Demonstration of Collaborative and Interactive Workflow-Based Data Analytics in Texera

Xiaozhen Liu, Zuozhi Wang, Shengquan Ni, Sadeem Alsudais, Yicong Huang, Avinash Kumar, and Chen Li
Department of Computer Science, UC Irvine, CA 92697, USA
{xiaozl3,zuozhiw,shengqun,salsudai,yicongh1,avinask1,chenli}@ics.uci.edu

ABSTRACT

Collaborative data analytics is becoming increasingly important due to the higher complexity of data science, more diverse skills from different disciplines, more common asynchronous schedules of team members, and the global trend of working remotely. In this demo we will show how Texera supports this emerging computing paradigm to achieve high productivity among collaborators with various backgrounds. Based on our active joint projects on the system, we use a scenario of social media analysis to show how a data science task can be conducted on a user friendly yet powerful platform by a multi-disciplinary team including domain scientists with limited coding skills and experienced machine learning experts. We will present how to do collaborative editing of a workflow and collaborative execution of the workflow in Texera. We will focus on data-centric features such as synchronization of operator schemas among the users during the construction phase, and monitoring and controlling the shared runtime during the execution phase.

PVLDB Reference Format:
Xiaozhen Liu, Zuozhi Wang, Shengquan Ni, Sadeem Alsudais, Yicong Huang, Avinash Kumar, and Chen Li. Demonstration of Collaborative and Interactive Workflow-Based Data Analytics in Texera. PVLDB, 15(12): 3738 - 3741, 2022.
doi:10.14778/3554821.3554888

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Texera/texera.

1 INTRODUCTION

Recently online collaboration and sharing services such as Google Docs, Lucid Chart, and Overleaf are becoming increasingly popular and important. These services allow people to collaboratively contribute to tasks, such as documents, drawings, and Latex articles. A similar emerging computing paradigm is *collaborative data analytics*, which allows people to jointly conduct a data-analysis job. Similar to the aforementioned services, this computing paradigm provides two unique benefits. First, it allows a big data-analysis project to be divided into small pieces, which can be done by different team members. Second, it allows collaborators to contribute either concurrently or asynchronously based on their own schedules, possibly in different time zones. These benefits become even more attractive due to the new norm of working remotely caused by the unprecedented COVID-19 pandemic.

Texera is an open source system we have been developing since 2016 to support collaborative data analytics at scale. We are particularly interested in the scenarios where the collaborators are from multiple disciplines with different backgrounds, and the analytics is machine learning (ML)-centric, as such tasks are becoming increasingly common and important. To allow data analytics by people without a strong IT background (e.g., no programming skills required), Texera provides a GUI-based interface for users to formulate a task as a DAG of operators [7].

In this demo we will use a scenario of social media analysis by an interdisciplinary team of researchers to show how Texera supports collaborative data analytics. It is based on more than seven collaborations our team is involved in with experts in areas such as public health, machine learning, and big data systems. We will describe how Texera supports shared editing and shared execution of a workflow. Compared to collaborative services such as Google Docs and Overleaf, one unique collaborative feature specific to data analysis is how operator schemas are recomputed and propagated to collaborators when one user is editing one operator (Section 3.1). Another is how to allow multiple users to share the execution phase of a workflow (Section 3.2), such as supporting multiple users to simultaneously view progressive execution results, control the execution by pausing or resuming the workflow, and independently inspect the workflow's internal state. We will focus on how these data-oriented features are supported in Texera.

Related Work: Table 1 illustrates the main differences between Texera and related systems. Workflow-based systems such as Alteryx [1], Knime [3], and RapidMiner [6] require users to download their software and apply update patches periodically. This "pre-cloud" architecture makes these systems hard to support real-time collaboration, since each developer mainly uses their local software. For example, a cheat sheet provided by Knime [4] shows the high complexity to do collaboration. Cloud-based collaborative services such as Google Colab and DeepNote are mainly targeting Python developers using Jupyter Notebook, and they are not suitable for domain scientists with limited programming skills. Einblick [2] emphasizes interactivity and real-time collaborative editing of workflows. To our best knowledge, there is no published work describing collaboration-related features in the system. Compared to these systems, Texera is unique as its backend engine called "Amber" not only supports parallel computing on clusters, but also provides

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554888

Table 1: A comparison of related data analytics systems

System	Task Formulation	Architecture	Main Target Users	Shared Editing	Shared Execution	Distributed Engine	Execution Pausing	Version Control	Open Source
Alteryx	GUI-based workflows	Software installation	Non-IT people and Python developers	No	No	No	No	Yes	No
Knime, RapidMiner	GUI-based workflows	Software installation	Non-IT people and Python developers	No	No	No	No	Yes	Yes
Google Colab	Jupyter Notebook	Cloud service	Python developers	No	No	No	Yes	Yes	No
DeepNote	Jupyter Notebook	Cloud service	Python developers	Yes	Yes	No	Yes	Yes	No
Einblick	GUI-based workflows	Cloud service	Non-IT people and Python developers	Yes	No	No	No	No	No
Texera	**GUI-based workflows**	**Cloud service**	**Non-IT people and Python developers**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

powerful debugging features such as pausing, resuming, and conditional breakpoints during the execution of a workflow [5, 8]. By supporting GUI-based workflows and allowing user-defined functions (UDF) in languages such as Python, Texera can be used by both domain scientists and experienced Python developers.

2 TEXERA SYSTEM OVERVIEW

Figure 1 shows Texera's architecture related to collaboration features. The system includes Amber as the backend engine, a web server, and a frontend for users to construct and execute workflows using the web UI. The Amber engine allows various kinds of runtime debugging interactions. It is capable of processing interaction requests with a sub-second latency even on a large cluster of machines, which greatly facilitates an interactive user experience. The web server consists of two modules. The *shared editing manager* allows multiple users to co-edit a workflow in real-time within the same editing session. The *synchronization manager* is responsible for handling edits from users, resolving potential conflicts, and propagating changes to other users to ensure all users have the same view of the workflow. The *workflow analyzer* supports workflow validation and schema propagation, which are used for error checking and auto-complete on the web UI.

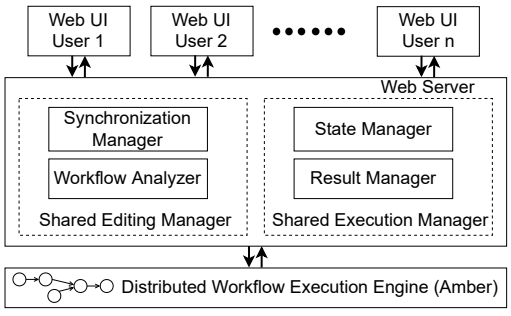

Figure 1: Texera architecture to support collaboration

After the workflow execution is started, the Web UIs of the collaborators connect to the *shared execution manager* and join the same session. The *state manager* monitors the state changes from the Amber engine and pushes the updates to the web UIs. The *result manager* keeps the progressive results. Users can submit interaction requests, such as pausing/resuming workflow and investigating the workflow state to the Amber engine through the shared execution manager. Details about these components will be explained shortly.

3 DEMONSTRATION SCENARIO

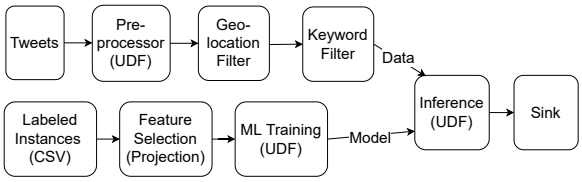

Figure 2: A workflow for ML-based tweet analysis

In this section, we will use an example to demonstrate the various features in Texera that enable a collaborative experience throughout the lifecycle of a workflow. Consider a cross-disciplinary research project involving multiple members. Alice is a machine learning expert in Pennsylvania and Bob is a social scientist in California. Additionally, Emily is Alice's colleague in New York who is particularly skilled at debugging ML models. They want to work together and use ML to analyze tweets to study the relationship between gun violence incidents and public opinions about law enforcement. They construct a workflow shown in Figure 2. (We use a simplified workflow for illustration purposes.) The workflow consists of three parts: 1) data cleaning, 2) ML model training, and 3) ML model inference. Given labeled training data, the workflow trains an ensemble ML model based on selected features. The workflow cleans the large collection of tweets and extracts a subset related to specific gun violence incidents and law enforcement. The workflow applies the trained ML model on these tweets to produce statistics and visualizations about people's opinions. The workflow requires both technical skills and domain knowledge to construct, execute, and debug, and the collaborators will take advantage of Texera's collaboration environment throughout the process.

3.1 Collaborative Workflow Construction

In the workflow-construction phase, Alice and Bob co-edit the workflow on their browsers in real-time.

Shared workflow editing. After Bob creates an initial workflow, he invites Alice as a collaborator, and they can edit the workflow concurrently. For example, Bob has a vague idea about what should be done to extract relevant tweets. In particular, some preprocessing steps (e.g., removing duplicates and adding geo-location information) are necessary, a geo-location-based filter should be applied to narrow the search space to specific incidents, and keywords related to law enforcement should be applied to identify relevant tweets. These advanced operations cannot be done using existing native operators in the system, and they can only be implemented as UDF operators in Python. Unfortunately, Bob has limited knowledge to write Python code. Here Alice offers help to write the UDF operators, while Bob can work on other operators, e.g., setting the properties of the downstream filter operator. Similarly, for ML model training, Bob uses his domain knowledge to edit the feature-selection operator, while Alice works on the training UDF operator. Both collaborators can concurrently edit the workflow and see what each other is doing. They can also co-edit the same operator, as Texera automatically resolves editing conflicts.

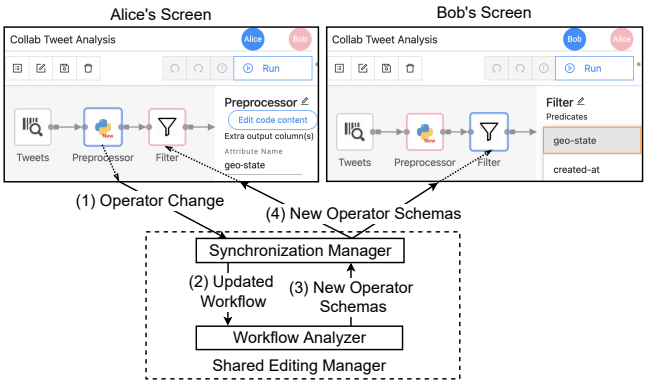

Figure 3: Real-time schema synchronization. On Alice's screen, she is working on the operator in the blue box, and can see Bob working on another operator in the pink box. For Bob, he also sees the operator he is working on in the blue box, and can see Alice's operator in the pink box.

Real-time schema synchronization. To enable a real-time shared-editing experience, it is critical for collaborators to be always "on the same page" in terms of the visible workflows in their frontends. To do so, whenever a user edits part of the workflow, this change should be immediately reflected in the frontends of other collaborators. For example, while Alice is working on the UDF, she adds a new column `geo-state` (Figure 3, upper left). Not only does Alice's frontend update its downstream operators' input schemas, Bob will also immediately notice in his filter operator that there is a new column as an auto-complete option (Figure 3, upper right).

Figure 3 shows how Texera supports real-time schema synchronization among the collaborators. The synchronization manager monitors editing changes from all users in the session. In step (1), Alice edits the UDF operator, and its output schema is updated with a new column `geo-state`. The change is sent to the synchronization manager, which propagates the change to Bob's web UI to keep both UIs synchronized. This is not shown on the figure for brevity. Next in step (3), the synchronization manager invokes the workflow analyzer with the updated workflow. In step (4), the workflow analyzer computes the updated schemas. In step (5), the new schemas of the operators are sent to all the collaborators, and each web UI updates its auto-complete options accordingly.

3.2 Collaborative Workflow Execution

After constructing the workflow together, Alice and Bob need to test the workflow and debug possible errors before applying it on the whole large dataset. Texera's built-in collaborative execution capabilities ensure a unified experience for the members.

Shared monitoring of progressive execution. After Bob hits the "Run" button, both collaborators see the workflow execution in real-time. Not only can they see runtime statistics of each operator (i.e., processing speed and number of processed tuples), they can also see progressive visualization. For instance, for the ML training operator, it is very natural to see the training progress visually. Texera allows seeing the training statistics graph updated periodically. For example, Alice and Bob can see plots on accuracy and loss over epochs, and the plots are updated dynamically so that collaborators can monitor the progress and make decisions in real-time.

Figure 4 (I) shows how Texera supports shared execution monitoring. The Amber engine monitors the state of the workflow, collects statistics from operators, and periodically pushes the latest state updates to the state manager, which broadcasts the updates to all web UIs in the execution session. Amber supports progressive computation of the workflow. Each operator performs incremental computation on its input data and pushes the incremental output results downstream. In Figure 4 (I), the sink operators periodically push the newly generated result incrementally (instead of the full snapshot) to the result manager. The result manager broadcasts the incremental results to each web UI, which displays the new result and updates the visualization. The execution results could be potentially too large and cannot fit in one machine. To solve the problem, the sink operators push incremental results to a distributed result storage, and notifies the result manager the updated metadata such as number of tuples. The result manager keeps a reference to the result storage. When the user checks the results, the result manager asynchronously queries the storage to fetch the data.

Shared execution control. In Texera, each collaborator with write access has the privilege to pause and resume the execution, and the action will immediately be visible to other users. For instance, after monitoring the training graph for a while, Alice notices that the accuracies have not improved. So she pauses the execution, and Bob immediately sees on his browser that the training is paused.

Figure 4 (II) shows how Texera supports shared execution control. In step (1), Alice sends a Pause request to the Amber engine through the shared execution manager. After all operators are paused, Amber sends a Paused notification to the state manager in step (2). The state manager updates the current state, and broadcasts the updated state to all the web UIs in step (3). Then every collaborator sees the

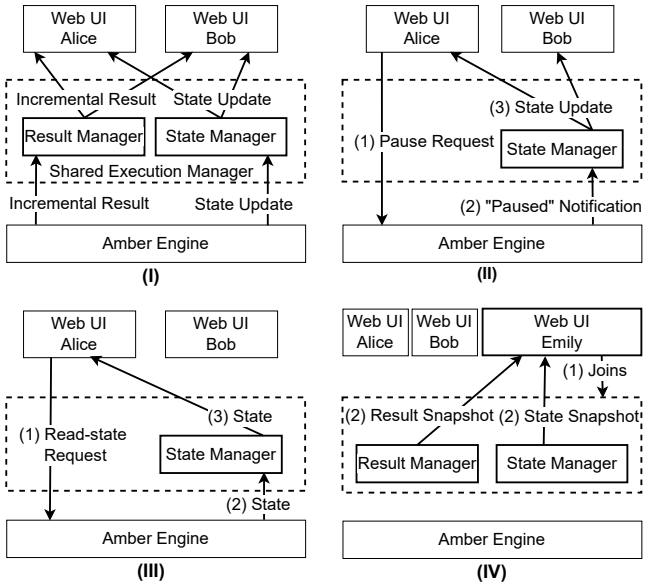

Figure 4: Collaborative execution in Texera. (I) The web server periodically pushes workflow status updates and new incremental results to all web UIs. (II) When a user pauses the workflow, the workflow state of all web UIs are updated. (III) A response to a request of investigating a state is sent back to the request initiator only. (IV) When a new user joins the execution session, the web server sends the state snapshot and result snapshot to bring the new user up-to-date.

workflow's execution state changed to "Paused." They can inspect and change the workflow, and resume or rerun the workflow using the new logic. If any editing conflict happens, it will be resolved before the execution resumes.

Independent runtime inspection. Texera allows the collaborators to view details of the paused workflow. For instance, the ensemble's loss is the sum of each model's loss. Alice takes advantage of Python UDF's "evaluate expression" capability to see each sub-model's loss. She notices that one model involving the "user followers count" feature is the culprit since it has an abnormally high loss. She then asks Bob to reconsider each feature. The two collaborators can independently inspect the current workflow state without affecting each other. Alice continues verifying the metric details and her UDF script. Meanwhile, Bob analyzes a few result tuples of the feature selection operator to make improvements. In general, when a user pauses the workflow, every collaborator can see the effect of the workflow execution being paused. When different users investigate the internal state of operators, each of them has their independent view of the shared workflow state.

Figure 4 (III) shows how Texera supports independent runtime inspection. In step (1), Alice sends a Read-state request to Amber through the shared execution manager. The engine retrieves the state from the target operator and notifies the state manager in step (2). The state manager internally keeps a view state for each user connected to the execution session. It updates the view state of Alice, and sends the state response back to only Alice in step (3).

Adding a new collaborator. There is no limit on the number of concurrent users in one workflow, so more collaborators can join and share their expertises. After correcting the feature selection and fixing a few bugs in the training script, the two collaborators rerun the training, and find out that the model is still unsatisfactory. They again pause the workflow. This time Alice invites her colleague Emily to help. Emily is added as a new collaborator and joins the debugging session. Now everything Alice and Bob see are also available to Emily, including the workflow and the execution state. Emily can also inspect the training metrics in detail, and evaluate expressions at exactly the same execution state as Alice and Bob. Together with Emily, they fix a problem with the script. They eventually train a good model, and apply it on the processed inference data so that Bob can perform further analysis.

Figure 4 (IV) shows how Texera supports adding new collaborators to a workflow execution session. The state manager maintains the current state snapshot of the execution. Whenever the state manager receives a state update from the Amber engine, the update is applied to the current state snapshot. Similarly, the result manager maintains the current result snapshot. When the result manager receives new incremental results from Amber, it merges the incremental update with the existing result to create a new result snapshot. When a new user such as Emily joins the execution session, the latest state snapshot and result snapshot are sent to Emily's web UI to bring her up-to-date with the current execution progress. This architecture also allows an existing user to safely leave the execution session and later reconnect to the execution.

3.3 Demonstration Plan

In the demonstration, we will provide a large twitter dataset and several COVID-19 datasets. We will provide several pre-constructed workflows over these datasets with various topics, such as social media analysis and COVID-19 data analysis. The audience can connect to a hosted Texera cloud service to experience the aforementioned collaborative features. Multiple users can construct and co-edit a workflow in real-time using the collaborative editing features. The audience can also run these workflows on a cluster, monitor the execution state, and collaboratively interact with the execution.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under the awards III 1745673 and III 2107150.

REFERENCES

[1] Alteryx 2022. https://www.alteryx.com/. last accessed: 2022-07-13.
[2] Einblick 2022. https://www.einblick.ai/. last accessed: 2022-07-13.
[3] Knime 2022. https://www.knime.com/. last accessed: 2022-07-13.
[4] Knime: How to collabrate using Knime Server 2022. https://www.knime.com/sites/default/files/2022-02/Knime%20Server%20How%20To.pdf. last accessed: 2022-07-13.
[5] Avinash Kumar, Zuozhi Wang, Shengquan Ni, and Chen Li. 2020. Amber: A Debuggable Dataflow System Based on the Actor Model. *Proc. VLDB Endow.* 13, 5 (2020), 740–753.
[6] RapidMiner 2022. https://rapidminer.com/. last accessed: 2022-07-13.
[7] Zuozhi Wang, Flavio Bayer, Seungjin Lee, Kishore Narendran, Xuxi Pan, Qing Tang, Jimmy Wang, and Chen Li. 2017. A Demonstration of TextDB: Declarative and Scalable Text Analytics on Large Data Sets. In *ICDE 2017*. 1403–1404.
[8] Zuozhi Wang, Avinash Kumar, Shengquan Ni, and Chen Li. 2020. Demonstration of Interactive Runtime Debugging of Distributed Dataflows in Texera. *Proc. VLDB Endow.* 13, 12 (2020), 2953–2956.