Announcements

● Feedback on Test 3
● Extra credit opportunities
● Milestones 3 and 4
Motivations for Dataflow

- 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
  - **PTransform**: a data transformation operation
- Supported languages: 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 PTransforms and the edges are PCollections

● General Structure of a Pipeline:
  ○ Reads one or more data sources as input PCollections
  ○ Applies one or more PTransforms on PCollections
  ○ Outputs resulting PCollection as one or more data sinks

● Executed as a single unit

● Runs in batch or streaming mode
**PCollection**

- A collection of data elements, either bounded or unbounded
- Elements can be made up primitive and complex types
- Distributed across machines
- PCollections are immutable
- Created from a data source or a PTransform
- Written to a data sink or passed to another PTransform
PTransforms

All operations on data are different kinds of PTransforms

- **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, Flatten

- **Composite:** combines element-wise and aggregation
  - GroupByKey -> ParDo
PTransform Properties

● Serializable
● Parallelizable
● Idempotent
ParDo Transform

- ParDo = "Parallel Do"
- Maps 1 input to (0, 1, many) outputs
- Takes as input a PCollection
- Applies the user-defined ParDo to the input
- Outputs results as new PCollection
- Typical usage: filtering, formatting, extracting parts of data, performing computations on data elements
class Multiply(beam.DoFn):
    def process(self, element):
        return [element * 10]

p = beam.Pipeline('DirectRunner', options=opts)

in_pcoll = p | beam.Create([1, 2, 3, 4, 5])

out_pcoll = in_pcoll | 'Multiply' >> beam.ParDo(Multiply())

out_pcoll | 'Write results' >> WriteToText('multiplied.txt')
GroupByKey Transform

- Input: PCollection where each element is a (key, value) pair
- Groups the values by unique key
- Output: PCollection where each element is a (key, list(value)) pair

(input)

('Nicole', '100 Avenue A')
('Erik', '21 Guadalupe')
('Sameer', '7071 Hamilton')
('Nicole', '200 Avenue B')

(output)

GroupByKey

('Nicole', ['100 Avenue A', '200 Avenue B'])
('Erik', '21 Guadalupe')
('Sameer', '7071 Hamilton')
Hello World Example 2

class SplitWords(beam.DoFn):
    def process(self, element):
        results = []
        words = element.split()

        for word in words:
            results.append((word, 1))

        return results

p = beam.Pipeline('DirectRunner', options=opts)

in_pcoll = p | beam.Create(['here are some words', 'here a few more words'])

split_pcoll = in_pcoll | 'Split Words' >> beam.ParDo(SplitWords())

out_pcoll = split_pcoll | 'Group Words' >> beam.GroupByKey()
Beam + Dataflow Setup

Hands-on Exercises

git clone https://github.com/cs327e-fall2020/snippets.git
How to develop Beam pipelines:

1. Start with a working code example and incrementally add to it.
2. Test and debug one transform at a time.
3. Write temporary and final PCollections to log files.
4. You may encounter jupyter notebook issues.
5. Start on the assignment **as early as possible**. The Beam Python documentation is sparse and learning Beam requires *patience*, *perseverance*, and *experimentation*.
6. Piazza won’t be a good way to debug.
7. If you get stuck, go to OHs. If you can’t make OHs, make an appointment with the TAs.
Milestone 3