Building a Collaborative Data Analytics System: Opportunities and Challenges

Zuozhi Wang and Chen Li
Tutorial Outline

1. Overview of collaborative data analytics

2. Challenges and solutions:
   a. Shared workflow editing
   b. Shared workflow execution
   c. Interacting with runtime execution
   d. Runtime co-debugging on Python UDFs

3. Open challenges
Tutorial Outline

1. → Overview of collaborative data analytics (Chen)

2. Challenges and solutions:
   a. Shared workflow editing (Chen)
   b. Shared workflow execution (Zuozhi)
   c. Interacting with runtime execution (Zuozhi)
   d. Runtime co-debugging on Python UDFs (Zuozhi)

3. Open challenges (Chen and Zuozhi)
Popularity of Collaboration Cloud Services

Benefits:

- Cloud services
- Shared editing
- Version control
- Sharing
Python Notebooks

- Support real-time collaborations
- For programmers

```
column_names=['monthlycharges', 'tenuremonths', 'contract', 'churnvalue']
where_filter='[DSL, 'Fiber optic']
```
Big Data Processing Systems

Efficient and scalable
Big Data Processing Systems

- Start to support collaborations
- For SQL experts

Databricks notebooks
Need of Multi-disciplinary Collaborations

- Domain experts:
  - Rich domain knowledge
  - Limited IT/coding skills

- IT experts
  - Limited domain knowledge
  - Strong coding skills
Workflow Systems

Pros:
- Easy-to-use interface
- No coding required

Cons:
- Pre-cloud architecture
- No collaboration features
- Limited scalability
ChatGPT Code Interpreter

- Easy to use, no programming
- Integratable into a workflow system

Python code snippet:

```python
import matplotlib.pyplot as plt

# Group tweets by city location
location_counts = {}
for tweet in tweets:
    city = tweet['geo']['name']
    location_counts[city] = location_counts.get(city, 0) + 1

# Sort by count
sorted_locations = sorted(location_counts.items(), key=lambda x: x[1], reverse=True)
```

Can you draw me a bar chart showing the count of tweets group by city location.
System Requirements for Collaborations

1. GUI-based workflows for non-IT people
2. Collaborative editing
3. Interactions during runtime
4. Supporting multiple languages: Python, R, …
5. Supporting machine learning (training, inference, …)
6. Scalable!
A system for collaborative data analytics

- Started in 2016; open source
- Used by many research projects
- Powerful features:
  - Cloud services
  - Version control
  - Shared editing
  - Commenting
  - Sharing
  - …
Demo!
Collaboration Feature: Version Control

CSV File Scan → Keyword Search → Regular Expression → Python Lambda Function → View Results

Version Diff

Historical Version List

<table>
<thead>
<tr>
<th>Version#</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>08/17/2023 02:57:54 GMT-7</td>
</tr>
<tr>
<td>14</td>
<td>08/17/2023 02:57:53 GMT-7</td>
</tr>
<tr>
<td>13</td>
<td>08/17/2023 02:57:52 GMT-7</td>
</tr>
<tr>
<td>12</td>
<td>08/17/2023 02:57:51 GMT-7</td>
</tr>
<tr>
<td>11</td>
<td>08/17/2023 02:57:50 GMT-7</td>
</tr>
<tr>
<td>10</td>
<td>08/17/2023 02:57:49 GMT-7</td>
</tr>
<tr>
<td>9</td>
<td>08/17/2023 02:57:47 GMT-7</td>
</tr>
</tbody>
</table>
Collaboration Feature: Resource Sharing
Collaboration feature: commenting

Zuozi Wang
08/17/2023, 03:12:41 AM GMT-7
@Chen can you take a look at this Python code?

add new comment

OK
High School Students Using Texera!
Tutorial Outline

1. Overview of collaborative data analytics (Chen)

2. Challenges and solutions:
   a. → Shared workflow editing (Chen)
   b. Shared workflow execution (Zuozhi)
   c. Interacting with runtime execution (Zuozhi)
   d. Runtime co-debugging on Python UDFs (Zuozhi)

3. Open challenges (Chen and Zuozhi)
Goal: enabling shared editing similar to Google Docs

Lorem ipsum

Workflow shared editing
Shared Editing

Alice

Bob

U C I

U C I

Building a Collaborative Data Analytics System. Zuozhi Wang and Chen Li
Two users doing concurrent edits

Alice

Bob

UCXI

UCYI
Shared Editing: Good (same result)
Shared Editing: Also good (same result)

Alice

Bob

UCYXI

UCYXI

✓
Shared Editing: Bad (different results)
Shared Editing: How to determine an order

1st method: Operational Transformation (OT)

- Using a central server
R₀: insert(“X”, pos=2)

R₀: insert(“Y”, pos=2)
Received Alice’s request
Update R0 to R1

Propagate Alice’s Edit
Received Bob’s request on R0: `insert(“Y”, pos=2)`
on R1: `insert(“Y”, pos=3)`
Final Result

- **Central Server**
  - Document: UCXYI
  - Operations: 

- **Alice**
  - UCXYI

- **Bob**
  - UCXYI
Shared Editing: How to Determine Order

2nd method: Conflict-free Replicated Data Type (CRDT)

- Peer-to-peer communication
- Everyone follows the same policy.
CRDT (Conflict-free Replicated Data Type)

recent sequence #

Alice

Edit by Alice

ID of “C”

6A 7A 8A

U C I

Bob

6A 7A 8A

U C I

Building a Collaborative Data Analytics System. Zuozhi Wang and Chen Li
CRDT: concurrent edits

Alice

9A: { 7A, ins(“X”) }
new id  pos  operation

6A  7A  9A  8A

UCX I

Bob

9B: { 7A, ins(“Y”) }
new id  pos  operation

6A  7A  9B  8A

UCY I
CRDT: transmitting edits to other users

9A: \{ 7A, \text{ins}("X") \}  
9B: \{ 7A, \text{ins}("Y") \}

6A 7A 9A 8A  
U C X I  
6A 7A 9B 8A  
U C Y I
CRDT: independently comparing edits

Alice

9B: { 7A, ins(“Y”) }

9A > 9B

6A 7A 9A X 8A

U C X I

Bob

9A: { 7A, ins(“X”) }

9A > 9B

6A 7A 9B 8A

U C Y I
CRDT: following same policy to decide

Alice

Bob

6A 7A 9A 9B 8A
U C X Y I

6A 7A 9A 9B 8A
U C X Y I
## Comparison of the Two Methods

<table>
<thead>
<tr>
<th></th>
<th>Central server</th>
<th>Implementation complexity</th>
<th>Performance</th>
<th>Open-source ecosystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>OT</td>
<td>Yes</td>
<td>Complex</td>
<td>Good</td>
<td>Less rich</td>
</tr>
<tr>
<td>CRDT</td>
<td>No</td>
<td>Simple</td>
<td>Acceptable</td>
<td>Rich</td>
</tr>
</tbody>
</table>
Workflow Shared Editing

CRDT “documents” are:

- Workflow DAG
- Properties
Adding Two Operators Concurrently: no conflicts
Adding Two Operators Concurrently: no conflicts
Moving Operators to Different Positions: conflicts

CRDT resolves it based on user ID

Alice

Bob
Changing Operator’s Properties: conflicts

CRDT resolves it based on user ID

**Regular Expression**

- Case Insensitive *
- Regex match is case sensitive
- Attribute *
- Column to search regex on
- Regex *
- Fool

**Regular Expression**

- Case Insensitive *
- Regex match is case sensitive
- Attribute *
- Column to search regex on
- Regex *
- Bar

Alice  Bob
Tutorial Outline

1. Overview of collaborative data analytics (Chen)

2. Challenges and solutions:
   a. Shared workflow editing (Chen)
   b. Shared workflow execution (Chen and Zuozhi)
   c. Interacting with runtime execution (Zuozhi)
   d. Runtime co-debugging on Python UDFs (Zuozhi)

3. Open challenges (Chen and Zuozhi)
Need of Shared Execution

If users have their own executions, then collaboration is not possible.
Benefits of Shared Execution

- View same results
- Shared control
- Co-debugging
- ……
Benefits of Shared Execution (cont.)

Invite new collaborators at runtime

Shared Execution State
Shared Execution: Demo
Workflow Status: Running/Paused/Failed/Completed
Operator Status: Input/Output Tuple Count

Completed Operators

Running Operators

Input #

Output #
Execution Results
User Interaction History

- self._context.operator_manager.operator.average_distance
- 155.3607987084146
- self._context.tuple_processing_manager.current_input_tuple
- Tuple['id': 'e1e35d357ceefb52', 'long_low': -118.4019312, 'lat_low': -300.0, 'long_high': -118.352695, 'lat_high': -300.0, 'id#1': 1471642336094396417, 'text': 'My number 1 motivation to stop climate change is so that I can make my son Cesar so he can be CP3']
Shared Execution: How to Share?

Execution State
Approach 1: Periodically Broadcast State to All Frontends

Alice

State

Bob
Approach 1: Periodically Broadcast State to All Frontends

Adding a new user at runtime
Problems with Approach 1

- Interaction history: growing
- Execution results: large
Approach 2: Periodically Send Incremental Updates

Alice  Incremental Updates  Bob
Approach 2: Periodically Send Incremental Updates

An example

Before

After
Approach 2: Periodically Send Incremental Updates

Another example

Before

After
Approach 2: Periodically Send Incremental Updates

Adding a new user

Alice → Bob → Charlie

(1) Joins

(2) Incremental Updates
Problem with Approach 2 When Adding a New User

Incomplete state

CSV File Scan → Hash Join → Python UDF → View Results

58.0k → 58.0k → 57.6k
Problem with Approach 2 When Adding a New User

Incomplete interaction history

```
Tuple['id': 'e1e35d357ceefa52', 'long_low': -118.4019312, 'lat_low': -300.0, 'long_high': -118.352695, 'lat_high': -300.0, 'id#@1': 1471642336094396417, 'text': 'My number 1 motivation to stop climate change is so that I can make my son Cesar so he can be CP3']
```
Approach 3: Keep State and Send Incremental Updates

Alice  Incremental Updates  Bob

Execution state
Approach 3: Keep State and Send Incremental Updates

Adding a new user

Execution state
Tutorial Outline

1. Overview of collaborative data analytics (Chen)

2. Challenges and solutions:
   a. → Shared workflow editing (Chen)
   b. → Shared workflow execution (Chen and Zuozhi)
   c. → Interacting with runtime execution (Zuozhi)
   d. Runtime co-debugging on Python UDFs (Zuozhi)

3. Open challenges (Chen and Zuozhi)
Supporting Interactions at Runtime

Requirements of Pause:

- Fast
- Interact with an operator after pause
Approach 1: Using OS Signals

SIGSTOP (signal stop) / SIGCONT (signal continue)
Using OS Signals to Pause

User

```
(base) zuozhiw- $kill -SIGSTOP 39061
(base) zuozhiw- $
```

Operator

```
(base) zuozhiw- $java DataProcessor
Current Process ID (PID): 39061
Current progress: 20
Current progress: 40
Current progress: 60
Current progress: 80
Current progress: 100
Current progress: 120
Current progress: 140
Current progress: 160
zsh: suspended (signal) java DataProcessor
(base) zuozhiw- $
```
Using OS Signals to Resume

User

```bash
(base) zuozhiw-$ kill -SIGSTOP 39061
(base) zuozhiw-$ kill -SIGCONT 39061
(base) zuozhiw-$
```

Operator

```bash
Current progress: 40
Current progress: 60
Current progress: 80
Current progress: 100
Current progress: 120
Current progress: 140
Current progress: 160
zsh: suspended (signal) java DataProcessor
(base) zuozhiw-$Current progress: 180
Current progress: 200
Current progress: 220
Current progress: 240
Current progress: 260
Current progress: 280
```
Approach 1: Using OS Signals

Pros: 🎉
- Suspends immediately
- OS native support - little implementation effort

Cons: 😞
- Unable to interact with an operator
Approach 2: Using Thread-level Signal

In Java: `Thread.Suspend() / Thread.Resume()`
Approach 2: Using Thread-level Signal

Coordinator

Operator

Control Processing (CP) Thread

Data Processing (DP) Thread
Approach 2: Using Thread-level Signal

Pause operator execution
Approach 2: Using Thread-level Signal

Pause operator execution

Coordinator

CP Thread

Input tuples

DP Thread
Approach 2: Using Thread-level Signal

Pause operator execution

Coordinator

Pause

CP Thread

DPThread.Suspend()

DP Thread

Input tuples
Approach 2: Using Thread-level Signal

Inspect operator state
Approach 2: Using Thread-level Signal

Resume operator execution

Coordinator

Resume

CP Thread

DP Thread

Input tuples
Approach 2: Using Thread-level Signal

Pros:

- Suspends immediately
- Able to interact with operator after pause
Problem with Approach 2

Stopping point is arbitrary and incomprehensible to users.

Operator’s call graph

User’s code

“Deep” libraries
Approach 3: Checking for Pause between Tuples

Coordinator

CP Thread

Paused flag: false

DP Thread

Input tuples
Approach 3: Checking for Pause between Tuples

Coordinator

CP Thread

Paused flag: false

DP Thread

Input tuples
Approach 3: Checking for Pause between Tuples
Approach 3: Checking for Pause between Tuples

Coordinator

Pause

Set flag

CP Thread

DP Thread

Input tuples
Approach 3: Checking for Pause between Tuples

Coordinator

CP Thread

Paused flag: true

DP Thread

Input tuples

paused
Approach 3: Checking for Pause between Tuples

Pros: 🎉
- Meaningful stopping points

Cons: 😞
- Slower than approaches 1 and 2
- Runtime overhead of checking (10%)
## Comparison of three approaches

<table>
<thead>
<tr>
<th></th>
<th>Approach 1 (OS signals)</th>
<th>Approach 2 (Thread signals)</th>
<th>Approach 3 (Between tuples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pause Speed</td>
<td>Fast</td>
<td>Fast</td>
<td>OK</td>
</tr>
<tr>
<td>Interactions</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Meaningful Stopping Point</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Tutorial Outline

1. Overview of collaborative data analytics (Chen)

2. Challenges and solutions:
   a. → Shared workflow editing (Chen)
   b. → Shared workflow execution (Chen and Zuozhi)
   c. → Interacting with runtime execution (Zuozhi)
   d. → Runtime co-debugging on Python UDFs (Zuozhi)

3. Open challenges (Chen and Zuozhi)
User Defined Functions in Workflows
Runtime Co-debugging on Python UDFs

Data errors
- misformatted data
- unexpected nulls
- corrupted data
- ...

UDF errors
- corner cases
- wrong parameters
- ...

...
Co-debugging Demo

Domain experts need technical assistance in debugging.

Alice  Bob

Runtime Error
Approach 1: Attaching a Remote Debugger to a UDF
Approach 1: Attaching a Remote Debugger to a UDF
Approach 1: Attaching a Remote Debugger to a UDF

Pro: 😊

- Works out-of-the-box. (Adopted by PyFlink, PySpark)

Con: 😞

- Cannot co-debug: only a single debugger can be attached
Approach 2: Coordinator Controls Debuggers

Co-debugging: coordinator manages multiple debug frontends
Approach 2: Coordinator Controls Debuggers

Execution model
Approach 2: Coordinator Controls Debuggers

Setting a breakpoint
Approach 2: Coordinator Controls Debuggers

Hitting a breakpoint

![Diagram showing the interaction between different threads and the coordinator during debugging]
Approach 2: Coordinator Controls Debuggers

Resuming from the breakpoint
Approach 2: Coordinator Controls Debuggers

Pro: 😊 
- Supports co-debug

Con: 😞 
- High overhead (5x): always running in debug mode
Reducing Overhead: Invoke Debugger on Demand

Starting without a debugger
Reducing Overhead: Invoke Debugger on Demand

Attaching a debugger

![Diagram of debugger attach process]

- CP Thread
- SIGINT
- bp_info
- Debugger attached
- DP Thread
- Coordinator
- Operator
- UDF Line
- Debugger
## Adoption of Texera

<table>
<thead>
<tr>
<th>Deployments</th>
<th>Users</th>
<th>Projects</th>
<th>Workflows</th>
<th>Executions</th>
<th>Versions</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>100+</td>
<td>30+</td>
<td>1000+</td>
<td>10,000+</td>
<td>100,000+</td>
<td>100+</td>
</tr>
</tbody>
</table>

Statistics of Texera Service as of May 2023
Tutorial Outline

1. Overview of collaborative data analytics

2. Challenges and solutions:
   a. Shared workflow editing
   b. Shared workflow execution
   c. Interacting with runtime execution
   d. Runtime co-debugging on Python UDFs

3. Open challenges
Open Challenges

- Fault tolerance
- Serverless computation for elasticity
- Workflow ecosystem for knowledge sharing in scientific communities
- Reproducibility of workflows
- Dynamic reconfiguration: our research talk “Fries”: 3:30 pm, C8, Junior Ballroom AB
- Debugging Python UDFs: our SIGMOD 2024 “Udon” paper
Acknowledgements

Speakers:
Dr. Chen Li
Dr. Zuozhi Wang

Core Texera Team:
Dr. Sadeem Alsudais
Yunyan Ding
Yicong Huang
Dr. Avinash Kumar
Xinyuan Lin
Xiaozhen Liu
Raj Mohanty
Shengquan Ni

National Science Foundation (NSF) III 1745673, III 2107150.
Building a Collaborative Data Analytics System: Opportunities and Challenges

Zuozhi Wang and Chen Li

UC Irvine

Texera GitHub Repo