The impracticality of manually reviewing production data to create test inputs is compounded by the fact that data can come from various sources and may be unstructured, poorly formatted, or lacking a schema. Additionally, even though data records follow the same data parallel pipeline, different records may not take the same program path through each user-defined function. Random sampling of data is not a sufficient solution as it may compromise test adequacy and fault detection ratios.

During iterative program development, developers may need to select a subset of relevant data or generate new input records to test program modifications. Furthermore, regression testing of data flow programs can be complicated by the interaction of user-defined functions and data flow operators, necessitating test selection and augmentation techniques that consider both aspects together.

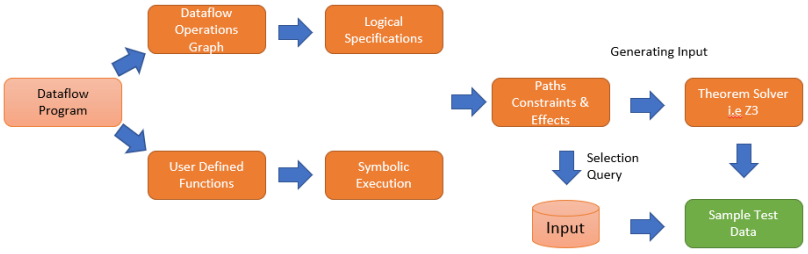

Figure 3: BigSample's Overall Architecture

We have developed a new technique called BigSample for selecting and augmenting tests in DISC applications. BigSample is designed to enhance code statement coverage and fault detection rates in big data analytics applications by directly considering the semantics of dataflow programs. The technique involves a symbolic execution engine that models the path conditions of user-defined functions (UDFs) together with dataflow operator semantics, such as join and group-by. BigSample employs test minimization, augmentation, and selection to reduce the amount of time required for testing. To test big data workflows, BigSample converts the path conditions from UDFs and dataflow operator specifications into a test input selection query that produces a subset of input records for iterative development and testing. This test minimization technique is akin to a new white-box data sampling method. If the reduced set of records does not cover certain path conditions, BigSample's test augmentation algorithm generates the required input records using off-the-shelf theorem provers like Z3. This process is illustrated in Figure 3. Finally, BigSample performs regression test selection by evaluating the impact of application logic changes on data. To evaluate our test selection and augmentation technique, we intend to compare its efficiency and effectiveness against two baseline techniques. The first baseline technique only considers the semantics of data flow operators at a high level during test selection and augmentation. The second baseline technique is a straightforward re-testing method that reruns the program on all input records. We will assess the testing efficiency of our technique in terms of two aspects. Firstly, we will measure the extent to which the technique reduces the size of the original input records that are selected. Secondly, we will examine the trade-offs between fault detection rates and the size of the selected input records.

|| e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | SJIF: 8.423|| Volume 12, Special Issue 2, April 2023 ||

8th National Conference on Advanced Computing Technologies (NCACT'23)

Organized by

Department of CSE, Velammal Engineering College, Chennnai, India

V. CONCLUSION

In this article, the present situation and upcoming plans of BIGDEBUG are presented, which is designed to offer developers interactive and tool-supported debugging features for big data analysis. By utilizing BIGDEBUG, developers can considerably diminish the time required for debugging, resulting in decreased overall time to bring their big data applications to the market.

REFERENCES

[1] Jong-Deok Choi and Andreas Zeller. 2002. Isolating Failure-inducing Thread Schedules. In Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '02). ACM, New York, NY, USA, 210–220. https://doi.org/10.1145/566172.566211

[2] Holger Cleve and Andreas Zeller. 2005. Locating Causes of Program Failures. In Proceedings of the 27th International Conference on Software Engineering (ICSE '05). ACM, New York, NY, USA, 342–351. https://doi.org/10.1145/1062455.1062522

[3] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. Tools and Algorithms for the Construction and Analysis of Systems (2008), 337–340.

[4] Muhammad Ali Gulzar, Matteo Interlandi, Xueyuan Han, Mingda Li, Tyson Condie, and Miryung Kim. 2017. Automated Debugging in Data-intensive Scalable Computing. In Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17). ACM, New York, NY, USA, 520–534. https://doi.org/10.1145/3127479.3131624

[5] Muhammad Ali Gulzar, Matteo Interlandi, SeunghyunYoo, Sai Deep Tetali, Tyson Condie, Todd Millstein, and Miryung Kim. 2016. BigDebug: Debugging Primitives for Interactive Big Data Processing in Spark. In Proceedings of the 38th International Conference on Software Engineering (ICSE '16). ACM, New York, NY, USA, 784–795. https://doi.org/10.1145/2884781.2884813

[6] Matteo Interlandi, Kshitij Shah, Sai Deep Tetali, Muhammad Ali Gulzar, SeunghyunYoo, Miryung Kim, Todd Millstein, and Tyson Condie. 2015. Titian: Data Provenance Support in Spark. Proc. VLDB Endow. 9, 3 (Nov. 2015), 216–227. https://doi.org/10.14778/2850583.2850595

[7] Jeffrey D. Ullman. 1990. Principles of Database and Knowledge-Base Systems: Volume II: The New Technologies. W. H. Freeman & Co., New York, NY, USA.

[8] MateiZaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12). USENIX Association, Berkeley, CA, USA, 2–2. http://dl.acm.org/citation.cfm?id=2228298.2228301Nobuo Ezaki, Marius BulacuLambert ,Schomaker , "Text Detection from Natural Scene Images: Towards a System for Visually Impaired Persons" , Proc. of 17th Int. Conf. on Pattern Recognition (ICPR), IEEE Computer Society, pp. 683-686, vol. II, 2004

[9]Muhammad Ali Gulzar. University of California, Los Angeles.Interactive and Automated Debugging for Big Data Analytics.In Companion Proceedings of the2018ACM/IEEE 40th International Conference on Software Engineering.

INNO SPACE
SJIF Scientific Journal Impact Factor

SJIF: 8.423

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INTERNATIONAL CENTRE

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details