Advanced Data Management (CSCI 640/490)

Scalable Dataframes

Dr. David Koop
History of Dataframes

• R, open-source alternative to S, developed in 2000 (with dataframes)
• Pandas, 2009
• Spark, 2010 (resilient distributed dataset [RDD], Dataset API)
Pandas Workflow: Ingest, Cleaning, Analysis

R1. Read HTML
```python
import pandas as pd
products = pd.read_html(...)  # Assume there are multiple tables
products
```

C1. Ordered point updates
```python
products.iloc[2, 0] = "12MP"
products
```

C2. Matrix-like transpose
```python
products = products.T
```

C3. Column transformation
```python
iphone_df = prices.merge(
    one_hot_df,  # Assume one_hot_df is created
    left_index=True, right_index=True
)
```

A1. One-to-many column mapping
```python
one_hot_df = pd.get_dummies(products)
iphone_df = prices.merge(
    one_hot_df,  # Assume one_hot_df is created
    left_index=True, right_index=True
)
```

A2. Joins
```python
iphone_df_cov()  # Assume this is a method to compute covariance
```

A3. Matrix Covariance
```python
iphone_df_cov()  # Assume this is a method to compute covariance
```

[D. Petersohn et al., 2020]
Problems Scaling: From Pandas to Other Solutions

- New Data Source
- New spec
- New requirements

Prototyping
Exploring
Rewrite
Testing
Production

Rewrite
Feedback

Spark
Laptop/Workstation
Small Cluster
Large Cluster

y_i = \beta_0 + \beta_1 x_i + \epsilon_i

D. Koop, CSCI 640/490, Spring 2024
Modin as a Solution

New Data Source
New spec
New requirements

Prototyping
Exploring

Testing

Production

Laptop/Workstation
Small Cluster
Large Cluster

MODIN

Feedback

[D. Petersohn]
Modin Positioning

Ease of use
expressiveness, flexibility, agility

Performance
scalability, robustness, efficiency

pandas
(powered by Modin)

big data frameworks

[D. Petersohn]
Dataframe Data Model

- Combines parts of matrices, databases, and spreadsheets
- Ordered, but not necessarily sorted
  - Rows and columns
- No predefined schema necessary
  - Types can be induced at runtime
- Typed Row/column labels
  - Labels can become data
- Indexing by label or row/column number
  - “Named notation” or “Positional notation”
Comparing Dataframes and Relational Stores

- **Dataframe Characteristics**
  - Ordered table
  - Named rows labels
  - A lazily-induced schema
  - Column names from \( d \in \text{Dom} \)
  - Column/row symmetry
  - Support for linear alg. operators

- **Relational Characteristics**
  - Unordered table
  - No naming of rows
  - Rigid schema
  - Column names from att
  - Columns and rows are distinct
  - No native support

[D. Petersohn et al., 2020]
Comparing Dataframes and Matrices

- **Dataframe Characteristics**
  - Heterogeneously typed
  - Numeric & non-numeric types
  - Explicit row and column labels
  - Support for rel. algebra operators

- **Matrix Characteristics**
  - Homogeneously typed
  - Only numeric types
  - No row or column labels
  - No native support
Dataframe Algebra

<table>
<thead>
<tr>
<th>Operator</th>
<th>(Meta)data</th>
<th>Schema</th>
<th>Origin</th>
<th>Order</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECTION</td>
<td>×</td>
<td>static</td>
<td>REL</td>
<td>Parent</td>
<td>Eliminate rows</td>
</tr>
<tr>
<td>PROJECTION</td>
<td>×</td>
<td>static</td>
<td>REL</td>
<td>Parent</td>
<td>Eliminate columns</td>
</tr>
<tr>
<td>UNION</td>
<td>×</td>
<td>static</td>
<td>REL</td>
<td>Parent</td>
<td>Set union of two dataframes</td>
</tr>
<tr>
<td>DIFFERENCE</td>
<td>×</td>
<td>static</td>
<td>REL</td>
<td>Parent</td>
<td>Set difference of two dataframes</td>
</tr>
<tr>
<td>CROSS PRODUCT / JOIN</td>
<td>×</td>
<td>static</td>
<td>REL</td>
<td>Parent</td>
<td>Combine two dataframes by element</td>
</tr>
<tr>
<td>DROP DUPLICATES</td>
<td>×</td>
<td>static</td>
<td>REL</td>
<td>Parent</td>
<td>Remove duplicate rows</td>
</tr>
<tr>
<td>GROUPBY</td>
<td>×</td>
<td>static</td>
<td>REL</td>
<td>New</td>
<td>Group identical values for a given (set of) attribute(s)</td>
</tr>
<tr>
<td>SORT</td>
<td>×</td>
<td>static</td>
<td>REL</td>
<td>New</td>
<td>Lexicographically order rows</td>
</tr>
<tr>
<td>RENAME</td>
<td>(×)</td>
<td>static</td>
<td>REL</td>
<td>Parent</td>
<td>Change the name of a column</td>
</tr>
<tr>
<td>WINDOW</td>
<td>×</td>
<td>static</td>
<td>SQL</td>
<td>Parent</td>
<td>Apply a function via a sliding-window (either direction)</td>
</tr>
<tr>
<td>TRANSPOSE</td>
<td>(×)</td>
<td>dynamic</td>
<td>DF</td>
<td>Parent</td>
<td>Swap data and metadata between rows and columns</td>
</tr>
<tr>
<td>MAP</td>
<td>(×)</td>
<td>dynamic</td>
<td>DF</td>
<td>Parent</td>
<td>Apply a function uniformly to every row</td>
</tr>
<tr>
<td>TOLABELS</td>
<td>(×)</td>
<td>dynamic</td>
<td>DF</td>
<td>Parent</td>
<td>Set a data column as the row labels column</td>
</tr>
<tr>
<td>FROMLABELS</td>
<td>(×)</td>
<td>dynamic</td>
<td>DF</td>
<td>Parent</td>
<td>Convert the row labels column into a data column</td>
</tr>
</tbody>
</table>

[D. Petersohn et al., 2020]
Pivot Example

Wide Table of MONTHS

<table>
<thead>
<tr>
<th>Month</th>
<th>2001</th>
<th>2002</th>
<th>2003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>100</td>
<td>150</td>
<td>300</td>
</tr>
<tr>
<td>Feb</td>
<td>110</td>
<td>200</td>
<td>310</td>
</tr>
<tr>
<td>Mar</td>
<td>120</td>
<td>250</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Pivot

Wide Table of YEARS

<table>
<thead>
<tr>
<th>Year</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>100</td>
<td>110</td>
<td>120</td>
</tr>
<tr>
<td>2002</td>
<td>150</td>
<td>200</td>
<td>250</td>
</tr>
<tr>
<td>2003</td>
<td>300</td>
<td>310</td>
<td>NULL</td>
</tr>
</tbody>
</table>

[D. Petersohn et al., 2020]
Modin Challenges

• Massive API: 240+ operators, but with a lot of redundancy
• Parallel Execution: row-based, column-based, and block-based
• Data Model Challenges: Schema induction, reusing type info
• Order is important
• Supporting billions of columns: Row/Column equivalence (transpose)
• Metadata is data (and vice versa)
• Users want immediate feedback
• Users want to create queries incrementally
Assignment 4

• Work on Data Integration and Data Fusion
• Integrate artist datasets from different institutions (Met, NGA, AIC, CMA)
  - Integrate information based on ids and matching
• Record Matching:
  - Which artists are the same?
• Data Fusion:
  - Names
  - Dates
  - Nationalities
Test 2

- Upcoming… April 8
- Similar format, but more emphasis on topics we have covered including the research papers
Dataframes, Databases, and the Cloud

• How do we take advantage of different architectures?
• Lots of work in scaling databases and specialized computational engines
• What is the code that people actually write?
Magpie: Python at Speed and Scale using Cloud Backends

A. Jindal

[Data Science Jungle]

[A. Jindal et al., 2021]
## Magpie Goals

<table>
<thead>
<tr>
<th>Pythonic Environment</th>
<th>Familiar Python surface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unified Dataframe API</td>
<td>Ongoing standardization</td>
</tr>
<tr>
<td><strong>Magpie Middleware</strong></td>
<td>Batching Pandas into large query expressions</td>
</tr>
<tr>
<td>PyFroid Compiler</td>
<td>Backend selection using past workloads</td>
</tr>
<tr>
<td>Cross Optimization</td>
<td>Cache commonly seen dataframes</td>
</tr>
<tr>
<td>Common Data Layer</td>
<td>Multi-backend environments and libraries</td>
</tr>
<tr>
<td><strong>Polyengines &amp; Mappers</strong></td>
<td>Cloud backends</td>
</tr>
<tr>
<td><strong>Database Backends</strong></td>
<td></td>
</tr>
</tbody>
</table>
ConnectorX: Databases to Dataframes

- Write read_sql queries but write SQL
- Written in Rust
- Returns a dataframe

```python
query = f""
SELECT l_orderkey,
    SUM(l_extendedprice * ( 1 - l_discount )) AS revenue,
    o_orderdate,
    o_shippriority
FROM customer,
    orders,
    lineitem
WHERE c_mktsegment = 'BUILDING'
    AND c_custkey = o_custkey
    AND l_orderkey = o_orderkey
    AND o_orderdate < DATE '1995-03-15'
    AND l_shipdate > DATE '1995-03-15'
GROUP BY l_orderkey,
    o_orderdate,
    o_shippriority
""

df = read_sql("postgresql://postgres:postgres@localhost:5432/tpch", query,
             partition_on="l_orderkey", partition_num=4)
```
ConnectorX Speed & Memory

[X. Wang, 2022]
Improvements in ConnectorX

- Written in native language (Rust)
- Copy exactly once (even during parallel computations)
- CPU cache-friendly: process in a streaming fashion
An Opinionated Introduction to Polars

Nico Kreiling
Handling Large Data with Polars

Etienne Bacher
Discussion

- Data in the cloud and local exploration
- Languages: SQL or Pandas or Ibis or….?