Building a Collaborative Data Analytics System: Opportunities and Challenges

Zuozhi Wang
UC Irvine
Irvine, CA
zuozhiw@ics.uci.edu

Chen Li
UC Irvine
Irvine, CA
chenli@ics.uci.edu

ABSTRACT

Real-time collaboration has become increasingly important in various applications, from document creation to data analytics. Although collaboration features are prevalent in editing applications, they remain rare in data-analytics applications, where the need for collaboration is even more crucial. This tutorial aims to provide attendees with a comprehensive understanding of the challenges and design decisions associated with supporting real-time collaboration and user interactions in data analytics systems. We will discuss popular conflict resolution technologies, the unique challenges of facilitating collaborative experiences during the workflow construction and execution phases, and the complexities of supporting responsive user interactions during job execution.

PVLDB Reference Format:
Zuozhi Wang and Chen Li. Building a Collaborative Data Analytics System: Opportunities and Challenges. PVLDB, 16(12): 3898 - 3901, 2023.
doi:10.14778/3611540.3611580

1 INTRODUCTION

Real-time collaborative editing services, such as Google Docs and Overleaf, have gained significant popularity and brought immense value to society. These services enable individuals to easily collaborate and jointly contribute to various tasks, including document editing, spreadsheet management, presentations, and drawings. The benefits of these services have become more appealing following the recent shift toward remote work. Although real-time collaboration becomes increasingly prevalent, it remains a rare feature in data-analytics applications. The need for collaboration features is arguably even more crucial in data-analytics applications, particularly with the growing involvement of domain scientists in the process. Domain experts, such as public health scientists and medical researchers, are vital in the context of data analytics because their valuable domain knowledge can unlock the full potential of data-driven insights. However, domain experts often lack technical skills required to analyze large datasets, and they need collaboration with technical experts who have the technical abilities but may not have the relevant domain knowledge. Jupyter Notebook [9], one of the most widely used platforms for data analytics in Python, has real-time collaborative editing and execution of notebooks as a highly appreciated feature from the community [7].

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611580

Supporting real-time collaboration in data analytics systems is not trivial. For example, it took Jupyter Notebook nearly a decade to add collaboration support to the open-source version [13]. One challenge stems from the complexities of supporting real-time collaborative editing in the frontend. Fortunately, recent advances and maturity in frontend JavaScript technologies have made it increasingly easier to support shared editing. Numerous open-source libraries now provide such capabilities [3, 4, 15, 20]. As a result, it is now more feasible than ever to integrate collaboration support into data analytics systems, enhancing the user experience and promoting a more efficient teamwork in data-driven tasks.

A main distinction between supporting real-time collaboration in document editors and data analytics systems is that collaboration must occur not only during editing but also throughout the execution process. Data analytics jobs processing large datasets can be time-consuming, leaving users without feedback on results until the very end. This lack of real-time visibility can impede collaboration, as users cannot view results, identify workflow issues, or implement quick fixes and iterations to improve the data workflows. As a result, it is essential for data analytics systems to facilitate user interaction during execution. These systems should support progressive computation and deliver real-time updates on execution status and early results. Additionally, they should allow users to pause or resume execution, examine execution status, inspect intermediate states and results, and modify workflows as needed. All execution states must be shared among users, allowing a new user to join an execution session at any moment.

This tutorial aims to equip attendees with a solid understanding of the challenges and design decisions associated with supporting real-time collaboration and user interactions in data analytics systems. The tutorial will be a 1.5-hour lecture-style presentation. We will explain and compare popular conflict resolution technologies, delve into the unique challenges of facilitating collaborative experiences during the workflow construction and execution phases, and explore the complexities of supporting responsive user interactions during a job execution.

Target Audience. This tutorial is designed for researchers, practitioners, and data scientists working in the field of large-scale data analytics who have interest in real-time collaboration and interactive data analysis. The target audience also include those involved in the development of data analytics platforms or those seeking to enhance the collaborative capabilities of existing systems. We assume that attendees have an understanding of general principles in database and data analytics concepts. No specific expertise in real-time collaboration or data analytics system design is required, as the tutorial will provide a comprehensive overview of relevant concepts, challenges, and design tradeoffs.

Related Work. The growing demand for real-time collaboration among data analysts has led to the emergence of various commercial Python notebook-style offerings that provide this functionality, such as DeepNote [2], Google Colab [5], and Databricks Notebooks [6]. These tools are useful for users proficient in programming languages like Python and SQL. Big data systems, such as Spark [22], Flink [11], and Dask [17], have gained popularity among software developers due to their ability to efficiently process large-scale datasets in distributed environments. Although these tools work great for technical experts, they are less accessible to non-technical users who lack programming skills. These systems also do not support built-in collaboration features. Workflow-style systems are popular among domain scientists due to their easy-to-understand graphical user interface. Despite their popularity, many workflow-based data analytics systems, such as Alteryx [1], KNIME [8], and RapidMiner [10], have difficulty offering collaboration capabilities, as they are often built with a "pre-cloud" architecture and require users to install desktop software or clients. Einblick [12] offers an interactive canvas supporting collaborative editing of workflows, with a backend execution engine based on the dataflow model [19]. However, the system is not open source, and there is no published work describing its collaboration-related features. Texera [14] is an open-source workflow model-based system that enables scalable computation and delivers a collaborative and interactive user experience through its graphical user interface. Throughout this tutorial, we will use Texera as an example platform to explain concepts and challenges related to collaborative editing and execution, although the principles can be broadly applied.

This tutorial has never been offered elsewhere before, making it a unique learning opportunity for attendees. Furthermore, to our best knowledge, there has been no tutorial offered on the topic of real-time collaboration in data analytics systems at major data-centric research conferences in recent years.

2 TUTORIAL OUTLINE

In the introductory part of this tutorial, we begin by providing a brief overview of the significance of real-time collaboration in data analytics systems. We also discuss the current state of collaboration in popular data analytics tools and platforms. Furthermore, we give a live demonstration of various collaborative data analytics features using the Texera system, as shown in Figure 1. After the introduction, we will cover the following topics.

2.1 Supporting a Collaborative User Interface for Workflow Construction

We give overview of the two most popular real-time conflict resolution technologies, Operational Transformation (OT) and Conflict-free Replicated Data Types (CRDT). We compare their respective pros and cons and discuss the possible architectures for implementing each design in a data processing system. Additionally, we discuss how to offer smart auto-completion for data types and column names, which requires a workflow compiler in the backend to analyze the workflow in real-time to provide these features.

Operational Transformation [18] (OT) is an approach that enables real-time collaborative editing by transforming operations in such a way that they maintain consistency among different replicas

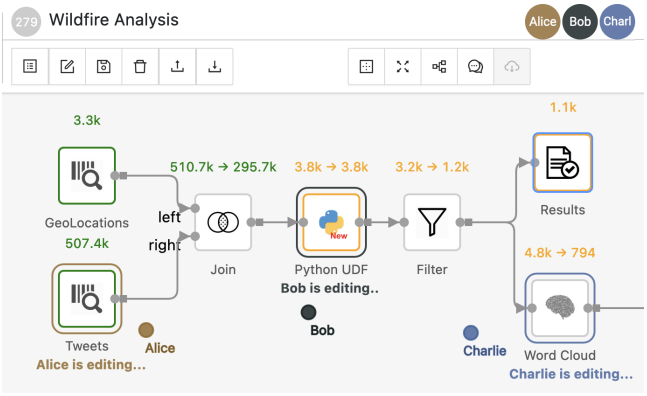

Figure 1: Illustration of collaborative data analytics in a workflow system. Alice, Bob, and Charlie are collaborators, and each contributes to different components of an active workflow process.

of shared data. In OT, when a user performs an action, this action is transformed with respect to other concurrent operations by other users before it is applied to the shared data. This transformation ensures that the order of operations does not matter, and the final state of the shared data remains consistent across all replicas. In a system using OT, each user's frontend client maintains a local copy of the shared data and communicates with a central server for synchronization. In an OT-based system, the central server plays a crucial role in maintaining consistency among clients by transforming and coordinating operations. The server can host a workflow compiler that listens to changes in the shared data, providing smart auto-complete and suggestions to the frontend clients. As the server maintains the ground truth of the data, it can provide accurate auto-complete suggestions to all its connected clients.

Conflict-free Replicated Data Types [16] (CRDT) is an alternative approach for maintaining consistency in distributed systems. CRDTs are data structures designed to be replicated across multiple nodes while allowing concurrent updates without coordination between the nodes. The main idea behind CRDT is that it should always be possible to merge different replicas of the shared data into a consistent state, even when updates happen concurrently. This is achieved by ensuring that all operations on the CRDT are commutative, associative, and idempotent. In a system using CRDT, clients can update their local copies without the need for coordinating with a central server, such as using a peer-to-peer communication with other clients. Since CRDTs allow for concurrent updates without coordination, it is more challenging for the compiler inside the server since the server does not have the ground truth.

The implementations of OT and CRDT mainly differ in their system architecture and coordination requirements. Although an OT-based implementation might be more suitable for a data analytics system that typically has an existing central server, CRDT-based implementations are still very compelling due to their better ecosystem and richer frontend support. In this tutorial, we will cover how both architectures can be integrated into data analytics systems.

2.2 Supporting Responsive User Interactions During Execution

In this part of the tutorial, we focus on how to pause the execution of an operator, which is a critical aspect of facilitating user interactions with the system. After users pause the execution of an operator, the system allows them to perform additional interactions, such as checking the execution status, inspecting intermediate states and results, and modifying the workflow. We discuss how the system can be still responsive to user interaction requests instantaneously, within sub-seconds. We will first delve into how to support pausing and resuming, then discuss how to support other user interactions.

We first discuss methods that rely on external entities to forcibly interrupt and halt the execution of a running program. We examine these methods at two levels: the operating system (OS) level, through stopping a process, and the Java Virtual Machine (JVM) level, through suspending a thread. The OS-level process interruption method utilizes the `SIGSTOP` signal in Unix-based systems to immediately halt the execution of an operator, and the process remains suspended until a `SIGCONT` signal is received. This method allows for quick and efficient pausing and resuming of a program, has native OS support, and is widely applicable to various programs. However, a drawback is that the program is frozen and cannot respond to user requests, which makes it challenging to access the program's state after pausing. JVM-level thread interruption uses Java's built-in `Thread.suspend()` method to pause an operator's execution by employing two threads: a data-processing thread and a control handling thread. While the data-processing thread is suspended, the control-handling thread can still access the program state using shared variables and continue to handle user requests. One drawback of this method is that the operator's processing logic might be interrupted at an arbitrary point, potentially leaving data structures in an inconsistent state and making it challenging to reason about the application's state.

An alternative design is to let data processing systems voluntarily suspend their execution at predetermined and predictable points, rather than being forcibly interrupted. This design takes advantage of natural stopping points in data processing systems, such as between the processing of two tuples. The operator checks for a pause signal before processing the next tuple and, if the pause signal is detected, the operator voluntarily suspends its own execution. This allows for user interactions and inspections during the pause. However, this method has a drawback: processing a single tuple might be time-consuming for complex user-defined functions (UDFs), which could reduce system responsiveness. To address this issue, one can further divide an operator's processing of a single tuple into multiple mini-steps, each performing a part of the computation logic. This approach transforms the operator logic into a state machine, which tracks the upcoming execution step.

In this tutorial, we will discuss and compare the aforementioned methods for pausing and resuming the execution of a workflow. We will provide detailed specifications on how a data-processing engine can be designed to support pausing for each method, and discuss their advantages, drawbacks, and suitability for different use cases. Furthermore, we will extend the design of supporting pausing to support various types of user interactions, such as inspecting internal states and intermediate results, and modifying logic.

2.3 Supporting Collaborative Execution Monitoring and Control

In this part of the tutorial, we discuss how to share execution states in a collaborative data analytics setting. In an interactive data processing system, there is a rich set of execution states to be shared across all users, such as the status of operators (e.g., initializing, running, paused, error, or completed), runtime statistics of operators (input and output counts), user interaction history (e.g., user commands, system responses, error messages, logs, and traces), and progressive execution results. When a user starts running a workflow, other users sharing the same workflow should see it as running and receive constant updates. Furthermore, a user should be able to invite a new collaborator to join an execution session at any time, to help analyze results or investigate issues. In this case, the execution state should be seamlessly shared with the new collaborator's frontend UI. In other words, any user can freely detach and attach to the shared execution state at any time.

A naive approach of periodically sending the entire execution state snapshot to the frontend is not viable due to the frontend's limited processing power and its network overhead. Additionally, the interaction history keeps growing as more user interactions occur during execution, and progressive execution results can become large. To provide a smooth user experience, we need incremental updates for various execution states. The challenge lies in the fact that different execution states require different incremental methods to be updated to the frontend. For instance, for workflow state and operator status, the backend sends only the status of changed operators. For the interaction history, the newly generated entries are sent to the frontend, which appends them to the history list. For sharing progressive execution results, the frontend receives metadata updates and only fetches results on demand.

However, if we only provide incremental updates to clients, a problem arises when a new user joins an execution session in the middle of an execution. If the new user's frontend only receives incremental updates from the moment they join, their frontend state will be incomplete. For example, the new user might only see interactions made after they join, and not be able to view past interaction histories, which could be critical for investigating problems and analyzing results. To address this problem, the server also maintains an execution state, and updates it using the same methods as the clients. When a new user joins, they are first fast-forwarded to the latest state using the server-maintained state. Afterward, all subsequent incremental updates are directed to the new user, ensuring they have a complete and up-to-date view of the execution state. In this tutorial, we explore best practices for delivering incremental updates of various execution states to the frontend.

2.4 Supporting Scalable and Fault-Tolerant Executions with Frequent User Interactions

Supporting fault tolerance is crucial in large-scale data-processing systems, with all major big data platforms considering it a key requirement. However, when real-time collaboration and user interactions are introduced, ensuring fault tolerance becomes increasingly complex. A common approach adopted in existing systems such as Apache Spark, Hadoop, and Flink is to roll back to the last checkpoint and rerun the computation [21]. This approach works well

when data processing logic does not involve user interactions since queries are often deterministic, and rerunning them produces the same result. However, when user interactions are involved, merely rerunning the data computation without accounting for these interactions can result in the loss of user interaction information, leading to incorrect outcomes.

To address this problem, we need to carefully reapply user control commands during the recovery phase. Specifically, we must log all non-deterministic factors, such as user interaction timing and control command content, which effectively tackles the issues with fault tolerance in systems involving user interactions. Logging-based fault tolerance is widely discussed in the literature, and in this tutorial, we will explore a few approaches that are most suitable for supporting fault tolerance in the context of collaborative and interactive data analytics systems. One approach is to log all input messages received by an operator. During recovery, we will replay the same sequence of messages to each operator, which leads to the same processing and identical output. This approach is simple and fast in recovery, but has high runtime overhead. An alternative approach is to only log the arrival order of input messages and rely on upstream operators to regenerate message content. This method reduces the log size; however, in certain cases with complex user interaction patterns, it may result in cascading rollbacks with increased recovery time. Finally, we discuss a hybrid approach that selectively logs the content of messages based on their characteristics.

3 PRESENTERS

Zuozhi Wang is a PhD student at UC Irvine focusing on distributed big data processing and query optimization. Over the last six years, Zuozhi has dedicated his research to building collaborative and interactive data analytics systems on the Texera project. His knowledge and expertise span across various layers of the system, including the distributed dataflow execution engine, RPC services, fault tolerance, compiler, web services, and frontend. With his hands-on experience in Texera, he is well-equipped to address challenges and design decisions related to real-time collaborations and user interactions in data analytics systems, providing attendees with valuable insights and practical understanding in this tutorial.

Chen Li is a professor in the Department of Computer Science at UC Irvine. He received his PhD degree in Computer Science from Stanford University, and his MS and BS degrees in Computer Science from Tsinghua University, China. He was a recipient of an NSF CAREER award and several test-of-time publication awards, a part-time visiting research scientist at Google, PC co-chair of VLDB 2015, General Chair of ICDE 2023, an ACM distinguished member, and an IEEE fellow. He is the treasurer and a board member of the VLDB Endowment. His research interests are in the field of data management, including data-intensive computing, databases, query processing and optimization, machine learning-based systems and data analytics, search, visualization. His current focus is building open source systems for big data management and analytics.

ACKNOWLEDGMENTS

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

REFERENCES

[1] [n.d.]. Data Science and Analytics Automation Platform | Alteryx — alteryx.com. https://www.alteryx.com/. [Accessed 17-Apr-2023].
[2] [n.d.]. Deepnote: Analytics and data science notebook for teams. — deepnote.com. https://deepnote.com/. [Accessed 17-Apr-2023].
[3] [n.d.]. GitHub - microsoft/FluidFramework: Library for building distributed, real-time collaborative web applications — github.com. https://github.com/microsoft/FluidFramework. [Accessed 18-Apr-2023].
[4] [n.d.]. GitHub - share/sharedb: Realtime database backend based on Operational Transformation (OT) — github.com. https://github.com/share/sharedb. [Accessed 18-Apr-2023].
[5] [n.d.]. Google Colab — research.google.com. https://research.google.com/colaboratory/faq.html. [Accessed 17-Apr-2023].
[6] [n.d.]. Introduction to Databricks notebooks | Databricks on AWS — docs.databricks.com. https://docs.databricks.com/notebooks/index.html. [Accessed 17-Apr-2023].
[7] [n.d.]. Notebook feature requests · Issue 977 · ipython/ipython — github.com. https://github.com/ipython/ipython/issues/977#issuecomment-5559489. [Accessed 17-Apr-2023].
[8] [n.d.]. Open for Innovation | KNIME — knime.com. https://www.knime.com/. [Accessed 17-Apr-2023].
[9] [n.d.]. Project Jupyter — jupyter.org. https://jupyter.org/. [Accessed 18-Apr-2023].
[10] [n.d.]. RapidMiner | Amplify the Impact of Your People, Expertise Data — rapidminer.com. https://rapidminer.com/. [Accessed 17-Apr-2023].
[11] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink™: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.* 38, 4 (2015), 28–38. http://sites.computer.org/debull/A15dec/p28.pdf
[12] @EinblickAI. [n.d.]. Multiplayer Python Notebooks on an Interactive Canvas — einblick.ai. https://www.einblick.ai/. [Accessed 17-Apr-2023].
[13] Kevin Jahns. [n.d.]. How we made Jupyter Notebooks collaborative with Yjs — blog.jupyter.org. https://blog.jupyter.org/how-we-made-jupyter-notebooks-collaborative-with-yjs-b8dff6a9d8af. [Accessed 17-Apr-2023].
[14] Xiaozhen Liu, Zuozhi Wang, Shengquan Ni, Sadeem Alsudais, Yicong Huang, Avinash Kumar, and Chen Li. 2022. Demonstration of Collaborative and Interactive Workflow-Based Data Analytics in Texera. *Proc. VLDB Endow.* 15, 12 (2022), 3738–3741. https://www.vldb.org/pvldb/vol15/p3738-liu.pdf
[15] Petru Nicolaescu, Kevin Jahns, Michael Derntl, and Ralf Klamma. 2015. Yjs: A Framework for Near Real-Time P2P Shared Editing on Arbitrary Data Types. In *Engineering the Web in the Big Data Era - 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings (Lecture Notes in Computer Science)*, Philipp Cimiano, Flavius Frasincar, Geert-Jan Houben, and Daniel Schwabe (Eds.), Vol. 9114. Springer, 675–678. https://doi.org/10.1007/978-3-319-19890-3_55
[16] Nuno M. Preguiça, Carlos Baquero, and Marc Shapiro. 2019. Conflict-Free Replicated Data Types CRDTs. In *Encyclopedia of Big Data Technologies*, Sherif Sakr and Albert Y. Zomaya (Eds.). Springer. https://doi.org/10.1007/978-3-319-63962-8_185-1
[17] Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *Proceedings of the 14th Python in Science Conference 2015 (SciPy 2015), Austin, Texas, July 6 - 12, 2015*, Kathryn Huff and James Bergstra (Eds.). scipy.org, 126–132. https://doi.org/10.25080/Majora-7b98e3ed-013
[18] Kumawat Santosh and Ajay Khunteta. 2010. A Survey on Operational Transformation Algorithms: Challenges, Issues and Achievements. *International Journal of Computer Applications* 3 (07 2010). https://doi.org/10.5120/787-1115
[19] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Philipp Eichmann, Navid Karimeddiny, Charlie Meyer, Wesley Runnels, and Tim Kraska. 2021. Davos: A System for Interactive Data-Driven Decision Making. *Proc. VLDB Endow.* 14, 12 (2021), 2893–2905. https://doi.org/10.14778/3476311.3476370
[20] Peng Yin, Haowen Lai, Shiqi Zhao, Ruijie Fu, Ivan Cisneros, Ruohai Ge, Ji Zhang, Howie Choset, and Sebastian A. Scherer. 2022. AutoMerge: A Framework for Map Assembling and Smoothing in City-scale Environments. *CoRR* abs/2207.06965 (2022). https://doi.org/10.48550/arXiv.2207.06965 arXiv:2207.06965
[21] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, 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 Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, Steven D. Gribble and Dina Katabi (Eds.). USENIX Association, 15–28. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia
[22] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010*, Erich M. Nahum and Dongyan Xu (Eds.). USENIX Association. https://www.usenix.org/conference/hotcloud-10/spark-cluster-computing-working-sets