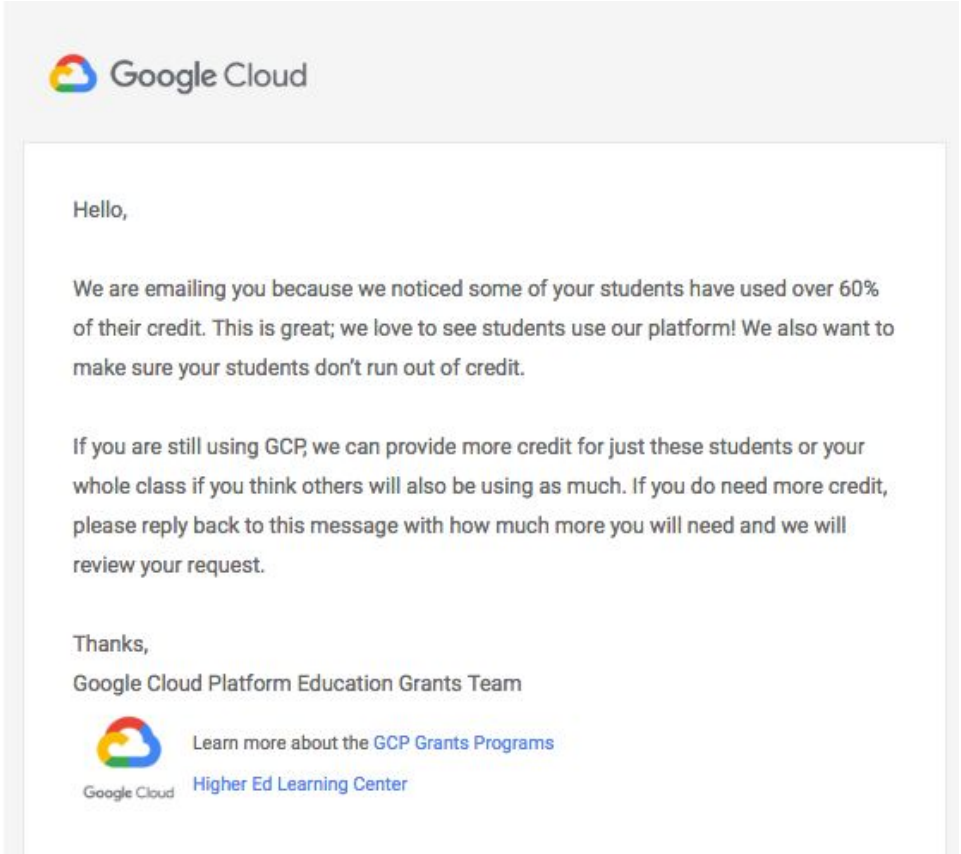
CS 327E Class 7

November 5, 2018

Check your GCP Credits :)



Google Cloud

Hello,

We are emailing you because we noticed some of your students have used over 60% of their credit. This is great; we love to see students use our platform! We also want to make sure your students don't run out of credit.

If you are still using GCP, we can provide more credit for just these students or your whole class if you think others will also be using as much. If you do need more credit, please reply back to this message with how much more you will need and we will review your request.

Thanks,
Google Cloud Platform Education Grants Team

Google Cloud

Learn more about the GCP Grants Programs

Higher Ed Learning Center

iClicker Question

Are you running low on GCP credits?

A. Yes
B. No

No Quiz Today

Dataflow Concepts

- A system for processing arbitrary computations on large amounts of data
- Can process batch data and streaming data using the same code
- Uses Apache Beam, an open-source programming model
- Designed to be very scalable, millions of QPS

Apache Beam Concepts

- A model for describing data and data processing operations:
  - `Pipeline`: a data processing task from start to finish
  - `PCollection`: a collection of data elements
  - `Transform`: a data transformation operation
- SDKs for Java, Python and Go
- Executed in the cloud on Dataflow, Spark, Flink, etc.
- Executed locally with Direct Runner for dev/testing

Beam Pipeline

- `Pipeline` = A directed acyclic graph where the nodes are the `Transforms` and the edges are the `PCollections`
- General Structure of a `Pipeline`:
  - Reads one or more data sources as input `PCollections`
  - Applies one or more `Transforms` on the `PCollections`
  - Outputs resulting `PCollection` as one or more data sinks
- Executed as a single unit
- Run in batch or streaming mode

PCollection

- `PCollection` = A collection of data elements
- Elements can be of any type (String, Int, Array, etc.)
- `PCollections` are distributed across machines
- `PCollections` are immutable
- Created from a data source or a `Transform`
- Written to a data sink or passed to another `Transform`

Transform Types

- Element-wise:
  - maps 1 input to (1, 0, many) outputs
  - Examples: `ParDo`, `Map`, `FlatMap`
- Aggregation:
  - reduces many inputs to (1, fewer) outputs
  - Examples: `GroupByKey`, `CoGroupByKey`
- Composite: combines element-wise and aggregation
  - `GroupByKey -> ParDo`

Transform Properties

- Serializable
- Parallelizable
- Idempotent

ParDo

- ParDo = “Parallel Do”
- Maps 1 input to (1, 0, many) outputs
- Takes as input a PCollection
- Applies the user-defined ParDo to the input PCollection
- Outputs results as a new PCollection
- Typical usage: filtering, formatting, extracting parts of data, performing computations on data elements

ParDo Example

```python
import apache_beam as beam
from apache_beam.io import ReadFromText
from apache_beam.io import WriteToText

# DoFn to perform on each element in the input PCollection.
class ComputeWordLengthFn(beam.DoFn):
  def process(self, element):
    words = element.strip().split(' ')
    result_list = []
    for word in words:
        result_list.append((word, len(word)))
    return result_list

# Create a Pipeline using a local runner for execution.
with beam.Pipeline('DirectRunner') as p:

    # create a PCollection from the file contents.
    in_pcoll = p | 'Read' >> ReadFromText('input.txt')

    # apply a ParDo to the PCollection
    out_pcoll = in_pcoll | beam.ParDo(ComputeWordLengthFn())

    # write PCollection to a file
    out_pcoll | 'Write' >> WriteToText('output.txt')
```

Source File: https://github.com/cs327e-fall2018/snippets/blob/master/word_length.py

Aggregation Example

```python
import apache_beam as beam
from apache_beam.io import ReadFromText
from apache_beam.io import WriteToText

# DoFn to perform on each element in the input PCollection.
class ComputeWordLengthFn(beam.DoFn):
  def process(self, element):
    words = element.strip().split(' ')
    result_list = []
    for word in words:
        result_list.append((len(word), word))
    return result_list

# Create a Pipeline using a local runner for execution.
with beam.Pipeline('DirectRunner') as p:

    # create a PCollection from the file contents.
    in_pcoll = p | 'Read' >> ReadFromText('input.txt')

    # apply a ParDo to the PCollection
    word_pcoll = in_pcoll | 'ParDo' >> beam.ParDo(ComputeWordLengthFn())

    # apply GroupByKey to the PCollection
    out_pcoll = word_pcoll | 'GroupByKey' >> beam.GroupByKey()

    # write PCollection to a file
    out_pcoll | 'Write' >> WriteToText('output.txt')
```

Source File: https://github.com/cs327e-fall2018/snippets/blob/master/group_words_by_length.py

BigQuery Data Sink Example

```python
import logging
import apache_beam as beam
from apache_beam.io import ReadFromText
from apache_beam.io import WriteToText

# DoFn to perform on each element in the input PCollection.
class ComputeWordLengthFn(beam.DoFn):
  def process(self, element):
    words = element.strip().split(' ')
    result_list = []
    for word in words:
        result = {'word' : word, 'length' : len(word)}
        result_list.append(result)
    return result_list

# Create a Pipeline using a local runner for execution.
with beam.Pipeline('DirectRunner') as p:

    # create a PCollection from the file contents.
    in_pcoll = p | 'Read from File' >> ReadFromText('input.txt')

    # apply a ParDo to the PCollection
    out_pcoll = in_pcoll | beam.ParDo(ComputeWordLengthFn())

    # write PCollection to a file
    out_pcoll | 'Write to File' >> WriteToText('output.txt')

    # write PCollection to a BQ table
    qualified_table_name = 'cs327e-fa2018:beam.Words'
    table_schema = 'word:STRING,length:INTEGER'
    out_pcoll | 'Write to BigQuery' >> beam.io.Write(beam.io.BigQuerySink(qualified_table_name,
                                                schema=table_schema,
                                                create_disposition=beam.io.BigQueryDisposition.CREATE_IF_NEEDED,
                                                write_disposition=beam.io.BigQueryDisposition.WRITE_TRUNCATE))
```

Source File: https://github.com/cs327e-fall2018/snippets/blob/master/word_length_bq_out.py

How to “Dataflow”

1. Start with some initial working code.
2. Test and debug **each** new line of code.
3. Write temporary and final `PCollections` to log files.
4. Test and debug **end-to-end** pipeline locally before running on Dataflow.
5. If you get stuck, make a Piazza post that has **enough** details for the instructors to reproduce the error and help you troubleshoot.
6. Start assignments **early**. The Beam Python documentation is sparse and learning Beam requires patience and experimentation.

Dataflow Set Up

https://github.com/cs327e-fall2018/snippets/wiki/Dataflow-Setup-Guide

Milestone 7

http://www.cs.utexas.edu/~scohen/milestones/Milestone7.pdf