Announcements

- What to expect from the next 3 milestones (Milestones 8 - 10)
- How to get feedback on your cross-dataset queries and pipeline designs today. Sign-up sheet: [https://tinyurl.com/y9fdoggk](https://tinyurl.com/y9fdoggk)
1) **How is a ParDo massively parallelized?**

A. **The ParDo’s DoFn** is run on multiple workers and each worker processes a different split of the input elements.

B. **The instructions inside the ParDo’s DoFn** are split up among multiple workers and each worker runs a single instruction over all the input elements.
2) If a `ParDo` is processing a `PCollection` of 100 elements, what is the **maximum** parallelism that could be obtained for this pipeline?

A. 50  
B. 100  
C. 200  
D. None of the above
3) If a PCollection of 100 elements is divided into 10 bundles by the runner and each bundle is run on a different worker, what is the actual parallelism of this pipeline?

A. 50  
B. 100  
C. 200  
D. None of the above
4) In a pipeline that consists of a sequence of \( \text{ParDos} \ 1-\ n \), how can the runner execute the transforms on multiple workers while minimizing the communication costs between the workers?

A. Alter the bundling of elements between each \( \text{ParDo} \) such that an element produced by \( \text{ParDo} 1 \) on worker A gets consumed by \( \text{ParDo} 2 \) on worker B.

B. Maintain the bundling of elements between the \( \text{ParDos} \) such that an element that is produced by \( \text{ParDo} 1 \) on worker A gets consumed by \( \text{ParDo} 2 \) on worker A.

C. Split up the workers into \( n \) groups and run each \( \text{ParDo} \) on a different group of workers.

D. Split up the \( \text{ParDos} \) into their own pipelines as it is not possible to reduce the communication costs when multiple transforms exist in the same pipeline.
5) What happens when a ParDo fails to process an element?

A. The processing of the failed element is restarted on the same worker.
B. The processing of the failed element is restarted on a different worker.
C. The processing of all the bundle is restarted on either the same worker or a different worker.
D. The processing of the entire PCollection is restarted on either the same worker or a different worker.
Case Study

Analysis Questions:

● Are young technology companies as likely to sponsor H1B workers as more established companies?
● How does the compensation of H1B workers compare to the average earnings of domestic workers who are performing the same role and living in the same geographic region?

Datasets:

● H1B applications for years 2015 - 2018 (source: US Dept of Labor)
● Corporate registrations for various states (source: Secretary of States)

Code Repo: [https://github.com/shirleycohen/h1b_analytics](https://github.com/shirleycohen/h1b_analytics)
Objectives

Cross-Dataset Query 1:

- Join H1B’s Employer table with the Sec. of State’s Corp. Registry table on the company’s name and location. Get the age of the company from the incorporation date of the company’s registry record. Group the employers by age (0 - 5 years old, 6 - 10 years old, 11 - 20 years old, etc.) and see how many younger tech companies sponsor H1B workers.

- Technical challenges: 1) matching employers within the H1B dataset due to inconsistent spellings of the company’s name and 2) matching employers across H1B and Corporate Registry datasets due to inconsistent spellings of the company’s name and address.
Objectives

Cross-Dataset Query 2:

- Join H1B’s Job table with the Bureau of Labor Statistics’ Wages and Geography tables on the soc_code and job location. Calculate the annual salary from the hourly wages reported in the Wages table and compare this number to the H1B workers’ pay.

- Technical challenges: joining the job location to the BLS geography area requires looking up the job location’s county and mapping the country name to the corresponding area code in the Geography table.
First Dataset
### Table Details: H1B_Applications_2017

#### 2015 table:
- 241 MB size, 618,804 rows

#### 2016 table:
- 233 MB size, 647,852 rows

#### 2017 table:
- 253 MB size, 624,650 rows

#### 2018 table:
- 283 MB size, 654,162 rows

#### Table Schemas:
- A few schema variations between the tables (column names, data types).
- All schema variations resolved through CTAS statements.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Nullable</th>
</tr>
</thead>
<tbody>
<tr>
<td>case_number</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>visa_class</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>case_status</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employer_name</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employer_business_dba</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employer_address</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employer_city</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employer_state</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employer_postal_code</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employer_country</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employer_province</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employer_phone</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employer_phone_ext</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>naics_code</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>soc_name</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>soc_code</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>job_title</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>total_workers</td>
<td>INTEGER</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>case_submitted</td>
<td>TIMESTAMP</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>decision_date</td>
<td>TIMESTAMP</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employment_start_date</td>
<td>TIMESTAMP</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>employment_end_date</td>
<td>TIMESTAMP</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>full_time_position</td>
<td>BOOLEAN</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>prevailing_wage</td>
<td>FLOAT</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>pw_unit_of_pay</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>wage_rate_of_pay_from</td>
<td>FLOAT</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>wage_rate_of_pay_to</td>
<td>FLOAT</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>wage_unit_of_pay</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>worksite_city</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>worksite_county</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>worksite_state</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>worksite_postal_code</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>agent_attorney_name</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>agent_representing_employer</td>
<td>BOOLEAN</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>agent_attorney_city</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>agent_attorney_state</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>h1b_dependent</td>
<td>BOOLEAN</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>willful_violator</td>
<td>BOOLEAN</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>original_cert_date</td>
<td>TIMESTAMP</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>new_employment</td>
<td>FLOAT</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>continued_employment</td>
<td>FLOAT</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>change_previous_employment</td>
<td>FLOAT</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>new_concurrent_employment</td>
<td>FLOAT</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>change_employer</td>
<td>FLOAT</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>amended_petition</td>
<td>FLOAT</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>pw_wage_level</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>pw_source</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>pw_source_year</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>pw_source_other</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>support_h1b</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>labor_con_agree</td>
<td>BOOLEAN</td>
<td>NULLABLE</td>
</tr>
<tr>
<td>public_disclosure_location</td>
<td>STRING</td>
<td>NULLABLE</td>
</tr>
</tbody>
</table>
-- Create Employer_Temp tables and assign each record a unique employer_id
-- Table contains duplicate employer records, will need to remove duplicates through Beam
CREATE TABLE h1b_split.Employer_Temp AS
SELECT generate_uuid() as employer_id, *
FROM
(SELECT DISTINCT employer_name, employer_address, employer_city, employer_state,
     employer_postal_code, employer_country, employer_province, CAST(employer_phone AS STRING) as employer_phone,
     CAST(CASE WHEN h1b_dependent = 'N' THEN 'False'
             WHEN h1b_dependent = 'Y' THEN 'True'
             ELSE NULL END as BOOL) AS h1b_dependent,
     willful_violator
FROM `cs327e-fa2018.h1b_raw.H1B_Applications_2018`
WHERE employer_name IS NOT NULL AND employer_name != '1' AND employer_city IS NOT NULL
UNION DISTINCT
SELECT DISTINCT employer_name, employer_address, employer_city, employer_state,
     employer_postal_code, employer_country, employer_province, employer_phone, h1b_dependent, willful_violator
FROM `cs327e-fa2018.h1b_raw.H1B_Applications_2017`
WHERE employer_name IS NOT NULL AND employer_name != '1' AND employer_city IS NOT NULL
UNION DISTINCT
SELECT DISTINCT employer_name, employer_address, employer_city, employer_state,
     employer_postal_code, employer_country, employer_province, employer_phone, h1b_dependent, willful_violator
FROM `cs327e-fa2018.h1b_raw.H1B_Applications_2016`
WHERE employer_name IS NOT NULL AND employer_name != '1' AND employer_city IS NOT NULL
UNION DISTINCT
SELECT DISTINCT employer_name, CONCAT(employer_address1, ' ', employer_address2) as employer_address, employer_city, employer_state,
     employer_postal_code, employer_country, employer_province, employer_phone, h1b_dependent, willful_violator
FROM `cs327e-fa2018.h1b_raw.H1B_Applications_2015`
WHERE employer_name IS NOT NULL AND employer_name != '1' AND employer_city IS NOT NULL
ORDER BY employer_name, employer_city;

Source File: https://github.com/shirleycohen/h1b_analytics/blob/master/h1b_ctas.sql
Notes:

Source Tables: h1b_raw.H1B_Applications_<year> where <year> = 2015 - 2018. Each table was loaded from a different CSV file.

Target Tables: Tables showed in this ERD were created in the h1b_split dataset. All tables created and populated from CTAS statements.

Issues with Target Tables:
- Employer_Temp contains duplicate records due to mispellings of the employer name and city.
- Job_Temp and Application_Temp are missing references to Employer table via employer_id.
Beam Transform for Employer Table

- Removes duplicate records from Employer Table
- Version 1 of pipeline uses the Direct Runner for testing and debugging

```python
with beam.Pipeline('DirectRunner', options=opts) as p:
    query_results = p | 'Read from BigQuery' >> beam.io.Read(beam.io.BigQuerySource(query='SELECT * FROM h1b_split.Employer ORDER BY employer_name limit 100'))

    query_results | 'Write to File 1' >> WriteToText('query_results.txt')

    # apply ParDo to the Employer records
tuple_pcoll = query_results | 'Transform Employer Name' >> beam.ParDo(TransformEmployerName())

    # write PCollection to log file
tuple_pcoll | 'Write to File 2' >> WriteToText('output_pardo_employer_tuple.txt')

    deduped_pcoll = tuple_pcoll | 'Dedup Employer Records' >> beam.GroupByKey()

    # write PCollection to log file
deduped_pcoll | 'Write to File 3' >> WriteToText('output_group_by_key.txt')

    # apply second ParDo to the PCollection
out_pcoll = deduped_pcoll | 'Create Employer Record' >> beam.ParDo(MakeRecord())
```

Source File: https://github.com/shirleycohen/h1b_analytics/blob/master/transform_employer_table_single.py
Beam Transform for Employer Table

- Removes duplicate records from Employer Table
- Version 2 of pipeline uses the Dataflow Runner for parallel processing

```python
# run pipeline on Dataflow
options = {
    'runner': 'DataflowRunner',
    'job_name': 'dedup-employer-table',
    'project': PROJECT_ID,
    'temp_location': BUCKET + '/temp',
    'staging_location': BUCKET + '/staging',
    'machine_type': 'n1-standard-8',
    'num_workers': 0
}
opts = beam.pipeline.PipelineOptions(flags=[], **options)

with beam.Pipeline(options=opts) as p:

    query_results = p | 'Read from BigQuery' >> beam.io.Read(beam.io.BigQuerySource(query='SELECT * FROM h1b_split.Employer_Temp ORDER BY employer_name'))

    # write PCollection to log file
    query_results | 'Write to File 1' >> WriteToText(DIR_PATH + 'query_results.txt')

    # apply ParDo to the Employer records
    tuple_pcoll = query_results | 'Transform Employer Name' >> beam.ParDo(TransformEmployerName())

    # write PCollection to a log file
    tuple_pcoll | 'Write to File 2' >> WriteToText(DIR_PATH + 'output_parbo_employer_tuple.txt')

    deduped_pcoll = tuple_pcoll | 'Dedup Employer Records' >> beam.GroupByKey()

Source File: https://github.com/shirleycohen/h1b_analytics/blob/master/transform_employer_table_cluster.py
Beam Transforms for Job and Application Tables

- Clean the employer name and city and find the matching employer_id from Employer table to use as reference in the Job and Application tables

- Pipeline Sketch for Job Table:
  1. Read in all the records from the Employer and Job tables in BigQuery and create a PCollection from each source
  2. Clean up the employer’s name and city from the Job PCollection (using ParDo)
  3. Join the Job and Employer PCollections on employer’s name and city (using CoGroupByKey).
  4. Extract the matching employer_id from the results of the join and add it to the Job element (using ParDo)
  5. Remove employer’s name and city from the Job element (using ParDo)
  6. Write out new Job table to BigQuery

- Repeat procedure for Application table
H1B Analytics ERD Version 2

Application
- PK: case_number
- case_status: String
- case_submitted: Date
- decision_date: Date
- visa_class: String
- job_id: String
- employer_id: String
- attorney_id: String

Employer
- PK: employer_id
- employer_name: String
- employer_address: String
- employer_city: String
- employer_state: String
- employer_postal_code: String
- employer_country: String
- employer_province: String
- employer_phone: String
- h1b_dependent: Boolean
- willful_violator: Boolean
- job_id: String
- employment_start_date: Date
- employment_end_date: Date
- job_title: String
- wage_rate_of_pay_from: Float
- wage_rate_of_pay_to: Float
- wage_unit_of_pay: String
- worksite_city: String
- worksite_county: String
- worksite_state: String
- worksite_postal_code: String
- soc_code: String
- soc_name: String
- total_workers: Integer
- full_time_position: Boolean
- prevailing_wage: Float
- pw_unit_of_pay: String
- pw_wage_level: String
- pw_source: String
- pw_source_year: Integer
- pw_source_other: String

Notes:
Source Tables: h1b_split.Employer_Temp, h1b_split.Application_Temp, h1b_split.Job_Temp
Target Tables: h1b_split.Employer, h1b_split.Application, h1b_split.Job. All new tables created and populated from Beam pipelines.
Changes since previous version:
- Removed 187,117 duplicate records from Employer table based on uniqueness criteria of (employer name, city) pairs.
- Added reference to employer_id from Job and Application tables.

Number of Rows
<table>
<thead>
<tr>
<th></th>
<th>v1</th>
<th>v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employer</td>
<td>348,876</td>
<td>161,759</td>
</tr>
<tr>
<td>Job</td>
<td>2,230,779</td>
<td>2,230,626</td>
</tr>
<tr>
<td>Application</td>
<td>2,633,426</td>
<td>2,633,156</td>
</tr>
<tr>
<td>Attorney</td>
<td>19,861</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Milestone 8