DSC 102: Systems for Scalable Analytics

PA0: Setting up AWS and Dask

Presenter: Aniruddha Das
Slide Contributors: Vraj Shah, Arun Kumar
Miscellaneous

• Office hours
  • Regular OHs: Every Thursday 1:00pm - 2:00pm PT
  • Extra in PA0 interval: 4:30pm-6:30pm PT on Oct 12, Oct 14 and Oct 17, and 12:30pm-2:30pm PT on Oct 18

• AWS link
  • https://ets-apps.ucsd.edu/individual/DSC102_FA22_A00/

• AWS credentials
  • https://ets-apps.ucsd.edu/individual/DSC102_FA22_A00?mode=env
Goals

1. Get comfortable with AWS
2. Read data from an S3 bucket
3. Run dask on data files that are too big to fit in a single-node machine
AWS Services

Elastic Compute Cloud (EC2)

Elastic Block Storage (EBS)

Simple Storage Service (S3)

AWS-internal Interconnect

Internet

You

AWS Data Center(s)
Launching an EC2 instance

1. Sign in to the AWS Management Console
2. Select the correct AMI
3. Configure your instance (memory, CPUs, storage etc.)
4. Configure security (who can access)
5. Authentication (to connect to the instance later)
6. Launch your instance
Connecting to an EC2 instance (SSH)

1. Confirm security group is configured properly
2. Get the IP address of your instance
3. Go to the folder containing the private key file in your terminal
4. Use the `ssh` command to connect to your instance.
Dask: Overview

Parallel computing framework that scales existing Python ecosystems

Breaks up work into tasks and executes them in task parallel manner

Dask provides APIs to create task graphs (DAG)

Dask also provides a scheduler that runs this DAG by assigning tasks to workers
Dask: APIs

Low-level APIs:

• Dask Delayed (Parallel lazy objects)
• Dask Futures (Parallel eager objects)

High-level APIs:

• Dask Array (Parallel NumPy)
• Dask DataFrame (Parallel Pandas)
• Dask Bag (Parallel Dictionary)
• Dask ML (Parallel Scikit-Learn)

Dataframe APIs enough for this assignment, feel free to check out other APIs if needed
Dask: DataFrame API

Consists of multiple smaller Pandas DFs

Smaller DFs can be present on disk of a single or multiple machines

Operation on Dask DF triggers operations on smaller Pandas DFs

Similar APIs as Pandas DF APIs

```python
import pandas as pd
df = pd.read_csv('filename.csv')
```

```python
import dask.dataframe as dd
df = dd.read_csv('filename.csv')
```

Use `.compute()` method to trigger computation

```
df.column.mean().compute()
```
## Assignment: Dataset Description

<table>
<thead>
<tr>
<th>Column name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>reviewerID</td>
<td>A32DT10X9WS4D0</td>
</tr>
<tr>
<td>asin</td>
<td>B003VX9DJM</td>
</tr>
<tr>
<td>reviewerName</td>
<td>Slade</td>
</tr>
<tr>
<td>helpful</td>
<td>[4, 5]</td>
</tr>
<tr>
<td>reviewText</td>
<td>this was a gift for my friend who loves touch lamps.</td>
</tr>
<tr>
<td>overall</td>
<td>1</td>
</tr>
<tr>
<td>summary</td>
<td>broken piece</td>
</tr>
<tr>
<td>time</td>
<td>1397174400</td>
</tr>
<tr>
<td>reviewTime</td>
<td>04 11, 2014</td>
</tr>
</tbody>
</table>

*Reviews* table
(1) Create a new *users* dataframe using only the *reviews* table with the following schema:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column example</th>
</tr>
</thead>
<tbody>
<tr>
<td>reviewerID (PRIMARY KEY)</td>
<td>A32DT10X9WS4D0</td>
</tr>
<tr>
<td>number_products_rated</td>
<td>20</td>
</tr>
<tr>
<td>avg_ratings</td>
<td>3.5</td>
</tr>
<tr>
<td>reviewing_since</td>
<td>2011</td>
</tr>
<tr>
<td>helpful_votes</td>
<td>15</td>
</tr>
<tr>
<td>total_votes</td>
<td>34</td>
</tr>
</tbody>
</table>
Grading Scheme

Accuracy (80)

- 5 columns
- 16 points per column
- Error tolerance 1%

Runtime (20)

<table>
<thead>
<tr>
<th>Absolute single node runtimes</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Under 20 mins</td>
<td>20</td>
</tr>
<tr>
<td>Between 20 mins and 30 mins</td>
<td>12</td>
</tr>
<tr>
<td>Between 30 mins and 1 hr</td>
<td>8</td>
</tr>
<tr>
<td>Anything above 1 hr</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Grading Scheme

We tolerate 1% error margin, but solutions should match exactly on element-wise basis to ours
Best Practices for PA0

• Use private GitHub repo if possible for handling code and logs.

• Terminate the AWS instance every time after usage; launch again & read from S3 again next time to save budget.

• Since the development set is large, work on a much smaller subset first. Move to the full dataset later.

• Some helpful dask APIs
  • `groupby`
  • `map_partitions`
  • `apply`

• While performing groupby aggregations on large #groups (millions or more), use `split_out` to split output into multiple partitions to avoid worker error. (see `this`)
Other Helpful Links

Any Questions?
Demo