Introduction To Ray

A Distributed Computing Framework

Bhagirathi Hegde

Sarath Srinivas

Simple Batch Inference

```python
def predict(image_batch):
    label_list = []
    for image in image_batch:
        try:
            img = Image.open(image)
            processor = ViTImageProcessor.from_pretrained("google/vit-base-patch16-224")
            model = ViTForImageClassification.from_pretrained("google/vit-base-patch16-224")
            inputs = processor(images=img, return_tensors="pt")
            outputs = model(**inputs)
            logits = outputs.logits
            predicted_class_idx = logits.argmax(-1).item()
            label_list.append(model.config.id2label[predicted_class_idx])
        except ValueError as e: ...
    return label_list


if __name__ == "__main__":
    images = glob.glob(f"{DATA_FOLDER}*.JPEG")[:300]
    # Split into chunks of 15 images each
    image_batches = [images[i:i + 15] for i in range(0, len(images), 15)]
    st = time.perf_counter()
    results = [predict(image_batch) for image_batch in image_batches]
    en = time.perf_counter()
    print("Inference Throughput (images/sec): ", len(images) / (en - st))
```

Program Execution

```
ray@805d0f20ec54:~/ray-scaling-experiments$ python3 scripts/simple_batch_inference.py
Inference Throughput (images/sec):  0.7533543410439415
```

Single processor

```
ray@805d0f20ec54:~/ray-scaling-experiments$ python3 scripts/simple_batch_inference.py
Inference Throughput (images/sec):  2.1059700257584897
```

Multiprocessing on 4 cores

```
ray@805d0f20ec54:~/ray-scaling-experiments$ python3 scripts/simple_batch_inference.py
2024-05-19 22:35:18,377 INFO packaging.py:530 -- Creating a file package for local directory '/home/ray/ray-scaling-exper
2024-05-19 22:35:18,463 INFO packaging.py:358 -- Pushing file package 'gcs://_ray_pkg_ab7f7809f7bd9c09.zip' (8.36MiB) to
2024-05-19 22:35:20,838 INFO packaging.py:371 -- Successfully pushed file package 'gcs://_ray_pkg_ab7f7809f7bd9c09.zip'.
Inference Throughput (images/sec):  10.026132188471529
```

Ray cluster with 16 cores (multiple machines)

How to scale?

Distributed Execution

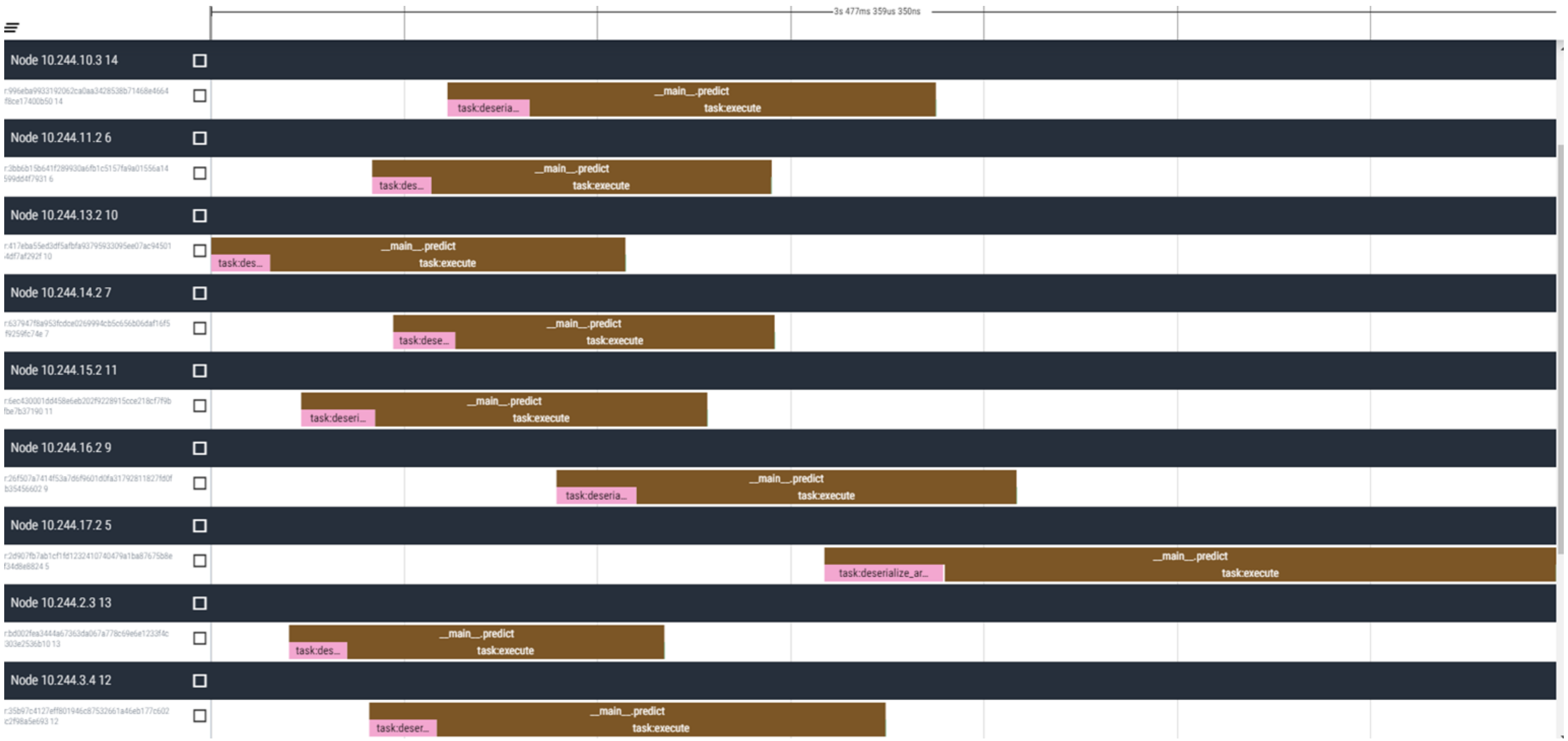
Ray distributes tasks across a cluster of machines, accelerating workflows

1 task : 15 imgs
300 imgs in total
So, 20 tasks
distributed
across 16 cores

- Tasks in Ray can be any arbitrary functions that are executed asynchronously on separate workers
- So multiple tasks can execute simultaneously

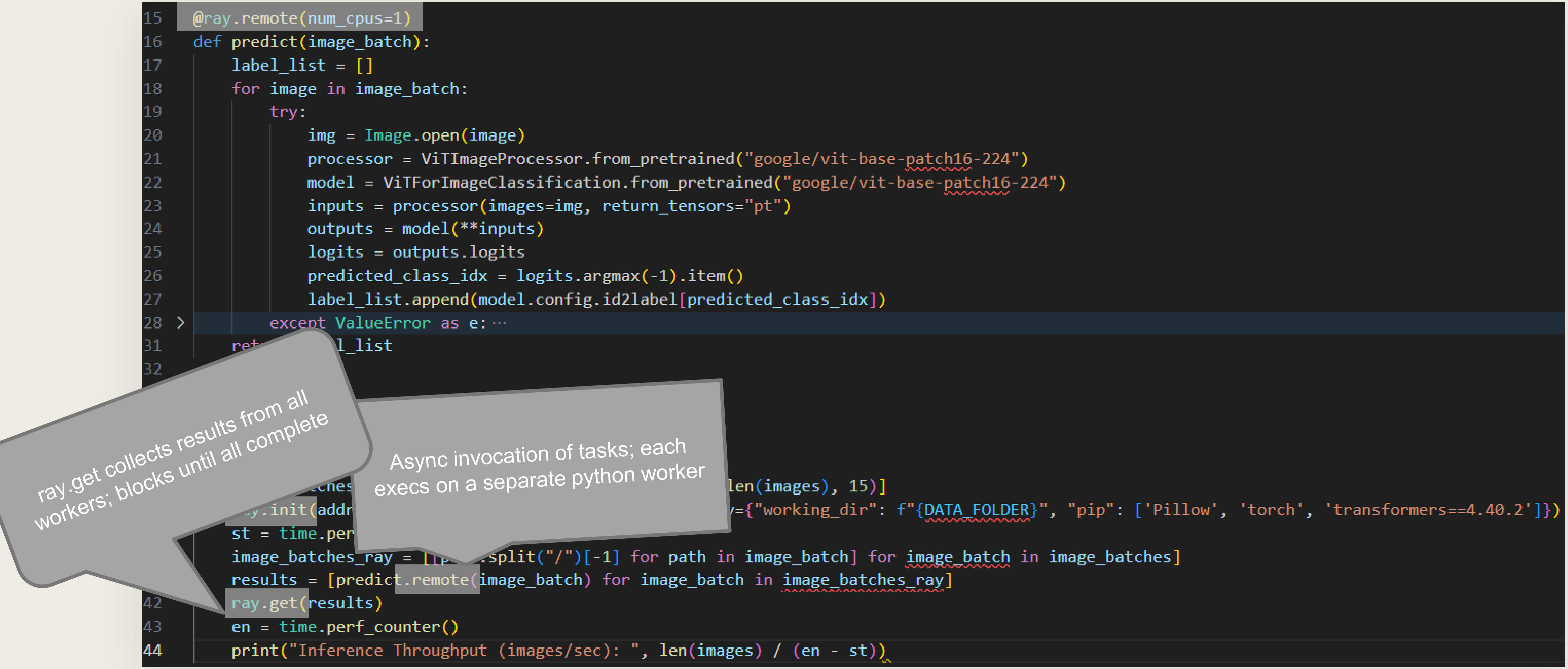
Distribution overhead

Amortized for big enough tasks

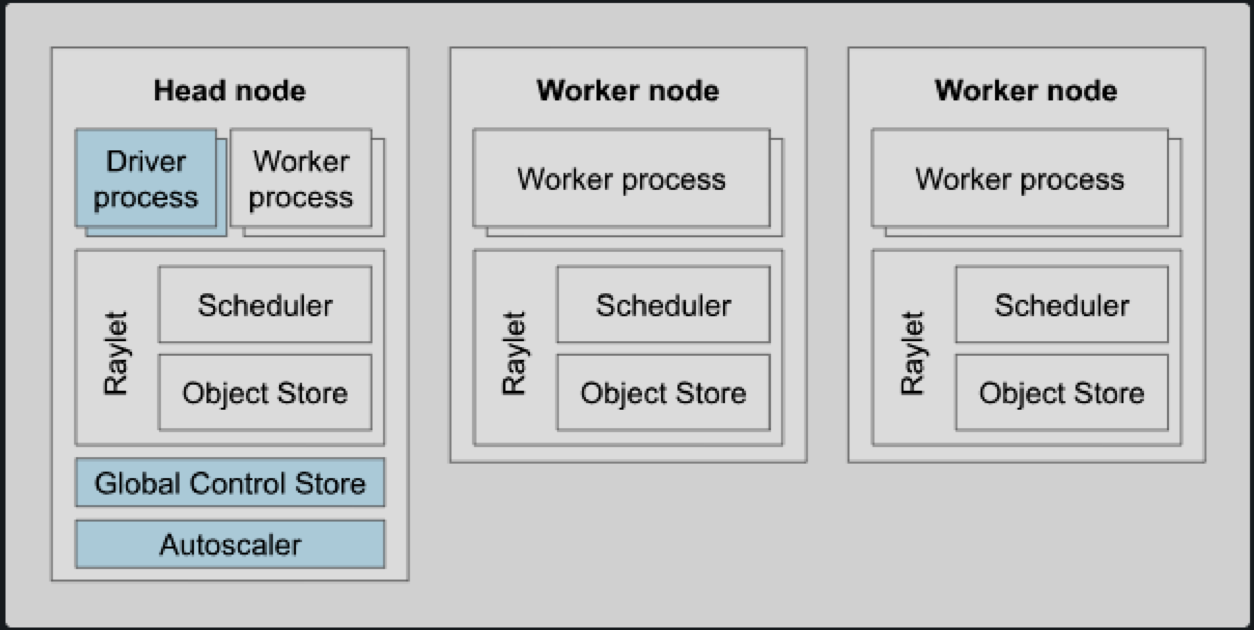



```
ray@805d0f20ec54:~/ray-scaling-experiments$ python3 scripts/simple_batch_inference.py
Inference Throughput (images/sec):  1.5786928062833752
```

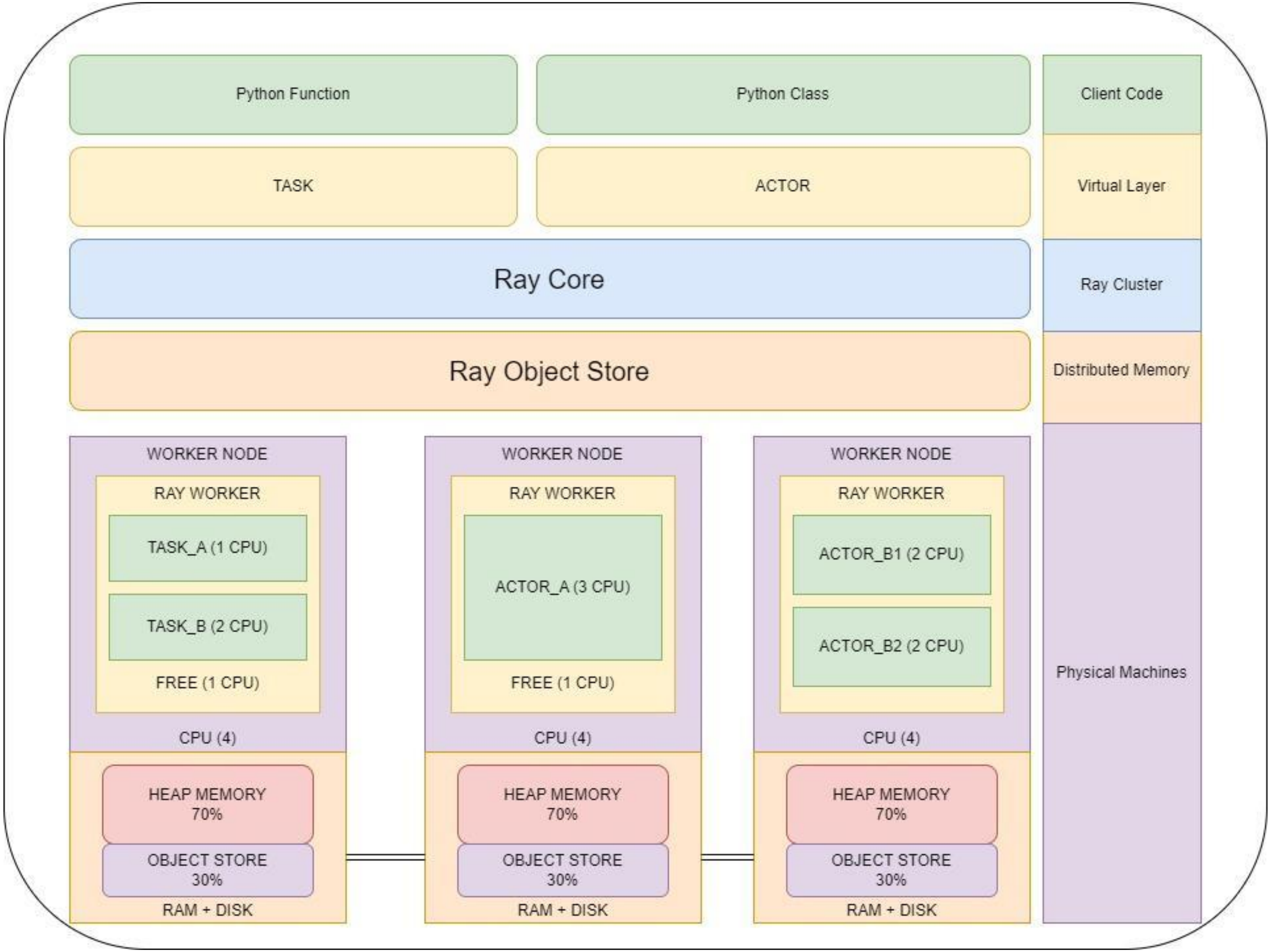
The only code changes



Ray cluster



A Ray cluster with two worker nodes. Each node runs Ray helper processes to facilitate distributed scheduling and memory management. The head node runs additional control processes (highlighted in blue).



Python Function
Python Class
Client Code
TASK
ACTOR
Virtual Layer
Ray Core
Ray Cluster
Ray Object Store
Distributed Memory
WORKER NODE
RAY WORKER
TASK_A (1 CPU)
TASK_B (2 CPU)
FREE (1 CPU)
CPU (4)
HEAP MEMORY 70%
OBJECT STORE 30%
RAM + DISK
WORKER NODE
RAY WORKER
ACTOR_A (3 CPU)
FREE (1 CPU)
CPU (4)
HEAP MEMORY 70%
OBJECT STORE 30%
RAM + DISK
WORKER NODE
RAY WORKER
ACTOR_B1 (2 CPU)
ACTOR_B2 (2 CPU)
CPU (4)
HEAP MEMORY 70%
OBJECT STORE 30%
RAM + DISK
Physical Machines

Starting a ray cluster is very simple

cluster.yaml

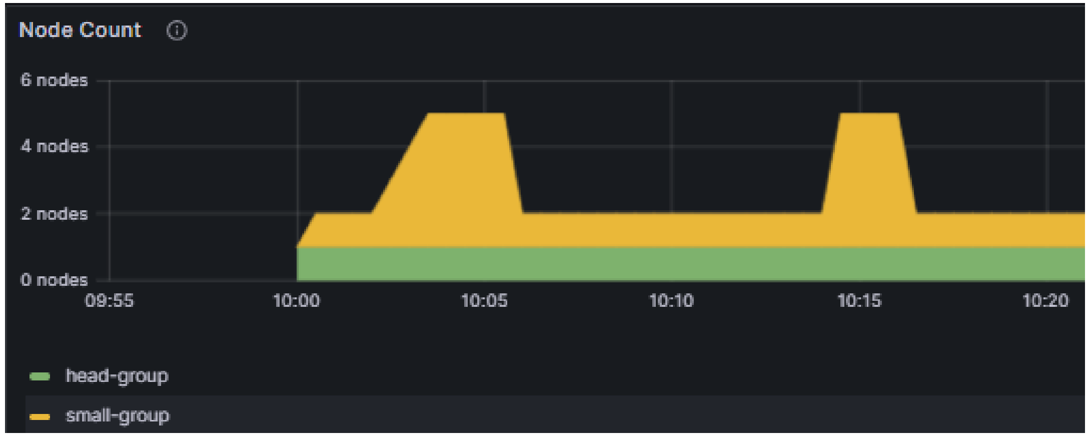
```yaml
cluster_name: default

provider:
   type: local
   head_ip: 20.106.179.167
   worker_ips: [20.127.236.169, 20.127.238.96]

# How Ray will authenticate with newly launched nodes.
auth:
   ssh_user: ray-admin
   ssh_private_key: ~/.ssh/id_rsa

# Command to start ray on the head node. You don't need to change this.
head_start_ray_commands:
   - ray stop
   - ulimit -c unlimited && ray start --head --port=6379 --num-cpus=0

# Command to start ray on worker nodes. You don't need to change this.
worker_start_ray_commands:
   - ray stop
   - ray start --address=$RAY_HEAD_IP:6379  --num-cpus=NUM_CPUS_VAR
```

```
ray-admin@ray-head-vm:~/ray-scaling-experiments/deploy$ ray up cluster.yaml
```

Autoscaler with kuberay

Ray scales up and down resources dynamically based on requirements/requests

```
ray@805d0f20ec54:~$ python3 ray-scaling-experiments/scripts/ray_core.py 10
SIGTERM handler is not set because current thread is not the main thread.
(autoscaler +6s) Tip: use `ray status` to view detailed cluster status. To
(autoscaler +6s) Adding 4 node(s) of type small-group.
(autoscaler +21s) Resized to 3 CPUs.
(autoscaler +26s) Resized to 4 CPUs.
```



Run Hybrid Workloads

By using fractional CPU for I/O Bound tasks

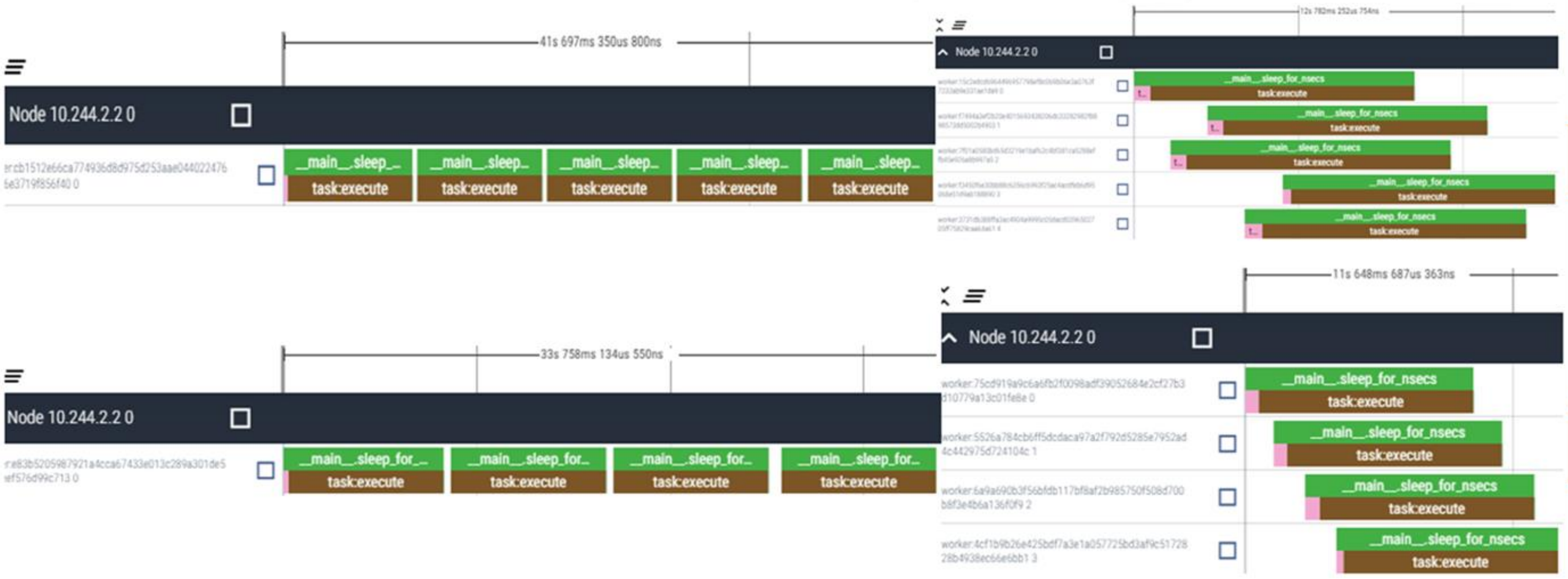
~0% cpu utilization with fractional cpu req

ng-experiments / scripts / io_bound.py

Blame 41 lines (28 loc) · 956 Byte

```python
NUM_CPUS = 0.1  # Fractional CPU
NSECS = 8


@ray.remote(num_cpus=NUM_CPUS)
def sleep_for_nsecs(n):
    time.sleep(n)
    return "done sleeping"
```

Task Timelines: Blocking vs. Non-Blocking sleep



5 concurrent sleeps

4 concurrent sleeps

Benefits of Ray over other distributed computing options

RAY

- Python-centric
- Flexible and general-purpose
- Scales efficiently
- Optimized for real-time and low-latency applications

vs

Spark

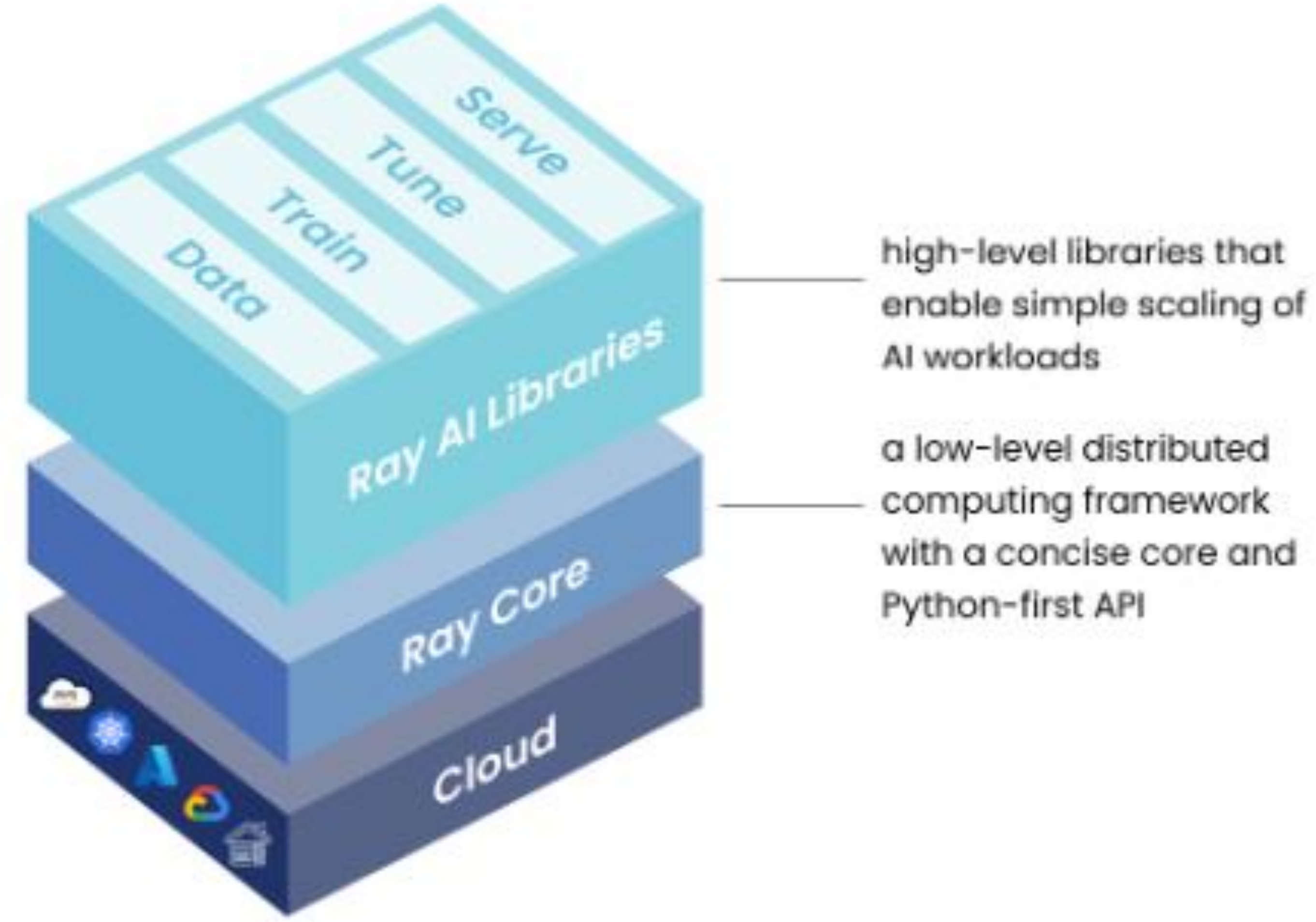
- MapReduce centric, dataflow-based – limited task flexibility
- Natively scala

Complex, Limited task management

dask

- Mostly Data Parallel/ Mapreduce oriented

Ray Frameworks



Summary

- Ray is a distributed computing framework for scaling up Python applications.
- Benefits:
  - Enables parallel execution of tasks across multiple machines or cores.
  - Simplifies building scalable and fault-tolerant applications.
  - Offers a user-friendly API for distributed computing.

From program on laptop to a high-performance distributed application with relatively few additional lines of python code

Resources

- Ray documentation
- Our GitHub repo