Hello!

CS 744: PYWREN

Shivaram Venkataraman
Fall 2022
Project checkins due Nov 23rd
Poster presentation: Dec 13th
Final report: Dec 20th

Project grade breakdown → 30%.
   Intro: 5%
   Mid-semester checkin: 5%
   Poster: 10%
   Final Report: 10%

one page write-up
is how things are going
Scalable Storage Systems

Datacenter Architecture

Applications
- Machine Learning
- SQL
- Streaming
- Graph

Computational Engines

Resource Management

DGL

Datacenter as a computer
NEW DATA, HARDWARE MODELS
Cloud computing
\[ \downarrow \text{data analytics} \]

Serverless Computing

\[ \lambda \]

\[ \downarrow \text{Machine Learning} \]

Compute Accelerators

New System designs

Infiniband Networks

Non-Volatile Memory

Cloud computing stack
SERVERLESS COMPUTING

not actually without servers
Motivation: Usability

What instance type?
What base image?
How many to spin up?
What price? Spot?

User who is not a CS major use this?

Virtual machine instances

User

Amazon EC2

Or 100 machines?
Cloud

When to use the Cloud?

Data
- Large amounts of data. Can't store locally
- Shared data across users
- Long term storage

Compute
- Need lots of CPUs for short times
- Varying compute needs
- No admin cost for servers

EE grads
MRI images

Push a button

Why is there no "cloud button"?
ABSTRACTION LEVEL?

Application ▲
Compute Framework ▼
Hardware ▲

Deep Learning Training
Logistic Regression
Word Count | Sort
Spark | PyTorch
Amazon EC2
CloudLab
Private Cluster
...

Pre-provisioned allocate, downloading, configuration etc.

Language Integrated
simple programming
model

Run

Application ▲
Compute Framework ▼
Cloud resources

...
“SERVERLESS” COMPUTING

Ephemeral compute resources
- Fixed configuration

300-900 seconds single-core

512-512-10240 MB in /tmp
- Upto fixed memory

3-10GB RAM

Python, Java, node.js, Ruby, Go etc.

Support for containers
- Docker to pull in dependencies

Google Cloud Platform
CLOUD FUNCTIONS ALPHA
A serverless platform for building event-based microservices
STATELESS DATA PROCESSING

Operators cannot save state each time output is produced. Storage engines persist state for compute layers.

Key Value Store (Low Latency)

Blob Store (High Bandwidth)

Function Scheduler

Operators launch compute tasks on containers.

Serverless computing (900s)

Launch compute tasks on containers.

Launch compute tasks on containers.

Launch compute tasks on containers.
Function Scheduler can run on your laptop

Application side

- Tracking which operator should be run next

- MRQ → map() finished | start reduce tasks

Infrastructure side (AWS)

- Decisions to minimize cost etc.

- Container

Which machine

Launch container( )

Resources that are use
PYWREN API

Configuration:
- cloud API key
- S3 bucket

```
import pywren
import numpy as np

def addone(x):
    return x + 1

wrenexec = pywren.default_executor()
xlist = np.arange(10)
futures = wrenexec.map(addone, xlist)

print [f.result() for f in futures]
```

The output is as expected:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```
**PYWREN: HOW IT WORKS**

```python
future = runner.map(fn, data)
serialize function & data
store it on S3
app scheduler
```

```
future.result()
```

![Diagram of how it works](#)

- **Your laptop**
  - Polling
  - Fetch result from cloud

- **The cloud**
  - Read the data function
  - Running the function
  - Output
  - Storage service
  - Parallelism
  - Storage
**HOW IT WORKS**

```python
future = runner.map(fn, data)
```

- Serialize func and data
- Put on S3
- Invoke Lambda

```
func
data
```

- pull job from s3
- download anaconda runtime
- python to run code
- pickle result
- stick in S3

```
result
```

- poll S3
- unpickle and return

```
your laptop
```

```
the cloud
```
**STATELESS FUNCTIONS: WHY NOW?**

What are the trade-offs?

- All data accesses are remote accesses.
- Network/remote access is competitive to reading data from SSD.

<table>
<thead>
<tr>
<th>Storage Medium</th>
<th>Write Speed (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD on c3.8xlarge</td>
<td>208.73</td>
</tr>
<tr>
<td>SSD on i2.8xlarge</td>
<td>460.36</td>
</tr>
<tr>
<td>4 SSDs on i2.8xlarge</td>
<td>1768.04</td>
</tr>
<tr>
<td>S3</td>
<td>501.13</td>
</tr>
</tbody>
</table>

Lower than 4 SSDs.
MAP AND REDUCE?

Input Data

Sort application

1000 map container

1000 reduce container

Intermediate data files

1M intermediate data files

Output Data

diff storage system

Redis → store intermediate data

λ → λ
Sparse ML models

subset parameters for 1 iteration

Use lambdas to run “workers”

Parameter server as a service?

Lambda/Container limited CPU/memory resources.

Lambda workers

update

get

Parameter Server

KV store PS API

trainers/workers

→
WHEN SHOULD WE USE SERVERLESS?

Yes!
- Tasks are independent and can be run in parallel
- When latency is not a concern (service provider queue)
- Cluster utilization is low

Maybe not?
- Long running tasks and checkpoint overheads
- Iterative algorithms reuse outputs from memory
- Code/Dependencies require long install and initialization

Shuttle intensive?
SUMMARY

Motivation: Usability of big data analytics
Approach: Language-integrated cloud computing

Features
- Breakdown computation into stateless functions
- Schedule on serverless containers
- Use external storage for state management

Open question on scheduling, overheads
DISCUSSION

https://forms.gle/GFIkkME52tvDRdDC8
Redis becomes a bottleneck as we scale num workers. The invocation time is small, but improvements with more workers are not linear. Speedup increases relatively as we scale num workers. In the chart, Redis read/write speedup increases with more workers. Setup time is also significant but remains constant with more workers. S3 read/write time improves with more workers, but not as significantly as Redis read/write. Compute time remains relatively constant across different configurations.
Consider you are a cloud provider (e.g., AWS) implementing support for serverless. What could be some of the new challenges in scheduling these workloads compared to schedulers we have studied in this class? How would you go about addressing them?

- Infrastructure scheduler
  - all containers "same compute" → balance network bandwidth use
  - Fairness?
    - SLO in terms of waiting time?
    - Pricing → higher tier for better responsiveness
  - Base Images might vary → very light → Distribute images
Happy Thanksgiving!

Next steps:

- Mid-semester project check-in, Nov 23rd
- After break: Owl, TPU papers
- Midterm 2, Dec 6\textsuperscript{th}