

Final Report in VQA Project (2/15-3/26)

Fixstars Solutions, Inc.

Summary of Our Works

- Work contents
 - Preparing code execution environment for compute resources: Cloud, clusters, and local workstations
 - Creating and modifying scripts to run experiments and visualize their status
 - Managing code execution
 - Managing costs on AWS
 - Transferring technical knowledge
 - Writing documentation
- Deliverables
 - Committing code and docs to gitlab repository
 - https://gitlab.com/tomotake/understanding_reasoning/-/tree/fixstars-dev

Compute Resources

- AWS (Amazon)
 - g4dn.2xlarge, use 60 instances
 - RAM: 32 GB
 - CPU: Intel Custom Cascade Lake
 - GPU: NVIDIA T4
- DGX-1 (MIT)
 - RAM: 500 GB
 - CPU: 40 x Intel Xeon(R) E5-2698 v4 @2.2GHz
 - GPU: 8 x NVIDIA P100 (16GB)
- DGX-1 (FSS)
 - RAM: 500 GB
 - CPU: 40 x Intel Xeon(R) E5-2698 v4 @2.2GHz
 - GPU: 8 x NVIDIA V100 (16GB)
- OpenMind (MIT)
 - CPU: Multiple Intel procs
 - GPU: Multiple GPUs
- Satori (MIT)
 - RAM: 1 TB
 - CPU: 40 x Power8
 - GPU: NVIDIA V100 (32GB)
- Speed comparison
 - AWS > DGX-1 (FSS) > DGX-1 (MIT) > OpenMind (normal) > OpenMind (cbmm)

Developing Scripts for Execution and Visualization

- Job submission script for ray
 - With singularity
 - Without singularity
- Instance preparation script for AWS
- Visualization script with dash

Live VQA Progress

Case in progress: experiment_1

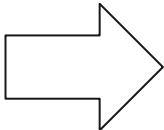
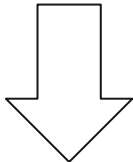
Current Time: 2021.02.16 - 17:58:02PM

Progress last updated at: 2021.02.15 - 19:39:47PM

Progress Table

Tag	Status	Elapsed Time
expIdx_500	Finished	3:47:05.718575
expIdx_516	Finished	2 days, 16:26:53.313691
expIdx_548	Finished	2 days, 16:26:48.303244
expIdx_1203	Finished	1 day, 18:59:24.801525

Why Do We Use Job Submission with Ray?

- Researchers' code for experiments
 - Python
 - PyTorch / TensorFlow with GPU
 - Prototyping code
 - Our expertise for software acceleration
 - Multiple cores
 - Multiple cpus
 - Multiple hosts
 - CPU / GPU / FPGA
- 
- We decided to parallelize code for a single experiment as job
- 
- We researched parallelization libraries and frameworks in Python

Comparison of Parallel Processing Libraries in Python

	multiprocessing (native)	concurrent.futures (native; >=3.2)	Joblib	Dask	Celery	Ray
Multi-threading	✗	✓	✓	✓	✗	✗
Multi-processing	✓	✓	✓	✓	✓	✓
Map func	✓	✓	✓	✓	✓	✓
Submit-fetch	✓	✓	✗	✓	✓	✓
Multi-hosts	✓	✗	✓	✓	✓	✓
Code transfer to worker	✓	✓	✓	✓	✗	✓
Keep results	✗	✗	✗	✗	✓	✗
Resource management	✗	✗	✗	✓	✓	✓
Management GUI	✗	✗	✗	✓	✓	✓
Fault tolerance	✗	✗	✗	✓	✓	✓
Support docker	✗	✗	✗	✓	✗	✓
Support Kubernetes	✗	✗	✗	✓	✗	✓



* Other libraries for huge data processing are also investigated and summarized at wiki pages

- Apache Airflow
- Kubernetes
- Nomad

Experiments and Costs

- Experiments patterns
 - Experiment 1
 - Trial 1 = 308
 - Experiment 2
 - Trial 1 = 270
 - Trial 2 = 270
 - Experiment 3
 - Trial 2 = 210
 - Experiment 4
 - Trial 1 = 210
 - Trial 2 = 118
- Total number of experiments
 - 1386 (1hr 2min to 10days 16hrs)
- Total time of calculation on DGX-1 (FSS)
 - 670 (hrs)

- Total time of calculation on AWS
 - Over 16,500 (hrs)
- Total costs on AWS
 - \$13,727.38



EC2	\$12,439.69
DataTransfer	\$35.21
kms	\$0.00
CloudTrail	\$0.00
Other Services	\$0.00
Tax	\$1,247.48
Total	\$13,722.38

This amount is only
for March 2021.
Feb 2021 cost is \$5.

Documentation and Technical Transfer

- Information Guide
 - Job submission operations with SLURM (OpenMind)
 - Job submission operations on AWS and DGX-1
- Session to Vanessa
 - Ray basic
 - Acceleration code on OpenMind
 - Changes to acceleration code for VQA
 - Job submission on OpenMind
 - Environment creation on DGX-1
 - Ray and job submission on DGX-1
- Ray basic introduction to biolins session
 - Ray basic

Confirming Eligibility in Satori Cluster

- Processor ... Power8
 - Code execution environment
 - Anaconda
 - The repository exists for power architecture.
 - Ray library
 - The latest version cannot be executed because its package is not provided and compilation does not succeed.
 - We can install the version of 0.8 from PowerAI channel, but it has issues.
 - PyTorch
 - Anaconda repository from IBM provides an older version that might be unstable.
 - The version of 1.3.0 can be compiled and it works well.
- ↓
- Without ray, we can run your code in Satori.



Summary of Parallelization Libraries and Frameworks

Low Level Parallelization

MultiProcessing

- Core parallelization library in Python
- Very easy to manage
- Supports distributive cluster, but, poorly.
- No in-built way for IPC

Joblib

- Easily parallelize from threading to hosts
- Interface to use other backends

DASK

- Easily parallelize from threading to hosts
- NumPy and Pandas are integrated
- Multi-process cluster is a little unstable

Ray

- Targeted towards ML parallelization
- Easily distribute remotes
- High Level APIs are only useful for particular scenarios.

Top Level Parallelization

Airflow

- Base on Python
- Excellent task management
- Concurrency size should be predefined

SageMaker

- Highly integrated user interface
- Run only on AWS

Container Orchestration

Kubernetes

Nomad