Scalable Reinforcement Learning on Cray Systems

Benjamin Robbins, Director - AI & Advanced Productivity
Ananda Kommaraju, Kristyn Maschhoff, Mike Ringenburg
Purpose: Demonstrate steps to reproduce a working distributed reinforcement learning configuration on a Cray system. Platform for further exploration.

Relevance: Reinforcement learning approaches have produced many of the recent state of the art results for Machine Learning. Reinforcement learning is resource and communication intensive and is, therefore, an excellent candidate to take advantage of high-performance-computing.

Results: Cray systems’ ability to support mixed node types (GPU, CPUs) and resource configuration flexibility make for an ideal platform to explore and push limits of reinforcement learning.
Recent Advances in AI

An animation of the gradient descent method predicting a structure for CASP13 target T1008
Reinforcement Learning

Learning from the rewards of a given action.
Reinforcement Learning (RL)

Day 19
I have successfully conditioned him to smile and write in his book every time I drool.
- Pavlov's dog.

Image: geekshumor.com
Reinforcement Learning (RL) Basics

Reinforcement Learning Illustration (https://i.stack.imgur.com/eoeSq.png)
The Goal of RL?

Find the optimal Policy.

The policy tells the agent how to act given a particular state.
Many RL Methods

Value Optimization
- DQN
- DDQN
- MMC
- PAL
- NAF

Policy Optimization
- Policy Gradient
- Actor Critic

Agent
- DFP

https://www.intel.ai/introducing-reinforcement-learning-coach-0-10-0/#gs.9190q
Distributed Reinforcement Learning

https://cdn-images-1.medium.com/max/1600/1*tYxWuyksovxA1Thu8PggPQ.jpeg
Motivation for Study of Distributed RL on Cray Systems

• Explore Reinforcement Learning algorithms
• Understand if possible to leverage existing open source frameworks
• Study nested parallelism and complex workflows
• Distributed RL is very active area of research. Participate in research and make available for other Cray end-users
• Scalability is a requirement for many real-world problems. How can distributed RL at the scale of Cray help solve these problems?
Environments for Study
Distributed Reinforcement Learning Generalized

Worker Node

Trained Policy

Actions

Experiences

Weights

Samples

Target Policy

Centralized Learner/Policy

Worker Node

Trained Policy

Actions

Experiences

Weights

Samples

Individual Environment

Worker Node

Individual Environment
Setup and Configuration
Mapping RL to Cray XC

• Nested Parallelism
• Resource Scaling
• Multiple algorithms can be mapped
What is Ray?

- Ray is a flexible, high-performance distributed execution framework.
- Developed by RISELab at UC Berkeley
- Ray has libraries for tuning and reinforcement learning
- RLlib is Ray’s reinforcement learning library
- RLlib supports many RL methods.
Deploy a Ray Cluster on Cray XC

• Allocate nodes through SLURM workload manager
  
  ```bash
  $salloc -N 1 -p ccm_queue -C P100 --gres=gpu --exclusive
  $module load ccm
  $ccmlogin -V
  ```

• ccmlogin (Cluster compatible mode)

• Start a Ray head node with initial settings

  ```bash
  #!/bin/bash
  source activate ray
  IP=$(ip -oneline -family inet addr list ipogif0 \  | head --lines 1 | grep --perl-regexp \  --only-matching 'inet \K\d.']
  echo $IP:6380 > $HOME/ray_head_node

  ray start --head --node-ip-address=$IP --redis-port =6380
  ```
Deploy a Ray Cluster on Cray XC

- Independently start Ray workers with initial settings
- Resource scaling - Increase or reduce workers/resources

```bash
$salloc -N 1 -C P100 --gres=gpu --exclusive
$srun sh worker.sh

#!/bin/bash
source activate ray
HEAD_IP=$(head -n 1 ray_head_node)
ray start --redis-address $HEAD_IP
while [ 1 ];
do
    sleep 1
done
```
Execute Rllib algorithms

```python
import ray
from ray.rllib.agents.ppo import PPOAgent
from ray.tune import run_experiments

def train_fn(config, reporter):
    agent1 = PPOAgent(env="CartPole-v0", config=config)
    for _ in range(100000):
        result = agent1.train()
        result["phase"] = 1
        reporter(**result)
        phase1_time = result["timesteps_total"]
        state = agent1.save()
        agent1.stop()
    if __name__ == "__main__":
        ray.init(redis_address="10.128.0.225:6380")
        run_experiments({
            "demo": {
                "run": train_fn,
                "local_dir": "/lus/scratch/user/ray_results/custom/",
                "config": {
                    "lr": 0.01,
                    "num_workers": 64,
                },
            },
        })
```

```
RL Methods and Results
RL Method – Actor/Critic

[Image of a child drawing on a wall with an adult watching]

Actor

Critic

https://www.cleaninginstitute.org/sites/default/files/assets/1/Photos/700x700/child-coloring-wall.jpg
Distributed RL Method – A3C

Asynchronous Advantage Actor Critic

1. Worker reset to global network
2. Worker interacts with environment
3. Worker calculates value and policy loss
4. Worker gets gradients from losses
5. Worker updates global network with gradients

https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c8f72a5e9f2
Distributed RL Method – A3C vs A2C

A3C (Async)

Training in parallel

Global Network Parameters

Agent 1

Agent 2

Agent 3

... 

Agent n

A2C (Sync)

Training in parallel

Global Network Parameters

Coordinator

Agent 1

Agent 2

Agent 3

... 

Agent n

https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#a2c
Multi-Node Scaling

<table>
<thead>
<tr>
<th>Qbert</th>
<th>Reward Mean</th>
<th>Timestep</th>
<th>Waittime</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2C_16w_1gpu_20cpu</td>
<td>2589</td>
<td>10M</td>
<td>43m 21s</td>
</tr>
<tr>
<td>A2C_1w_1gpu_36cpu</td>
<td>3561</td>
<td>10M</td>
<td>4h 50m 8s</td>
</tr>
<tr>
<td>A2C_2w_1gpu_36cpu</td>
<td>1621</td>
<td>10M</td>
<td>2h 40m 39s</td>
</tr>
<tr>
<td>A2C_4w_1gpu_36cpu</td>
<td>2106</td>
<td>10M</td>
<td>1h 33m 25s</td>
</tr>
<tr>
<td>A2C_8w_1gpu_36cpu</td>
<td>2155</td>
<td>10M</td>
<td>59m 6s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Qbert</th>
<th>Reward Mean</th>
<th>Timestep</th>
<th>Waittime</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPO_16w_1gpu_20cpu</td>
<td>8211</td>
<td>10M</td>
<td>1h 23m 1s</td>
</tr>
<tr>
<td>PPO_1w_1gpu_36cpu</td>
<td>1.1095e+4</td>
<td>10M</td>
<td>5h 16m 27s</td>
</tr>
<tr>
<td>PPO_2w_1gpu_36cpu</td>
<td>1.1662e+4</td>
<td>10M</td>
<td>3h 25m 3s</td>
</tr>
<tr>
<td>PPO_4w_1gpu_36cpu</td>
<td>1.117e+4</td>
<td>10M</td>
<td>3h 11m 9s</td>
</tr>
<tr>
<td>PPO_8w_1gpu_36cpu</td>
<td>8068</td>
<td>10M</td>
<td>2h 0m 41s</td>
</tr>
</tbody>
</table>
Distributed RL Method – Ape-X

Learner
Network

Replay
Experiences

Actor
Network
Environment

Sampled experience
Updated priorities

Network parameters
Initial priorities
Generated experience

https://openreview.net/pdf?id=H1Dy---OZ
Distributed RL Method - IMPALA

Importance Weighted Actor Learner Architecture

Observations are trajectories of experience (sequence of states, actions, and rewards) not gradients.

IMPALA and Ape-X

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Reward Mean</th>
<th>Timestep</th>
<th>Walltime</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPALA_Beamrider</td>
<td>2173</td>
<td>10M</td>
<td>28m 18s</td>
</tr>
<tr>
<td>IMPALA_Breakout</td>
<td>374</td>
<td>10M</td>
<td>28m 14s</td>
</tr>
<tr>
<td>IMPALA_Qbert</td>
<td>1077</td>
<td>10M</td>
<td>28m 18s</td>
</tr>
<tr>
<td>IMPALA_Spacelvader</td>
<td>747</td>
<td>10M</td>
<td>28m 9s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Reward Mean</th>
<th>Timestep</th>
<th>Walltime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ape-X_Beamrider</td>
<td>2122</td>
<td>10M</td>
<td>1h 2m 44s</td>
</tr>
<tr>
<td>Ape-X_Breakout</td>
<td>84.11</td>
<td>10M</td>
<td>1h 4m 17s</td>
</tr>
<tr>
<td>Ape-X_Qbert</td>
<td>2989</td>
<td>10M</td>
<td>1h 4m 54s</td>
</tr>
<tr>
<td>Ape-X_Spacelvader</td>
<td>644.9</td>
<td>10M</td>
<td>56m 25s</td>
</tr>
</tbody>
</table>
Mapping RL to Cray CS

- 8 GPUs total
- Assign 1 GPU as Head Node
- 7 GPUs as Worker Nodes
- 9 CPU cores per Worker
XC vs Dense GPU Node on CS

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Reward Mean</th>
<th>Timestep</th>
<th>Walltime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qbert_CS_A2C</td>
<td>1.5101e+4</td>
<td>10M</td>
<td>1h 35m 54s</td>
</tr>
<tr>
<td>Qbert_CS_PPO</td>
<td>1.2775e+4</td>
<td>10M</td>
<td>2h 24m 15s</td>
</tr>
<tr>
<td>Qbert_XC_A2C</td>
<td>4193</td>
<td>10M</td>
<td>1h 11m 48s</td>
</tr>
<tr>
<td>Qbert_XC_Ppo</td>
<td>6998</td>
<td>10M</td>
<td>2h 1m 40s</td>
</tr>
</tbody>
</table>
Key Results

- Explored Reinforcement Learning as a HPC workload
- Deploy a UC Berkeley’s RISElab’s Ray cluster on XC system
- Trained state-of-art RL agents
- UC Berkeley’s RLib using Ray’s distributed execution
- IMPALA, Ape-X, PPO, A2C/A3C on Atari games (from gym library)
- Variety of resource configurations
- Scaled training on multi-node and single-node XC with mixed CPU and GPU
What’s Next?

- Explore Optimization of Ray Libraries
- Identify problems sets that map well to Distributed RL
- Comparative studies against other published results
- Explore architecture needs for computational node support and future network requirements
SAFE HARBOR STATEMENT

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts.

These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.
THANK YOU

QUESTIONS?

brobbins@cray.com

cray.com
@cray_inc
linkedin.com/company/cray-inc-