Magpie
Python at Speed and Scale using Cloud Backends


Gray Systems Lab, Microsoft
Azure Data, Microsoft
Microsoft Research
University of Washington
The Python and The Cloud

- De-facto for ad-hoc analysis
- Pandas dataframes highly popular
- Performance is a challenge

+ Hyper-scale performance
+ Several SQL processing backends
+ Enterprise data already on cloud
The current landscape ... is a fragmented jungle!
Magpie

- **PyFroid Compiler**
- **Cross Optimization**
- **Common Data Layer**

**Magpie Middleware**

- **Unified Dataframe API**
- **Familiar Python surface**
- **Ongoing standardization**
- **Batching Pandas into large query expressions**
- **Backend selection using past workloads**
- **Cache commonly seen dataframes**

**Polyengines & Mappers**
- Azure Synapse Analytics
- Koalas
- SQL Alchemy

**Database Backends**
- Microsoft SCOPE
- Apache Spark
- PostgreSQL
- Apache MADlib
- Google BigQuery
- SQL Server

**Native Python**

**Multi-backend environments and libraries**

**Cloud backends**
The number of taxi trips per weekday over the NYC Taxi dataset

1. import pyfroid.pandas as pd
2. df = pd.read_sql('nyctaxi', con)
3. df = df[df.fare_amount > 0]
4. df['day'] = df.pickup_datetime.dt.dayofweek
5. df = df.groupby(['day'])['passenger_count'].sum()
6. print(df)

Blue parts: already in IBIS, Green parts: our contributions

Intermediate Representation
Pushing Data Science down

Ibis API

Ibis Expression

Pandas

SQL

Spark

Cloud backends

Lazy Translation

Blue parts: already in IBIS, Green parts: our contributions

SELECT DATEPART(WEEKDAY, pickup_datetime) AS day,
SUM(passenger_count)
FROM nyctaxi WHERE fare_amount > 0
GROUP BY DATEPART(WEEKDAY, pickup_datetime)
Pushing Data Science down

Blue parts: already in IBIS, Green parts: our contributions

df = script.extract(path, schema).select("fare_amount > 0") .groupby("day") .project("pickup_datetime.DayOfWeek.ToString() AS day", "passenger_count")
Impact: speed-up using SQL DW

Growing input size

- 1.4x to 20x speedup

Growing query complexity

- 7x to 380x speedup
Impact: scale-out using SCOPE

Scale data science to big data!

SCOPE vs Spark

Spark Wins

SCOPE Wins

Difference between PyScope and PySpark runtime (%)

Input data size

Runtime (sec)

Input data size

1 GB 10 GB 100 GB 1 TB

Query Query2

Query1 Query2

1 10 100 1

-140 -120 -100 -80 -60 -40 -20 0 20 40 60 80

9000 1000 100 10 1
Magpie

- **Familiar Python surface**
- **Ongoing standardization**
- **Batching Pandas into large query expressions**
- **Backend selection using past workloads**
- **Cache commonly seen dataframes**
- **Multi-backend environments and libraries**
- **Cloud backends**

**Magpie Middleware**

- **PyFroid Compiler**
- **Cross Optimization**
- **Common Data Layer**

**Polyengines & Mappers**

- Azure Synapse Analytics
- Koalas
- SQL Alchemy
- PyFroid Compiler
- Native Python

**Database Backends**

- Microsoft SCOPE
- Apache Spark
- PostgreSQL
- Apache MADlib
- Google BigQuery
- SQL Server
- Pandas
- Scalable Dataframe Compiler
Backend Selection

- Leverage past workloads from cloud backends to learn a decision tree
- At compile time:
  - User provides the list of available backends
  - Compile the plan into a common representation
  - Infer best backend using the decision tree

Blue parts: already in IBIS, Green parts: our contributions
Scenario 1: Pandas vs PySpark

- Question:
  - When to switch to a cluster?
  - Or to local execution?

- Decision tree:
  - 84% accuracy on test set
  - On Pandas:
    - 84% median improvement
    - Up to 99% improvement
Scenario 2: PyScope vs PySpark

- Cosmos has both SCOPE and Spark engines now
- Question: which one to use for data science?
- Decision tree
  - 87% accuracy on test set
  - On Spark:
    - Median improvement 85%
    - Up to 98% improvement
Magpie

**Pythonic Environment**

**Unified Dataframe API**

**Magpie Middleware**

- PyFroid Compiler
- Cross Optimization
- Common Data Layer

**Polyengines & Mappers**

- Azure Synapse Analytics
- Koalas
- SQL Alchemy

**Database Backends**

- Microsoft SCOPE
- Apache Spark
- PostgreSQL
- Apache MADlib
- Google BigQuery
- SQL Server

**Native Python**

**Cloud Backends**

- Familiar Python surface
- Ongoing standardization
- Batching Pandas into large query expressions
- Backend selection using past workloads
- Cache commonly seen dataframes
- Multi-backend environments and libraries
- Cloud backends
Common Data Layer

- **Lazy Translation**
- **Cost-based optimization**
- **Backend Selection**
- **Dataframe cache**
  - Generate unique signatures
  - Store repeated dataframes in ArrowFlight server
  - Skip accessing the backend in case of cache hit

**Cloud backends**

**Interactive experience**

**Blue** parts: already in IBIS, **Green** parts: our contributions

**Dataframe cache**

- Speedup
  - 2-3x
  - 4-11x
  - 800-3800x

**Input size (rows)**

- 100K
- 1M
- 10M
Summary

<table>
<thead>
<tr>
<th>Pythonic Environment</th>
<th>Lingua franca for many analyses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unified Dataframe API</td>
<td>Increasingly getting standardized</td>
</tr>
</tbody>
</table>

Magpie Middleware

Pandas Without Regret!
Write once, execute anywhere
Abstracting Data Processing Complexity

<table>
<thead>
<tr>
<th>Polyengines &amp; Mappers</th>
<th>From polystores to polyengines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Backends</td>
<td>Hyperscale performance Data already in the cloud</td>
</tr>
<tr>
<td>Native Python</td>
<td></td>
</tr>
</tbody>
</table>

Gray Systems Lab

https://azuredata.microsoft.com/

Hiring Summer Interns!