Demonstration of Interactive Runtime Debugging of Distributed Dataflows in Texera

Zuozhi Wang, Avinash Kumar, Shengquan Ni, and Chen Li
Department of Computer Science, UC Irvine, CA 92697, USA
{zuozhiw, avinask1, shengqun, chenli}@ics.uci.edu

ABSTRACT

We are developing Texera, an open source system that allows users to perform data analysis on a computing cluster using a GUI-based workflow. A unique functionality of the system is its support for interactive and responsive debugging on dataflows during their execution, while still being scalable and fault tolerant. In particular, users can pause/resume a workflow, investigate the state of operators, change the behavior of an operator, and set conditional breakpoints. In this way, a user will not feel "in the dark" during the long-running execution of an analytics task, a problem faced by other big data processing frameworks. In this demonstration we show this powerful functionality in Texera.

PVLDB Reference Format:
Zuozhi Wang, Avinash Kumar, Shengquan Ni, and Chen Li. Demonstration of Interactive Runtime Debugging of Distributed Dataflows in Texera. *PVLDB*, 13(12): 2953-2956, 2020.
DOI: https://doi.org/10.14778/3415478.3415517

1. INTRODUCTION

As information volumes in applications continuously grow, analytics of large amounts of data becomes increasingly important. This trend has fueled the emergence and popularity of many big data frameworks such as Apache Spark and Apache Flink. Big data analytics jobs can take hours or even days to finish. If a job fails, the earlier computing resources are wasted and a new job needs to be submitted from scratch. Existing big data frameworks provide limited support for identifying reasons of failures. Analysts undertake certain pre-execution analysis to avoid failures during job execution [4]. A common approach is running the job first on a small data set with the hope of detecting and solving problems earlier. Unfortunately, many run-time failures occur only on a big data set. For instance, a software bug that is triggered only by some rare, outlier records, which may not appear in a small data set [7]. Another approach is post-execution analysis, which requires instrumenting the software code to generate log entries that can be used to determine the errors. This approach has several limitations. First, the analyst has to add logs at many places in order to find bugs. Consequently, an inordinate amount of log records are produced that have to be analyzed offline, and most of them are irrelevant. Second, these log records may not reveal all the information about the run-time behavior, making it hard to identify errors. This situation is analogous to the scenario of debugging a C program. Instead of using `printf()` to produce log messages and do post-execution analysis, many developers prefer to use a debugger such as `gdb` to investigate the run-time behavior of the program *during* its execution.

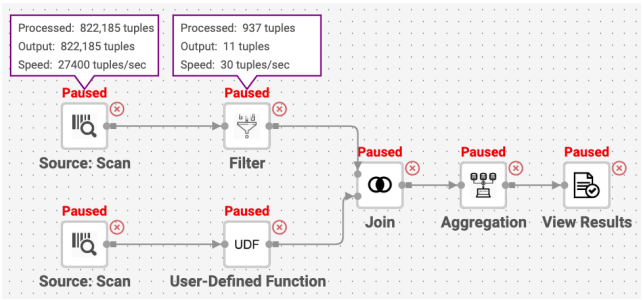

Figure 1: A paused workflow in Texera showing status of operators

In this paper we demonstrate interactive runtime debugging in Texera, a big data system being developed by our UCI team in the past four years. It allows users to construct an analytics job by formulating a workflow using a GUI interface [15]. Data analysts from different domains and possessing varied levels of IT expertise can use Texera without being handicapped by their programming skills. Texera is powered by a parallel and cluster-based engine called "Amber" [11], which can efficiently process large amounts of data. A main feature that differentiates Texera from other frameworks such as Spark is that Amber allows users to interact with and debug an analytics job *during* its execution. Users can pause the job during its execution, investigate the states of operators, and modify the parameters of the operators. The operators can run using the new parameters when the job is resumed. Users can also detect bugs and data errors by setting distributed conditional breakpoints before or during the execution of a workflow. Conditional breakpoints accept data-oriented predicates provided by the user. When a predicate is satisfied, the execution automatically pauses.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415517

An important aspect of Texera is that it supports these interactions and debugging features in a responsive way, which is vital for GUI-based services. Figure 1 shows the Texera web UI when a workflow is paused. The interface shows various information such as the status of each operator, its number of processed tuples, and processing speed.

Related Work: There are GUI-based workflow systems such as Alteryx [2] and Kepler [10]. These systems do not run on clusters and do not support debugging either. Apache Airavata [1] supports pause but a user has to wait for an operator to completely finish processing all its data. BigSift [8] provides an approach for finding input data responsible for producing erroneous results in Spark. BigDebug [7] introduces the concept of simulated breakpoint in Spark execution. After reaching a simulated breakpoint, the results computed till then are materialized, but the computation still continues. If the user makes changes to the workflow after the simulated breakpoint, the existing execution is cancelled, causing computing resources to be wasted. Texera is different since the developer can set a breakpoint or explicitly pause the execution at any time, and the computation is truly paused. Query-profiling tools such as Perfopticon [12] simplify the analysis of distributed query execution, but they are limited to discovering run-time bottlenecks and problematic data imbalances. StreamTrace [3] aids developers in constructing correct queries by producing visualization that illustrates the behavior of queries. Such tools cannot be used to support run-time debugging. RAMP [13] provides post-execution provenance support for MapReduce jobs. There are tools that allow tracing of tuples during runtime [14], but they are limited to identifying the provenance of the traced tuples. IBM DataStage [6] is a commercial system that provides support for debugging dataflows using conditional breakpoints. Its debugging features are limited when switching execution from a single-server mode to a cluster mode. TagSniff [5] is a debugging model for dataflows that uses instrumentation to tag and identify tuples for further analysis. It does not support features such as arbitrary run-time pause and distributed breakpoints that require collaboration among multiple nodes.

Compared to these existing systems, Texera has the following novelties in debugging:

1. Texera supports fast (sub-second) user-initiated pausing and resuming of jobs executing on a distributed cluster. A paused job is still responsive to user interactions, which allows users to debug a paused job.
2. Texera supports distributed conditional breakpoints where conditions are evaluated collaboratively by multiple machines of a cluster.
3. Debugging is efficient and scalable on large clusters.
4. Texera supports fault tolerance that enables recovery to the exact data processing and debugging states. This feature is in contrast to other systems where fault tolerance does not support recovery of debugging states.

2. TEXERA SYSTEM OVERVIEW

Figure 2 shows the architecture of Texera. Using a Web UI, Texera developer can construct a workflow using a rich operator library including relational operators, text processing operators, and machine learning operators. Texera also supports Java and Python User Defined Functions (UDF). The workflow is then compiled to a physical operator DAG by the query compiler. The physical operator DAG is executed on the backend engine called Amber, a distributed cluster-based dataflow processor. Debugging capabilities are provided by Amber on this physical operator DAG. The Web UI establishes a two-way communication channel with the backend through a WebSocket API, which enables interactive communication between the frontend and the backend.

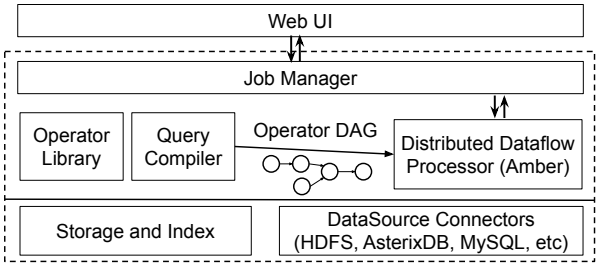

Figure 2: Texera system architecture

Amber is built on top of the actor model [9]. At run time an operator is implemented as multiple actors, each of which processes a data partition. Data goes through the workflow as data messages, and debugging requests are transmitted as control messages. Amber is able to process debugging requests in a timely manner using its expedited control message delivery mechanism. Experiments showed consistent sub-second latency for handling debugging requests even on large clusters with 100 machines [11]. Amber provides these features along with the support of fault tolerance. In case of a failure, Amber not only ensures the correctness of the final computation result, but also recovers the same consistent debugging state. Its performance is comparable to Apache Spark. Texera leverages the above mentioned capabilities of Amber and makes them easy to use through a user-friendly Web UI.

3. DEMONSTRATION SCENARIO

In this section, we illustrate the various debugging capabilities in Texera using an example. Suppose Emily is a data analyst who wants to study depression as a side-effect of tramadol (an opioid) addiction on various age groups using twitter data. She constructs a workflow as shown in Figure 3 using Texera to perform the analysis on a cluster of machines. Since Twitter does not provide information about user demographics, Emily develops a machine learning (ML) algorithm for inferring a user's age by analyzing the user profile. The ML algorithm is included in Texera as a user-defined operator. Emily adds a filter operator that uses a conjunction of a regular expressions (regex) and keywords to filter tweets. Tramadol is a frequently misspelled word, and Emily uses the regular expression "[Tt]ram+adol?s?" that enumerates its common misspellings. She also provides a keyword "depression" to the Filter operator. This operator first applies the regular expression to identify tweets pertaining to tramadol, then applies a keyword search using "depression" on the identified tweets. The filtered tweets are joined with user profiles on the user-id attribute. The Join results are grouped by user's age and the total count per age bracket is calculated.

In the demonstration, we will provide a large twitter data set, a data set of New York taxi events, and a MEDLINE

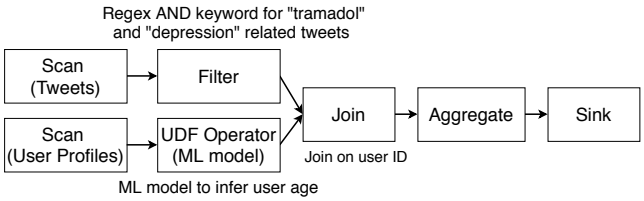

Figure 3: Example workflow of tweet analysis

medical data set. We will provide the aforementioned workflow and several pre-constructed workflows for tweet analysis of topics such as public health. Custom workflows can be constructed as well using a rich library of built-in operators. The audience will be able to run these workflows on a parallel cluster and enjoy the unique features of Texera, as explained next.

3.1 Pausing and Investigating Workflows

The capability to investigate the internal states of a long running job is vital to users. Many big data processing frameworks provide status updates for running jobs. However, these systems only allow users to monitor their jobs passively. That is, even if users notice anomalies happening during the execution, they can only either kill the job or wait for the job to run to its completion. In contrast, Texera users can pause and investigate a workflow if they have doubts about the correctness of the job.

The Texera interface displays runtime metrics of each operator, thus helping the users gain valuable insights about the jobs. In the running example, Emily uses the Texera interface to check the state of each operator, such as its number of input and output records and processing speed. She observes that the Filter operator is processing tuples very slowly. Based on her past experience, she has an inkling that the issue could be due to the complexity of the regular expression. She realizes that the performance of the Filter operator can be improved by changing the evaluation order of the regular expression match and keyword search, i.e., by performing the keyword search using the "depression" keyword first and then applying the regular expression match next. To make this change, Emily pauses the workflow and modifies the Filter operator accordingly. She resumes the workflow and observes an improvement in the processing speed. If the records processed using the old Filter logic are acceptable, Emily can simply resume the workflow after modifying the operator. In cases where the operator modification requires the earlier records to be reprocessed (e.g., change the aggregation function of an Aggregation operator), Texera also supports re-running the workflow from the last stage boundary.

Texera can support fast response (in subseconds) even on a large cluster, thanks to Amber's capability of expedited control message delivery. If a control message arrives while an actor is still processing data, Amber allows the actor to prioritize the control message and respond to the message. Once the workflow starts, the Texera Job Manager communicates with the backend Amber periodically (e.g., every second) to ask for the workflow status. Amber broadcasts a status-check control message to worker actors of the operators, collects the latest information, and reports it to the Job Manager. The Job Manager sends the updates to the Web UI through the WebSocket channel. Notice that when the execution is paused, the workflow is still responsive to control messages from the user.

3.2 Distributed Breakpoints

Texera supports conditional breakpoints that can be used to automatically pause the workflow when a condition becomes true. Conditional breakpoints accept data-oriented predicates, which can be used to enforce data-related constraints in the workflow. During the run time, if a predicate is true, the corresponding conditional breakpoint is triggered, the workflow is paused, and the Web UI notifies the user. For example, suppose Emily thinks that it is rare for a person above 90 years to use Twitter. So she puts a conditional breakpoint (inferred_age $>$ 90) on the output of the ML operator to detect such occurrences. Figure 4 shows the UI when a record triggers the conditional breakpoint. Emily can see an outlier record with an inferred age of 94. She can investigate the user profile to determine the validity of this record. If she believes the user's age is indeed 94, she can simply resume the processing. Alternatively, she has the option to ignore this record in the final result. She can also directly modify the content of the record if she knows the age of the user.

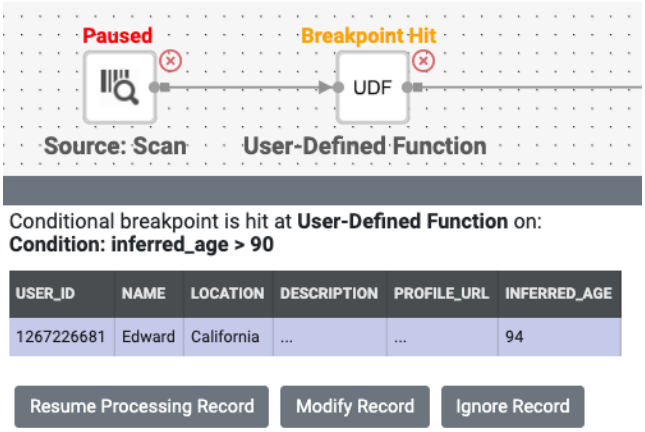

Figure 4: Conditional breakpoint

The condition mentioned above is local to each actor and hence can be independently detected by a single actor. As Texera is a distributed system, it is also critical to detect conditions collaboratively by multiple actors, e.g., the count of tuples produced by the actors of an operator should be within a threshold. Texera supports this type of global distributed conditional breakpoints as well. Such distributed conditional breakpoints are especially helpful in production environments to detect input data corruption and ML model degradation. In the running example, Join is a complex operation and Emily does not want to waste computing resources only to find out later that there was a problem with the workflow. In particular, she wants to review the first 100 tuples produced by Join to ensure the correctness of the output. Thus she sets a conditional breakpoint at the output of the Join operator to pause the workflow after 100 tuples have been produced by the operator (produced_tuples_count = 100). Amber coordinates with all the Join actors to pause the execution after producing exactly 100 tuples.

Pause on Exceptions. Texera provides a unique type of conditional breakpoint, called "exception breakpoint." When an exception occurs during the job execution, Amber automatically catches the exception and pauses the workflow. The Web UI displays the exception details and the culprit tuple to the user. This feature is especially helpful when the workflow involves user-defined functions (UDF's), that are usually tested on a small data set and can fail in a production environment. In a traditional dataflow engine, the entire job would crash if the UDF operator throws an exception, and consequently the computation is wasted. Moreover, the user is forced to go through the mundane and cumbersome process of checking logs to trace the exception.

In the running example, Emily creates a UDF operator using an ML model that infers the age of a user and enables the exception breakpoint. When she runs the workflow on a large amount of data, a NullPointerException occurred. Texera shows Emily the culprit record, which has a null value in the user-description field. It does not conform to the schema expected by the UDF operator that assumes the field is not null. Emily can see the exception information. She can modify or skip the record similar to any conditional breakpoint. Alternatively, she can provide a code fix, which will be applied on all the remaining records when the workflow resumes. Similar to the case in Section 3.1, if the code fix requires a recomputation of previous records, Emily can choose to rerun the workflow from the last stage boundary.

In the demonstration, we will showcase the exception breakpoint feature by providing erroneous input data and faulty UDF operators. The audience will be able to run the workflows, observe the runtime error, and provide fixes to the data or the UDF operator. We will also allow the audience to set up their own conditional breakpoints and check the workflow status when a breakpoint is hit.

3.3 Fault Tolerance

Existing big data processing systems ensure the correctness of the final computation results using techniques such as checkpointing, but not the intermediate states. The interactive features of Texera such as pause, conditional breakpoint, operator modification, and resume make it possible for users to change the runtime state and behavior during its execution. Thus, it is prudent to recover to the correct state in case of failures. Texera, through its engine Amber, guarantees not only correct computation results, but also recovery to the correct consistent state before each failure.

In the running example, the workflow has paused after the Join operator produced 100 tuples, and Emily starts to inspect the tuples. During this process, assume that a machine running a Join actor crashes due to software or hardware failures. Suppose this actor is paused at the 10^{th} tuple at the time of failure. Since this tuple is inspected by Emily, the recovered actor should pause at this precise tuple. Texera starts the recovery process by restarting the computation of the failed machine on another machine. The fault tolerance mechanism captures the content and order of control messages with respect to data messages in an efficient way. Consequently, Texera guarantees that the recovered actor pauses at exactly the same tuple. This process is transparent to Emily, as she continues to observe the exact same state after the recovery. Texera also provides fault tolerance for other debugging features such as operator logic modification. Suppose an operator's logic is modified after it has processed its 100^{th} tuple. In case of a failure, Texera recovers the states by executing the operator with the original logic for the first 100 tuples, and then switching to the modified logic for tuples thereafter. Therefore, the user observes the exact same behavior of the workflow, as if there were no failure. Interested readers can refer to [11] for details of fault tolerance. During the demonstration, we will allow the audience to kill a machine and observe that the system recovers to the same consistent state as before.

Acknowledgements: We want to thank the entire Texera team for their contributions to both the backend and the frontend of the system.

4. REFERENCES

[1] Airavata Website, https://airavata.apache.org/.
[2] Alteryx Website, https://www.alteryx.com/.
[3] L. Battle, D. Fisher, R. DeLine, M. Barnett, B. Chandramouli, and J. Goldstein. Making sense of temporal queries with interactive visualization. In *CHI 2016*, pages 5433–5443, 2016.
[4] I. Beschastnikh, P. Wang, Y. Brun, and M. D. Ernst. Debugging distributed systems. *Commun. ACM*, 59(8):32–37, 2016.
[5] B. Contreras-Rojas, J. Quiané-Ruiz, Z. Kaoudi, and S. Thirumuruganathan. Tagsniff: Simplified big data debugging for dataflow jobs. In *SoCC 2019*, pages 453–464. ACM, 2019.
[6] IBM DataStage, https://www.ibm.com/products/infosphere-datastage.
[7] M. A. Gulzar, M. Interlandi, T. Condie, and M. Kim. Debugging big data analytics in spark with *BigDebug*. In *SIGMOD 2017*, pages 1627–1630, 2017.
[8] M. A. Gulzar, M. Interlandi, X. Han, M. Li, T. Condie, and M. Kim. Automated debugging in data-intensive scalable computing. In *SoCC 2017*, pages 520–534, 2017.
[9] C. Hewitt, P. B. Bishop, and R. Steiger. A universal modular ACTOR formalism for artificial intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, USA, August 20-23, 1973*, pages 235–245, 1973.
[10] Kepler Website, https://kepler-project.org/.
[11] A. Kumar, Z. Wang, S. Ni, and C. Li. Amber: A debuggable dataflow system based on the actor model. *PVLDB*, 13(5):740–753, 2020.
[12] D. Moritz, D. Halperin, B. Howe, and J. Heer. Perfopticon: Visual query analysis for distributed databases. *Comput. Graph. Forum*, 34(3):71–80, 2015.
[13] H. Park, R. Ikeda, and J. Widom. RAMP: A system for capturing and tracing provenance in mapreduce workflows. *PVLDB*, 4(12):1351–1354, 2011.
[14] W. D. Pauw, M. Letia, B. Gedik, H. Andrade, A. Frenkiel, M. Pfeifer, and D. M. Sow. Visual debugging for stream processing applications. In *Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, pages 18–35, 2010.
[15] Z. Wang, F. Bayer, S. Lee, K. Narendran, X. Pan, Q. Tang, J. Wang, and C. Li. A demonstration of textdb: Declarative and scalable text analytics on large data sets. In *ICDE 2017*, pages 1403–1404, 2017.