Amazon Web Services (AWS) Batch Array

Fan Wang

2020-10-17

Contents

1 AWS Batch Run **1**
1.1 Preparing a Docker Image and a Python Function for Batch Array Job 1
1.2 Register A Batch Job Definition 1
1.3 Submit a Batch Array 3
1.4 Track the Status of a Submitted Batch Array Until it Finished 4

1 AWS Batch Run

Go to the **RMD**, **PDF**, or **HTML** version of this file. Go back to Python Code Examples Repository (bookdown site) or the pyfan Package (API).

1.1 Preparing a Docker Image and a Python Function for Batch Array Job

We want to set-up a function that can be used jointly with AWS Batch Array. With Batch Array, can run many simulations concurrently. All simulations might only differ in random seed for drawing shocks. This requires setting up the proper dockerfile as well as modifying the python function that we want to invoke slightly.

First, create and push a docker image, see this dockerfile. Following the AWS ECR instructions, this registers a docker image in AWS ECR with a URI: *XXXX7367XXXX.dkr.ecr.us-east-1.amazonaws.com/fanconda*

The dockerfile has for CMD: *CMD ["python", "/pyfan/pyfan/graph/exa/scatterline3.py"]*. This runs the function scatterline3.

Second, the scatterline3 function checks if *AWS_BATCH_JOB_ARRAY_INDEX* is in the *os.environ*. *AWS_BATCH_JOB_ARRAY_INDEX*, if exists, is used as a random seed to generate data for the graph. When the function is run in a docker container via batch, the function saves the graph output to a bucket in AWS s3. The pushing the s3 is achieved by pyfan.aws.general.path.py.

In the batch job, when *arrayProperties = {'size': 10}*, this will generate *AWS_BATCH_JOB_ARRAY_INDEX* from 1 through 10 in 10 sub-tasks of a single batch task. These *AWS_BATCH_JOB_ARRAY_INDEX* could be used as different random seeds, and could be used as folder suffixes.

Here, the scatterline3 function generates a graph, that will be stored for testing purpose in pyfan_gph_scatter_line_rand folder of fans3testbucket bucket, the images saved has *seed_0.png*, *seed_1.png*, ..., *seed_10.png* as names when *arrayProperties = {'size': 10}*.

1.2 Register A Batch Job Definition

Given the docker image we created: *XXXX7367XXXX.dkr.ecr.us-east-1.amazonaws.com/fanconda*, we can use this to register a batch job.

1. computing requirements: memory and cpu: *vCpus = 1* and *Memory=7168* for example
2. which container to pull from (ECR): List the image name: *XXXX7367XXXX.dkr.ecr.us-east-1.amazonaws.com/fanconda* for example
3. job role ARN: *arn:aws:iam::XXXX7367XXXX:role/ecsExecutionRole* to allow for proper in and out from and to the container.

These can be registered programmatically by using boto3: Boto3 Batch Documentation

In the example below, will register a new job definition, this will add *pyfan-scatterline3-test-rmd* to job definition as an additional job definition.

Everytime, when the code below is re-run, a new batch revision number is generated. AWS allows per batch job to have potential hundreds of thousands of revisions.

```
import boto3
import yaml
import pprint

# Load YAML file with security info
srn_aws_yml = "C:/Users/fan/fanwangecon.github.io/_data/aws.yml"
fl_yaml = open(srn_aws_yml)
ls_dict_yml = yaml.load(fl_yaml, Loader=yaml.BaseLoader)
aws_yml_dict_yml = ls_dict_yml[0]

# Dictionary storing job definition related information
job_dict = {"jobDefinitionName": 'pyfan-scatterline3-test-rmd',
            "type": "container",
            "containerProperties": {
                "image": aws_yml_dict_yml['main_aws_id'] + ".dkr.ecr." +
                         aws_yml_dict_yml['region'] + ".amazonaws.com/fanconda",
                "vcpus": int(1),
                "memory": int(1024),
                "command": ["python",
                            "/pyfan/pyfan/graph/exa/scatterline3.py",
                            "-A", "fans3testbucket",
                            "-B", "111"],
                "jobRoleArn": "arn:aws:iam::" + aws_yml_dict_yml['main_aws_id'] +
                              ":role/" + aws_yml_dict_yml['batch_task_executionRoleArn']
            },
            "retryStrategy": {
                "attempts": 1
            }}

# Use AWS Personal Access Keys etc to start boto3 client
aws_batch = boto3.client('batch',
  aws_access_key_id=aws_yml_dict_yml['aws_access_key_id'],
  aws_secret_access_key=aws_yml_dict_yml['aws_secret_access_key'],
  region_name=aws_yml_dict_yml['region'])

# Register a job definition
response = aws_batch.register_job_definition(
        jobDefinitionName = job_dict['jobDefinitionName'],
        type = job_dict['type'],
        containerProperties = job_dict['containerProperties'],
        retryStrategy = job_dict['retryStrategy'])
```

```
# Print response
pprint.pprint(response, width=1)
```

```
## {'ResponseMetadata': {'HTTPHeaders': {'connection': 'keep-alive',
##                                       'content-length': '169',
##                                       'content-type': 'application/json',
##                                       'date': 'Tue, '
##                                               '29 '
##                                               'Dec '
##                                               '2020 '
##                                               '03:56:53 '
##                                               'GMT',
##                                       'x-amz-apigw-id': 'YS9Y4FzloAMF1vw=',
##                                       'x-amzn-requestid': '155e519c-a2ed-46f5-9290-f2a139fca8eb',
##                                       'x-amzn-trace-id': 'Root=1-5feaa905-24a7e6352c158e9b0e5bcba6'}
##                        'HTTPStatusCode': 200,
##                        'RequestId': '155e519c-a2ed-46f5-9290-f2a139fca8eb',
##                        'RetryAttempts': 0},
##  'jobDefinitionArn': 'arn:aws:batch:us-east-1:710673677961:job-definition/pyfan-scatterline3-test-rm
##  'jobDefinitionName': 'pyfan-scatterline3-test-rmd',
##  'revision': 88}
```

1.3 Submit a Batch Array

Given the batch job definition that has been created. Create also Job Queues and related compute environments. Then we can run Batch Array. Upon submitting the batch array, you can monitor AWS EC2 instances, should notice potentially many instances of EC2 starting up. AWS is starting EC2 instances to complete the batch array jobs.

create a batch compute environment that uses spot price instances, which will be much cheaper than on demand costs. Will need to set proper AMI roles, *arn:aws:iam::XXXX7367XXXX:role/AmazonEC2SpotFleetRole* for *Spot fleet role*, and also proper securities.

When the *array_size* parameter is equal to 100, that starts 100 child processes, with 1 through 100 for *AWS_BATCH_JOB_ARRAY_INDEX*, which, could be used directly by the python function by taking in the parameter from the os environment as shown earlier. For demonstration purposes, will only set *array_size=3* in the example below.

Outputs from the scatterline3 has a timestamp, so each time the code below is run, will generate several new images, with the same set of random seeds, but different date prefix. The output s3 folder is public.

```
import boto3
import yaml
import pprint

import datetime as datetime

# Using the "jobDefinitionName": 'pyfan-scatterline3-test-rmd' from registering
jobDefinitionName = 'pyfan-scatterline3-test-rmd'

# How many child batch processes to start
# child process differ in: AWS_BATCH_JOB_ARRAY_INDEX
array_size = 3

# job name
timestr = "{:%Y%m%d%H%M%S%f}".format(datetime.datetime.now())
```

```
timesufx = '_' + timestr
st_jobName = jobDefinitionName + timesufx

# job queue (needs to design own queue in batch)
st_jobQueue = 'Spot'

# start batch service
# Load YAML file with security info
srn_aws_yml = "C:/Users/fan/fanwangecon.github.io/_data/aws.yml"
fl_yaml = open(srn_aws_yml)
ls_dict_yml = yaml.load(fl_yaml, Loader=yaml.BaseLoader)
aws_yml_dict_yml = ls_dict_yml[0]
# Use AWS Personal Access Keys etc to start boto3 client
aws_batch = boto3.client('batch',
                         aws_access_key_id=aws_yml_dict_yml['aws_access_key_id'],
                         aws_secret_access_key=aws_yml_dict_yml['aws_secret_access_key'],
                         region_name=aws_yml_dict_yml['region'])

# aws batch submit job
dc_json_batch_response = aws_batch.submit_job(
    jobName=st_jobName,
    jobQueue=st_jobQueue,
    arrayProperties={'size': array_size},
    jobDefinition=jobDefinitionName)

# Print response
pprint.pprint(dc_json_batch_response, width=1)
```

```
## {'ResponseMetadata': {'HTTPHeaders': {'connection': 'keep-alive',
##                                       'content-length': '198',
##                                       'content-type': 'application/json',
##                                       'date': 'Tue, '
##                                               '29 '
##                                               'Dec '
##                                               '2020 '
##                                               '03:56:54 '
##                                               'GMT',
##                                       'x-amz-apigw-id': 'YS9Y8HmIoAMFd0Q=',
##                                       'x-amzn-requestid': '15d3bfe3-bb83-4010-bcd1-1c497d2fd861',
##                                       'x-amzn-trace-id': 'Root=1-5feaa905-5d1441a14949ae277dc14544'}
##                        'HTTPStatusCode': 200,
##                        'RequestId': '15d3bfe3-bb83-4010-bcd1-1c497d2fd861',
##                        'RetryAttempts': 0},
##  'jobArn': 'arn:aws:batch:us-east-1:710673677961:job/d8d4cb4a-1352-40a0-8acf-07e2075f3e97',
##  'jobId': 'd8d4cb4a-1352-40a0-8acf-07e2075f3e97',
##  'jobName': 'pyfan-scatterline3-test-rmd_20201228215646005429'}
```

1.4 Track the Status of a Submitted Batch Array Until it Finished

To automate certain processes, often need to check and wait for job to complete. Can do this on web interface. Easier to do this via boto3 operations: describe_job and list_jobs

Given the batch array job we just generated above, first, parse out the job ID from the response from the batch array submission above. Then use list_jobs to check the length of JobSummaryList, and then use

describe_jobs to check overall job completion status.

```
import time
# Get Job ID
st_batch_jobID = dc_json_batch_response['jobId']
# Print Job ID
print(f'{st_batch_jobID=}')
# While loop to check status
```

```
## st_batch_jobID='d8d4cb4a-1352-40a0-8acf-07e2075f3e97'
```

```
bl_job_in_progress = True
it_wait_seconds = 0
while bl_job_in_progress and it_wait_seconds <= 600:
    # describe job
    dc_json_batch_describe_job_response = aws_batch.describe_jobs(jobs=[st_batch_jobID])
    # pprint.pprint(dc_json_batch_describe_job_response, width=1)
    it_array_size = dc_json_batch_describe_job_response['jobs'][0]['arrayProperties']['size']
    dc_status_summary = dc_json_batch_describe_job_response['jobs'][0]['arrayProperties']['statusSummary
    if dc_status_summary:
        # check status
        it_completed = dc_status_summary['SUCCEEDED'] + dc_status_summary['FAILED']
        if it_completed < it_array_size:
            bl_job_in_progress = True
            # sleep three seconds
            time.sleep(10)
            it_wait_seconds = it_wait_seconds + 10
        else:
            bl_job_in_progress = False

        print(f'{it_wait_seconds=}, ArrayN={it_array_size},' \
              f'SUCCEEDED={dc_status_summary["SUCCEEDED"]}, FAILED={dc_status_summary["FAILED"]}, ' \
              f'RUNNING={dc_status_summary["RUNNING"]}, PENDING={dc_status_summary["PENDING"]}, ' \
              f'RUNNABLE={dc_status_summary["RUNNABLE"]}')
    else:
        #empty statussummary
        bl_job_in_progress = True
        time.sleep(10)
        it_wait_seconds = it_wait_seconds + 10
        print(f'{it_wait_seconds=}, ArrayN={it_array_size}')
```

```
## it_wait_seconds=10, ArrayN=3
## it_wait_seconds=20, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=30, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=40, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=50, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=60, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=70, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=80, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=90, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=100, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=110, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=120, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=130, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=140, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=150, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
```

```
## it_wait_seconds=160, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=170, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=180, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=190, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=200, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=210, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=220, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=230, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=240, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=3
## it_wait_seconds=250, ArrayN=3,SUCCEEDED=0, FAILED=0, RUNNING=1, PENDING=0, RUNNABLE=0
## it_wait_seconds=250, ArrayN=3,SUCCEEDED=3, FAILED=0, RUNNING=0, PENDING=0, RUNNABLE=0
```