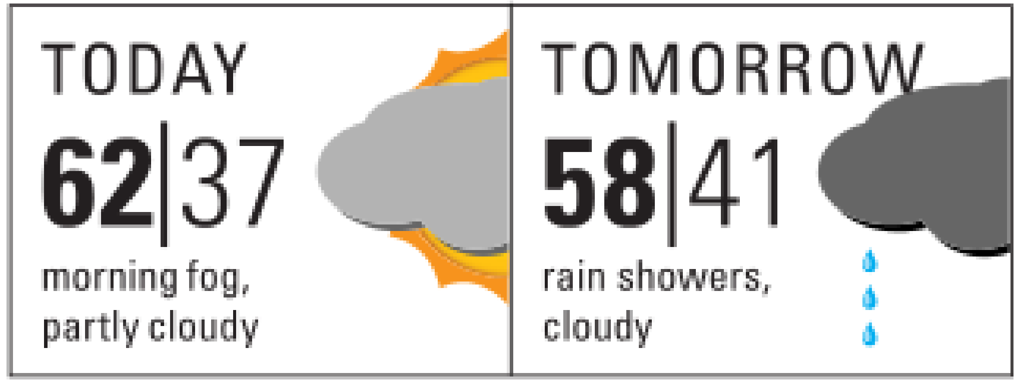
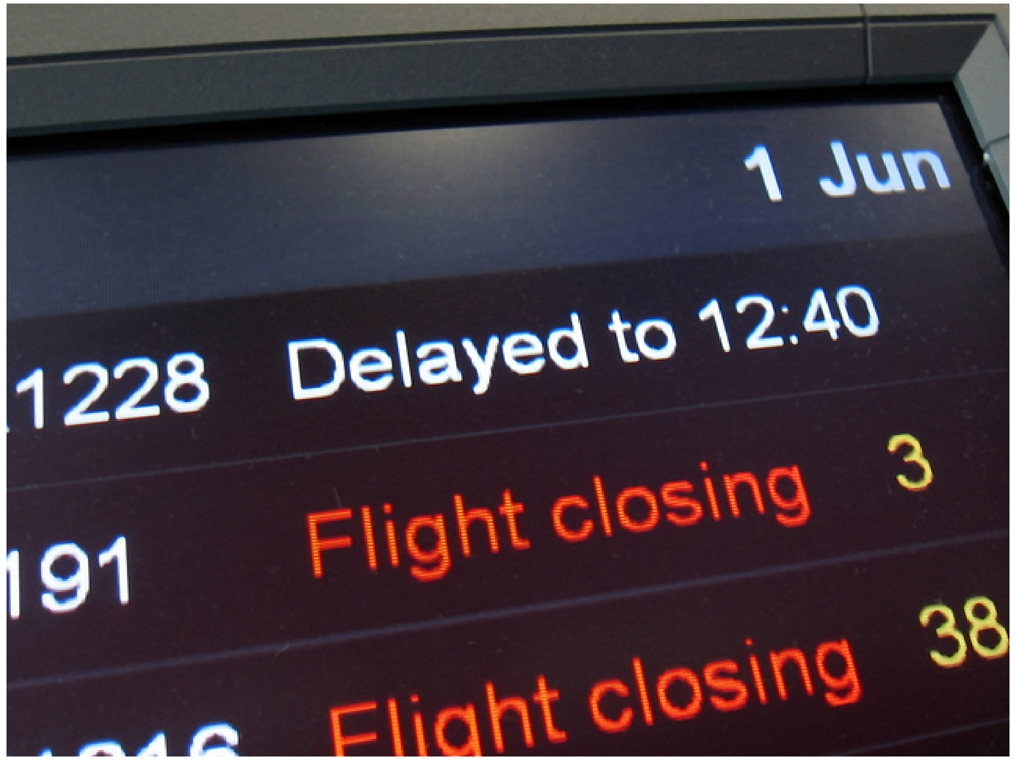
Preparing Flight Delay Data

PARALLEL COMPUTING WITH DASK

Dhavide Aruliah
Director of Training, Anaconda

Case study: Analyzing flight delays





Limitations of Dask DataFrames

- Reading data into Dask DataFrames:
  - A single file
  - Using `glob` on many files
- Limitations:
  - Unsupported file formats
  - Cleaning files independently
  - Nested subdirectories tricky with `glob`

Sample account data

`accounts/Alice.csv` :

```
date,amount
2016-01-31,103.15
2016-02-25,114.17
2016-03-06,4.03
2016-05-20,150.48
```

`accounts/Bob.csv` :

```
date,amount
2016-01-04,99.68
2016-02-09,146.41
2016-02-21,-42.94
2016-03-14,0.26
```

Reading/cleaning in a function

```
import pandas as pd
from dask import delayed

@delayed
    def pipeline(filename, account_name):

        df = pd.read_csv(filename)

        df['account_name'] = account_name
        return df
```

Using dd.from_delayed()

```
delayed_dfs = []
for account in ['Bob', 'Alice', 'Dave']:
    fname = 'accounts/{}.csv'.format(account)
    delayed_dfs.append(pipeline(fname, account))
import dask.dataframe as dd
dask_df = dd.from_delayed(delayed_dfs)
dask_df['amount'].mean().compute()
```

```
10.56476
```

Flight delays and weather

- Cleaning flight delays
  - Use `.replace()` : `0` $\rightarrow$ `NaN`
- Cleaning weather data
  - `'PrecipitationIn'` : text $\rightarrow$ numeric
  - Add column for airport code

Flight delays data

```
df = pd.read_csv('flightdelays-2016-1.csv')

df.columns
```

```
Index(['FL_DATE', 'UNIQUE_CARRIER', 'FL_NUM', 'ORIGIN',
       'ORIGIN_CITY_NAME', 'ORIGIN_STATE_ABR', 'ORIGIN_STATE_NM',
       'DEST', 'DEST_CITY_NAME', 'DEST_STATE_ABR',
       'DEST_STATE_NM', 'CRS_DEP_TIME', 'DEP_DELAY',
       'CRS_ARR_TIME', 'ARR_DELAY', 'CANCELLED', 'DIVERTED',
       'CARRIER_DELAY','WEATHER_DELAY', 'NAS_DELAY',
       'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY',
       'Unnamed: 22'],
      dtype='object')
```

Flight delays data

```
df['WEATHER_DELAY'].tail()
```

```
89160    NaN
89161    0.0
89162    NaN
89163    NaN
89164    NaN
Name: WEATHER_DELAY, dtype: float64
```

Replacing values

```
series
```

```
0    6
1    0
2    6
3    5
4    7
dtype: int64
```

```
new_series = series.replace(
                        6, np.nan)
new_series
```

```
0    NaN
1    0.0
2    NaN
3    5.0
4    7.0
dtype: float64
```

Let's practice!

PARALLEL COMPUTING WITH DASK

Preparing Weather Data

PARALLEL COMPUTING WITH DASK

Dhavide Aruliah
Director of Training, Anaconda

Daily weather data

```python
import pandas as pd
df = pd.read_csv('DEN.csv', parse_dates=True, index_col='Date')
df.columns
```

```
Index(['Max TemperatureF', 'Mean TemperatureF', 'Min TemperatureF',
       'Max Dew PointF', 'MeanDew PointF', 'Min DewpointF', 'Max Humidity',
       'Mean Humidity', 'Min Humidity', 'Max Sea Level PressureIn',
       'Mean Sea Level PressureIn', 'Min Sea Level PressureIn',
       'Max VisibilityMiles', 'Mean VisibilityMiles',
       'Min VisibilityMiles',
       'Max Wind SpeedMPH', 'Mean Wind SpeedMPH', 'Max Gust SpeedMPH',
       'PrecipitationIn', 'CloudCover', 'Events', 'WindDirDegrees'],
      dtype='object')
```

Daily weather data

```
df.loc['March 2016', ['PrecipitationIn','Events']].tail()
```

```
           PrecipitationIn              Events
Date
2016-03-27            0.00                 NaN
2016-03-28            0.00                 NaN
2016-03-29            0.04  Rain-Thunderstorm
2016-03-30            0.04           Rain-Snow
2016-03-31            0.01                Snow
```

Examining PrecipitationIn & Events columns

```
df['PrecipitationIn'][0]
type(df['PrecipitationIn'][0])
```

```
'0.00'
str
```

```
df[['PrecipitationIn', 'Events']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 366 entries, 0 to 365
Data columns (total 2 columns):
PrecipitationIn    366 non-null object
Events             115 non-null object
dtypes: object(2)
memory usage: 5.8+ KB
```

Converting to numeric values

```
series
```

```
0      0
1      M
2      2
3    1.5
4      E
dtype: object
```

```
new_series = pd.to_numeric(serie
                   errors='coerce
new_series
```

```
0    0.0
1    NaN
2    2.0
3    1.5
4    NaN
dtype: float64
```

Let's practice!

PARALLEL COMPUTING WITH DASK

Merging & Persisting DataFrames

PARALLEL COMPUTING WITH DASK

Dhavide Aruliah
Director of Training, Anaconda

Merging DataFrames

- Pandas: `pd.merge()`
- Pandas: `pd.DataFrame.merge()`
- Dask: `dask.dataframe.merge()`

Merging example

```
left_df
```

```
  cat_left  value_left
0        d           4
1        d           9
2        b           1
3        d           7
4        c           3
```

```
right_df
```

```
  cat_right  value_right
0         b            9
1         c            2
2         f            0
3         d            8
4         a            8
```

Merging example

```
left_df.merge(right_df,
              left_on=['cat_left'],
              right_on=['cat_right'],
              how='inner')
```

```
  cat_left  value_left cat_right  value_right
0        d           4         d            8
1        d           9         d            8
2        d           7         d            8
3        b           1         b            9
4        c           3         c            2
```

Dask DataFrame pipelines

- Flight delays & weather set up
  1. Read & clean 12 months of flight delay data
  2. Make `flight_delay` dataframe with `dd.from_delayed`
  3. Read & clean weather daily data from 5 airports
  4. Make `weather` dataframe with `dd.from_delayed`
  5. Merge the two dataframes

Dask DataFrame pipelines

- Flight delays & weather set up
  1. **Read & clean 12 months of flight delay data**
  2. Make `flight_delay` dataframe with `dd.from_delayed`
  3. **Read & clean weather daily data from 5 airports**
  4. Make `weather` dataframe with `dd.from_delayed`
  5. Merge the two dataframes

Repeated reads & performance

```
import dask.dataframe as dd
df = dd.read_csv('flightdelays-2016-*.csv')
%time print(df.WEATHER_DELAY.mean().compute())
```

```
2.701183508773752
CPU times: user 3.35 s, sys: 719 ms, total: 4.07 s
Wall time: 1.64 s
```

```
%time print(df.WEATHER_DELAY.std().compute())
```

```
21.230502105
CPU times: user 3.33 s, sys: 706 ms, total: 4.04 s
Wall time: 1.61 s
```

Repeated reads & performance

```
%time print(df.WEATHER_DELAY.count().compute())
```

```
192563
CPU times: user 3.36 s, sys: 695 ms, total: 4.06 s
Wall time: 1.66 s
```

Using persistence

```
%time persisted_df = df.persist()
```

```
CPU times: user 3.32 s, sys: 688 ms, total: 4.01 s
Wall time: 1.59 s
```

```
%time print(persisted_df.WEATHER_DELAY.mean().compute())
```

```
2.701183508773752
CPU times: user 15.1 ms, sys: 9.24 ms, total: 24.3 ms
Wall time: 18.5 ms
```

Using persistence

```
%time print(persisted_df.WEATHER_DELAY.std().compute())
```

```
21.230502105
CPU times: user 29.6 ms, sys: 12.5 ms, total: 42.1 ms
Wall time: 29.5 ms
```

```
%time print(persisted_df.WEATHER_DELAY.count().compute())
```

```
192563
CPU times: user 9.88 ms, sys: 2.98 ms, total: 12.9 ms
Wall time: 9.43 ms
```

Let's practice!

Final thoughts

PARALLEL COMPUTING WITH DASK

Matthew Rocklin & Dhavide Aruliah
Instructors, Anaconda

What you've learned

- How to:
  - Use Dask data structures and delayed functions
  - Set up data analysis pipelines with deferred computation
  - ... while working with real-world data!

Next steps

- Deploying Dask on your own cluster
- Integrating with other Python libraries
- Dynamic task scheduling and data management
- **https://dask.org/**

Congratulations!

PARALLEL COMPUTING WITH DASK