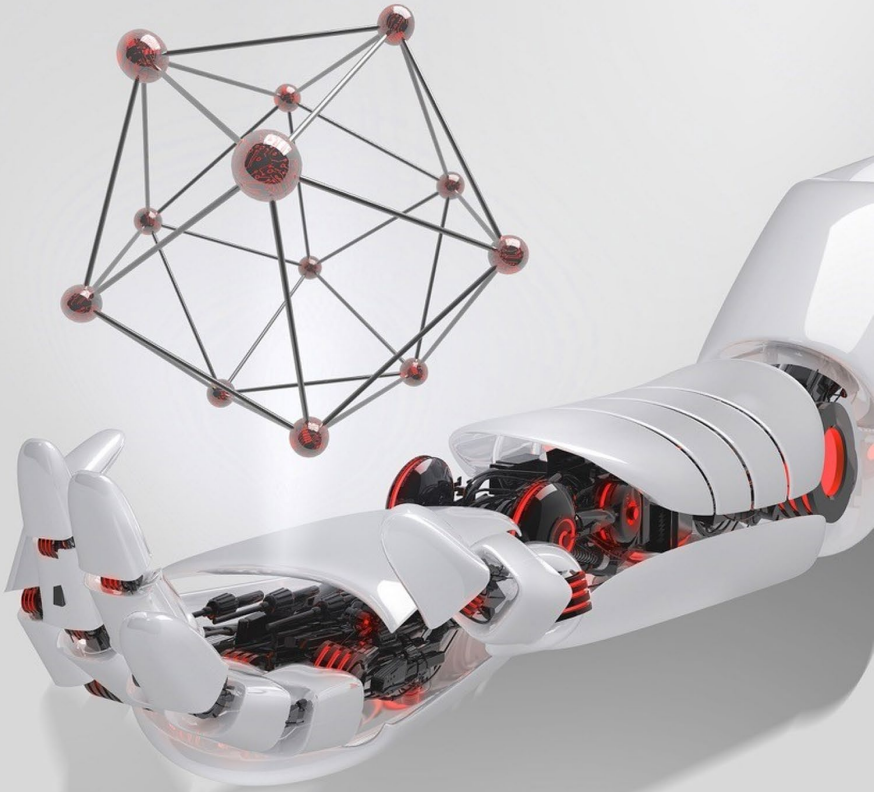
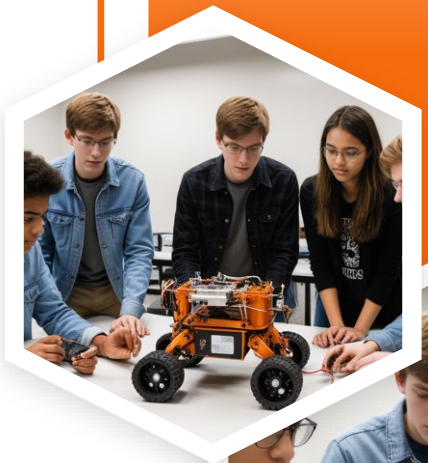
Ray 在分布式视频数据处理中的实践

演讲人：吕召刚

阿里巴巴 通义实验室-应用视觉 架构师



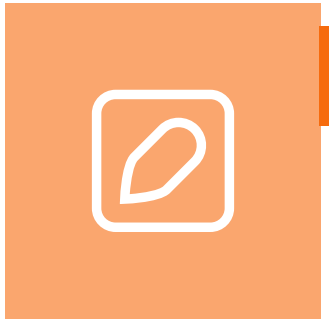
CONTENTS



01

背景

视频模型训练的数据预处理

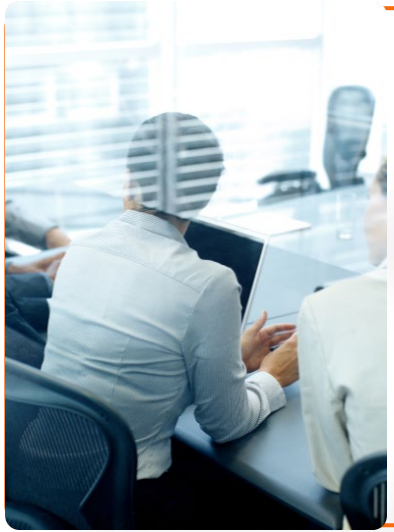


模型训练使用的数据直接影响到最终模型的效果

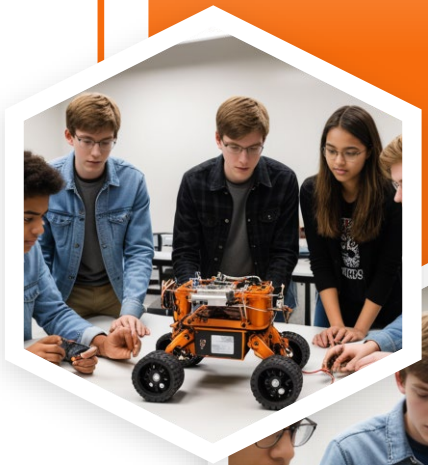
大量视频数据需要处理

分布式处理的优势

提高数据处理速度，加快模型训练过程



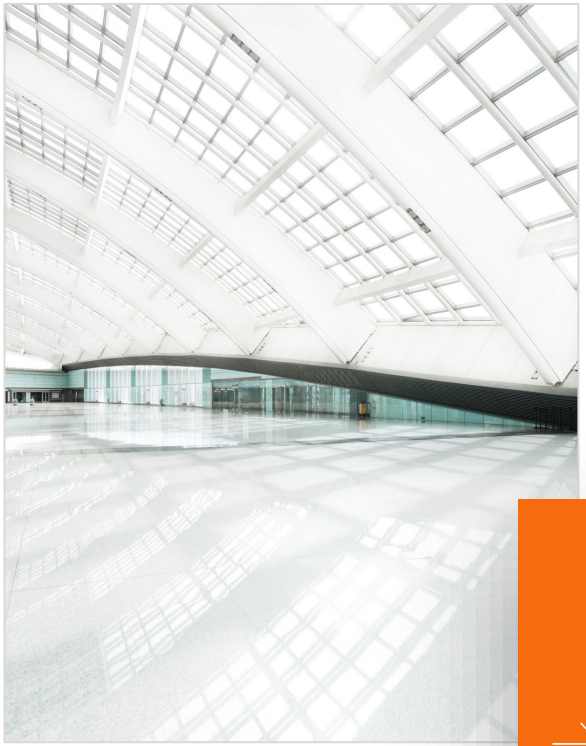
充分利用现有的硬件资源，提高资源利用率



02

视频预处理遇到的问题

时间要求的挑战



单机处理视频数据耗时过长，无法满足快速迭代的需求

基于 Ray 的分布式处理能够显著缩短数据处理时间，提升效率

算力要求的挑战

01

视频数据处理需要大量的CPU/GPU资源，单机无法满足

02

基于 Ray 的分布式系统可以动态分配资源，按需扩展 CPU/GPU 算力

算法多样化的挑战

视频预处理涉及多种算法，不同算法有不同的环境依赖和硬件需求

Ray 分布式系统支持算法的并行执行和动态调度，提高处理灵活性

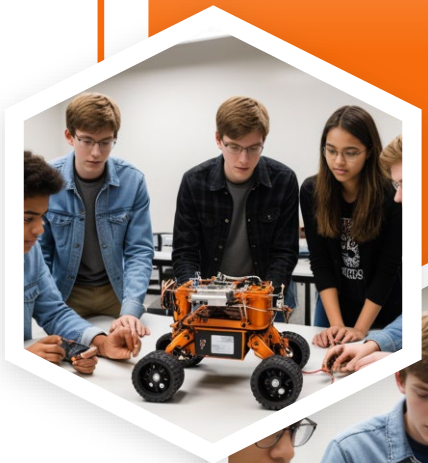
资源有限的挑战

01

GPU资源有限，需要充分利用GPU，提升大规模的视频数据处理效率

02

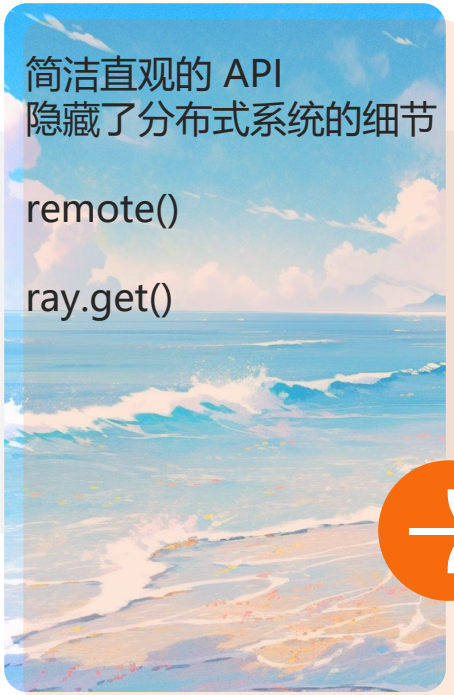
基于 Ray 的分布式系统可以极限利用显卡资源



03

Ray 框架用于数据处理的优点

简单的API设计




```python
import ray

# 初始化 Ray
ray.init()

# 定义一个远程函数
@ray.remote
def square(x):
    return x * x

# 提交多个任务
results = [square.remote(i) for i in range(10)]

# 获取结果
final_results = ray.get(results)
print(final_results)  # 输出 [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

算法集成的便利性

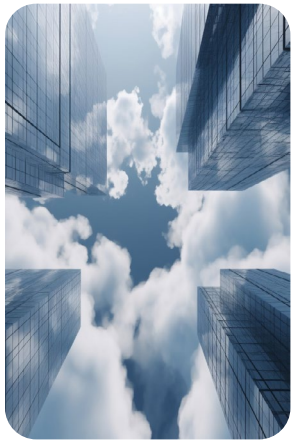
01.

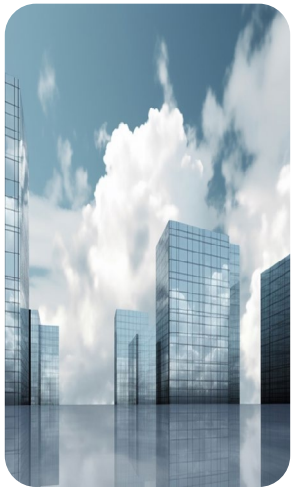
Ray框架能够与多种视频处理算法无缝集成，提供统一的处理接口。

02.

通过Ray，可以实现算法的快速部署和灵活调度，提高处理效率。

不同硬件资源的灵活调度





01

Ray框架支持对不同硬件资源的灵活调度：CPU，GPU，内存，磁盘

02

通过Ray，可以根据任务需求动态调整资源分配，优化资源使用

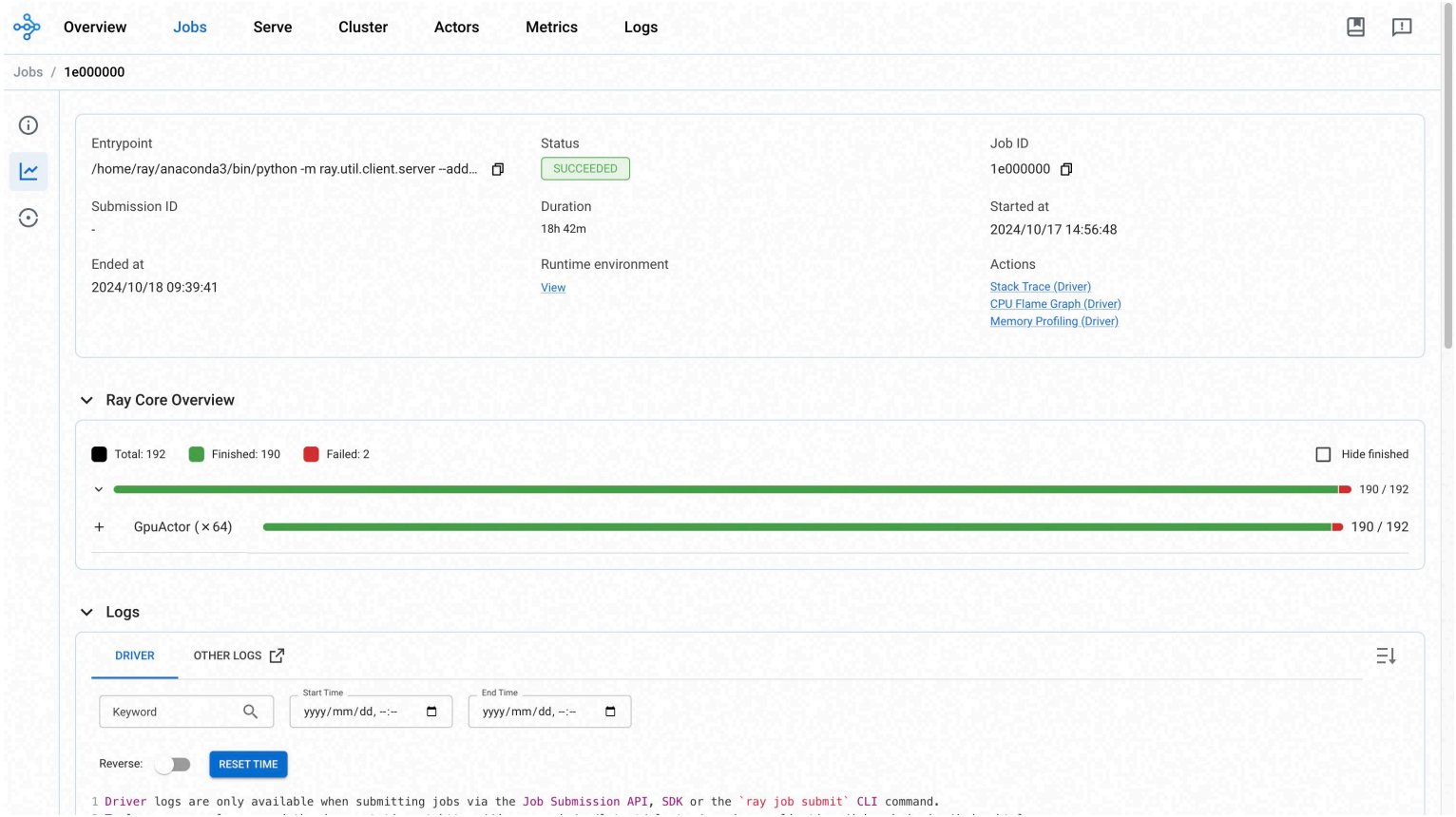
算法模型的按需加载

通过任务调度按需加载不同的算法模型，实现算法的快速切换和灵活组合

VS

任务调度系统实现数据批处理，一个步骤使用单个模型分布式批量处理所有数据

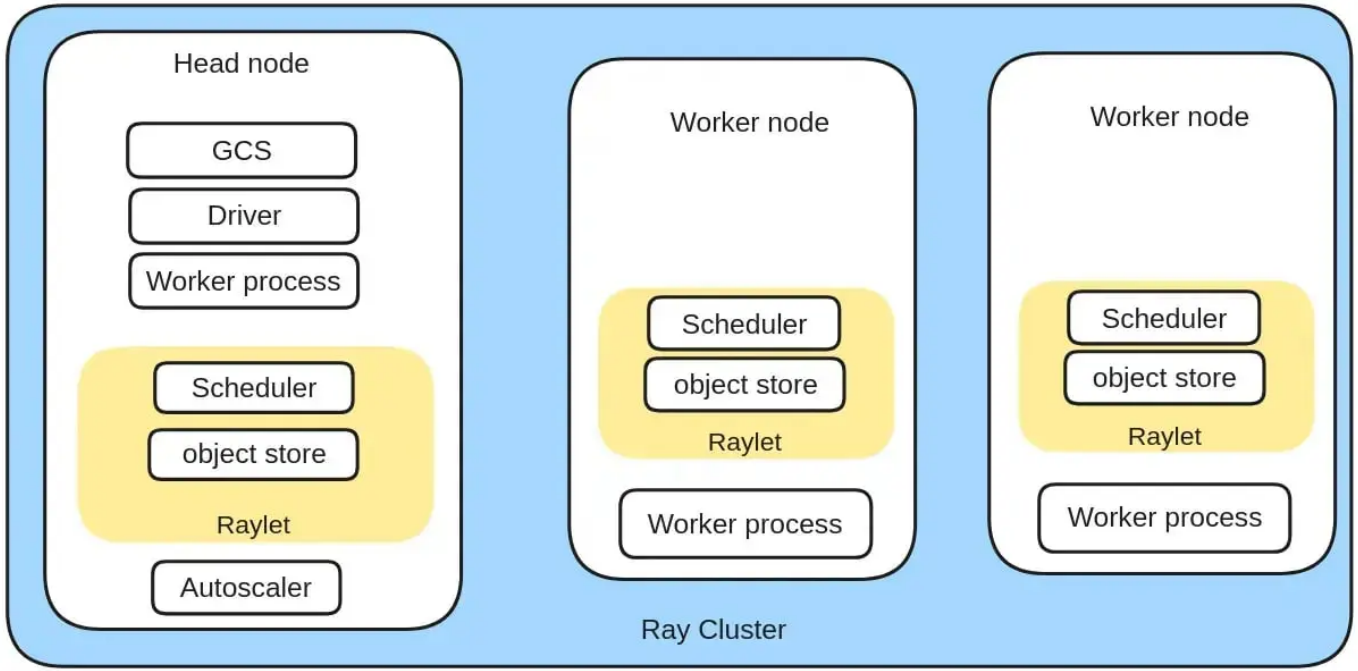
任务可扩展，可观测


Overview
Jobs
Serve
Cluster
Actors
Metrics
Logs
Jobs / 1e000000
Entrypoint
/home/ray/anaconda3/bin/python -m ray.util.client.server --add...
Status
SUCCEEDED
Job ID
1e000000
Submission ID
-
Duration
18h 42m
Started at
2024/10/17 14:56:48
Ended at
2024/10/18 09:39:41
Runtime environment
View
Actions
Stack Trace (Driver)
CPU Flame Graph (Driver)
Memory Profiling (Driver)
Ray Core Overview
Total: 192
Finished: 190
Failed: 2
Hide finished
190 / 192
GpuActor (× 64)
190 / 192
Logs
DRIVER
OTHER LOGS
Keyword
Start Time
yyyy/mm/dd, --:--
End Time
yyyy/mm/dd, --:--
Reverse:
RESET TIME
1 Driver logs are only available when submitting jobs via the Job Submission API, SDK or the `ray job submit` CLI command.

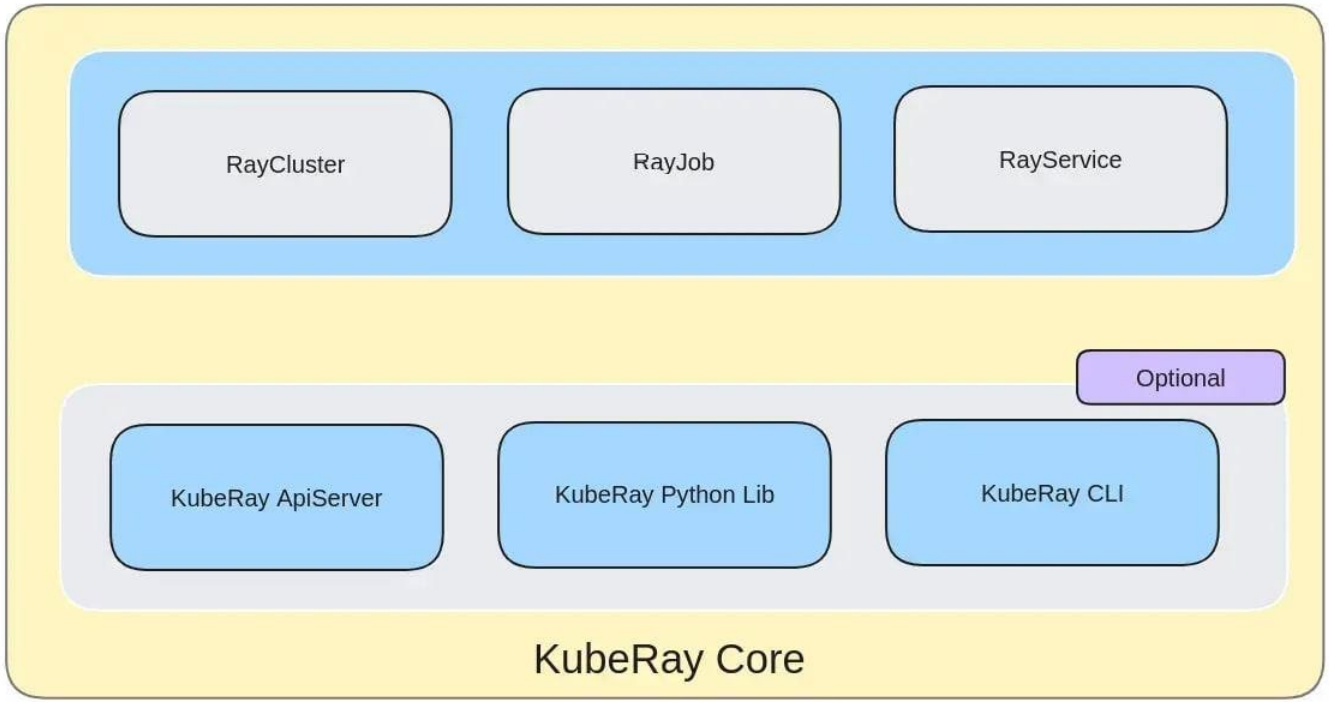
04

基于 K8S 的 Ray 架构

Ray 集群的组成


Head node
GCS
Driver
Worker process
Scheduler
object store
Raylet
Autoscaler
Worker node
Scheduler
object store
Raylet
Worker process
Worker node
Scheduler
object store
Raylet
Worker process
Ray Cluster

KubeRay 部署架构



KubeRay 任务管理方案的选择

客户端命令行提交的问题

安全性不足

可维护性差：版本一致性要求高

- Python 版本
- Ray 版本

集群内命令行提交的易用性

安全性不足：需要开放 k8s

易用性差：开发人员直连 k8s/ray

灵活度高：可以使用 ray 命令查询集群状态

管理控制台的Web界面提交

开发管理控制台的Web界面

可以方便地向Ray集群提交任务

缺点：复杂性提高

需要考虑：

- **如何与Ray集群通信，**
- **如何将任务提交到多个集群**
- **任务状态管理**

KubeRay 任务管理： 从客户端命令行提交任务

Kubernetes 开放 Ray 集群管理端口

客户端连接到 Ray 端口，提交任务

```
~/workspace/video
RAY_ADDRESS='https://ray1.xxxx.aliyun.com:8265' ray job submit --working-dir . -- python simple_task.py
```

KubeRay 任务管理： 集群内命令行提交任务

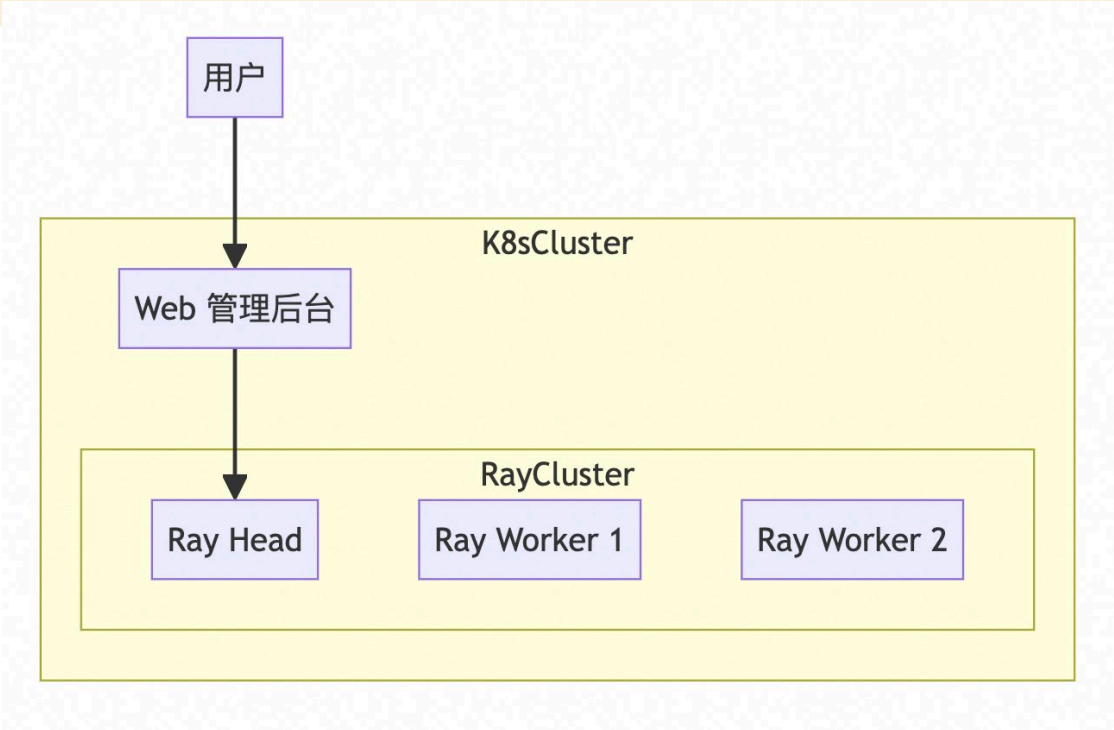
登陆到 k8s ray head 节点容器，使用 Ray 命令提交

前提条件：代码已经存放在 Ray Head 节点容器上

```
→  ~ kubectl --kubeconfig=$DATA_[redacted] --namespace talking-head exec -it raycluster-autoscaler-head-4fbhp -- /bin/bas
h
Defaulted container "ray-head" out of: ray-head, autoscaler
(base) ray@raycluster-autoscaler-head-4fbhp:~$ ray job list | wc
     30     788   18583
(base) ray@raycluster-autoscaler-head-4fbhp:~$ ray job list | grep "job_id='1e000000'"
 JobDetails(type=<JobType.DRIVER: 'DRIVER'>, job_id='1e000000', submission_id=None, driver_info=DriverInfo(id='1e0
00000', node_ip_address='172.16.10.186', pid='3855188'), status=<JobStatus.SUCCEEDED: 'SUCCEEDED'>, entrypoint='/h
ome/ray/anaconda3/bin/python -m ray.util.client.server --address=172.16.10.186:6379 --host=0.0.0.0 --port=23028 --
mode=specific-server --redis-password=[redacted], message=None, error_type=None, start_time=1729148208088,
 end_time=1729215581400, metadata={}, runtime_env={'working_dir': 'gcs://_ray_pkg_5731ec9218356e42.zip'}, driver_a
gent_http_address=None, driver_node_id=None, driver_exit_code=None)]
(base) ray@raycluster-autoscaler-head-4fbhp:~$ export RAY_ADDRESS="http://127.0.0.1:8265"
(base) ray@raycluster-autoscaler-head-4fbhp:~$ ray job submit --working-dir /data/project -- python script.py
```

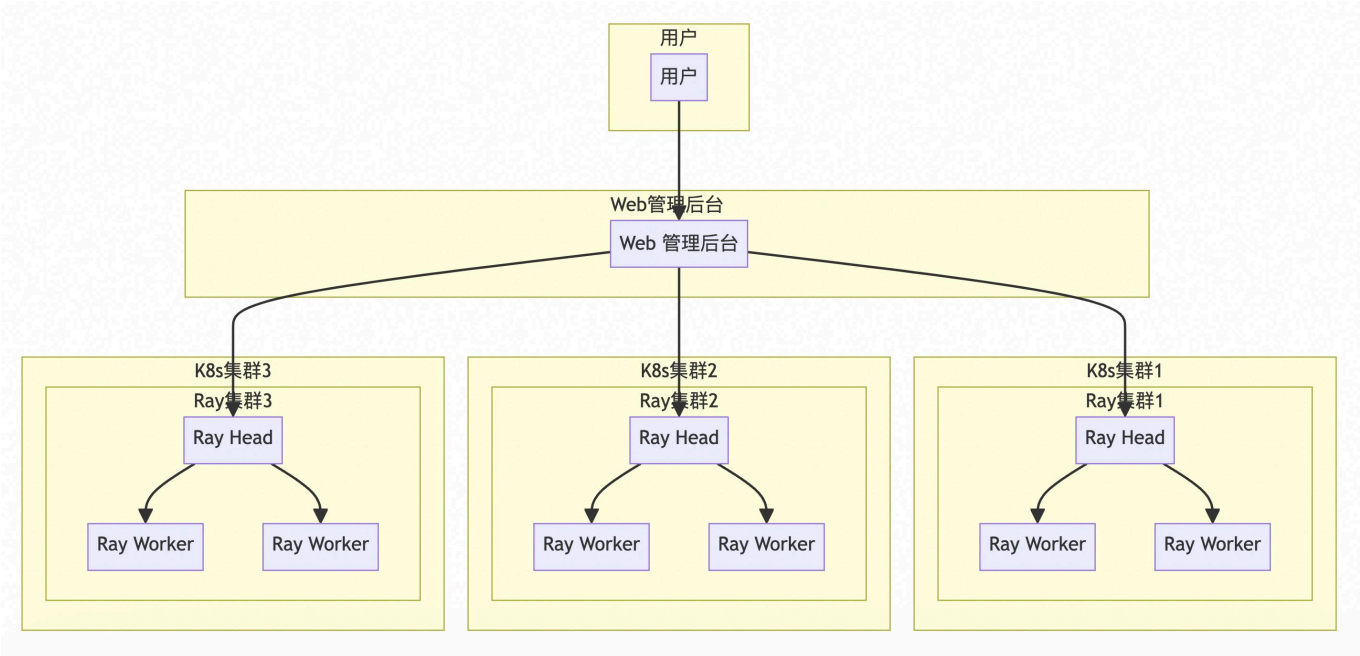
KubeRay 任务管理： 定制的管理控制台 Web 页面提交任务

从 Web 页面提交， 前提条件：Web 页面部署在 Ray/K8S 集群内部，需要代码可以在集群内部访问到



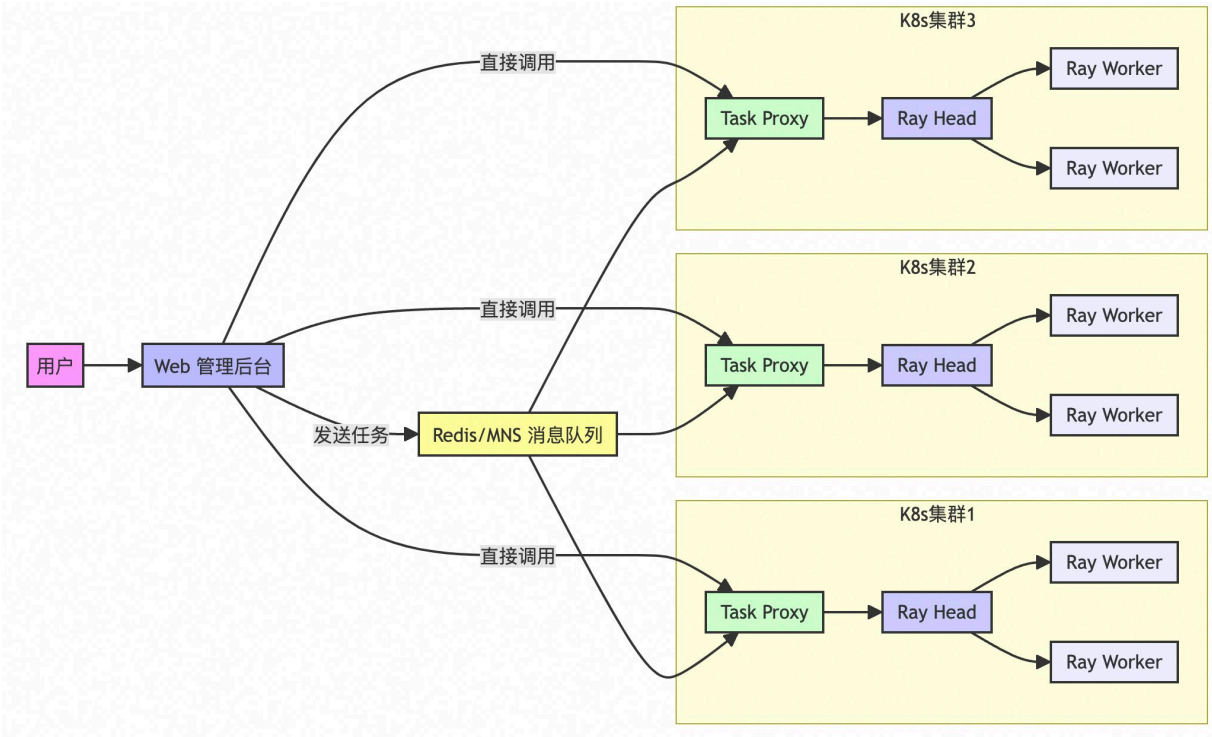
KubeRay 任务管理： 定制的管理控制台 Web 页面提交任务

从 Web 页面提交，同时往多个 k8s/ray 集群提交； 前提条件： Ray/K8S 集群开放管理端口到 Web 管理后台



KubeRay 任务管理： 定制的管理控制台 Web 页面提交任务

从 Web 页面提交，同时往多个 k8s/ray 集群提交； 前提条件： Ray/K8S 集群开放管理端口到 Web 管理后台



如何进行分布式的任务处理：典型的处理架构

RayExample parallel_example.py

Run

```python
import ray

# 初始化 Ray
ray.init()

# 定义一个远程函数
@ray.remote
def square(x):
    return x * x

# 提交多个任务
results = [square.remote(i) for i in range(10)]

# 获取结果
final_results = ray.get(results)
print(final_results)  # 输出 [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

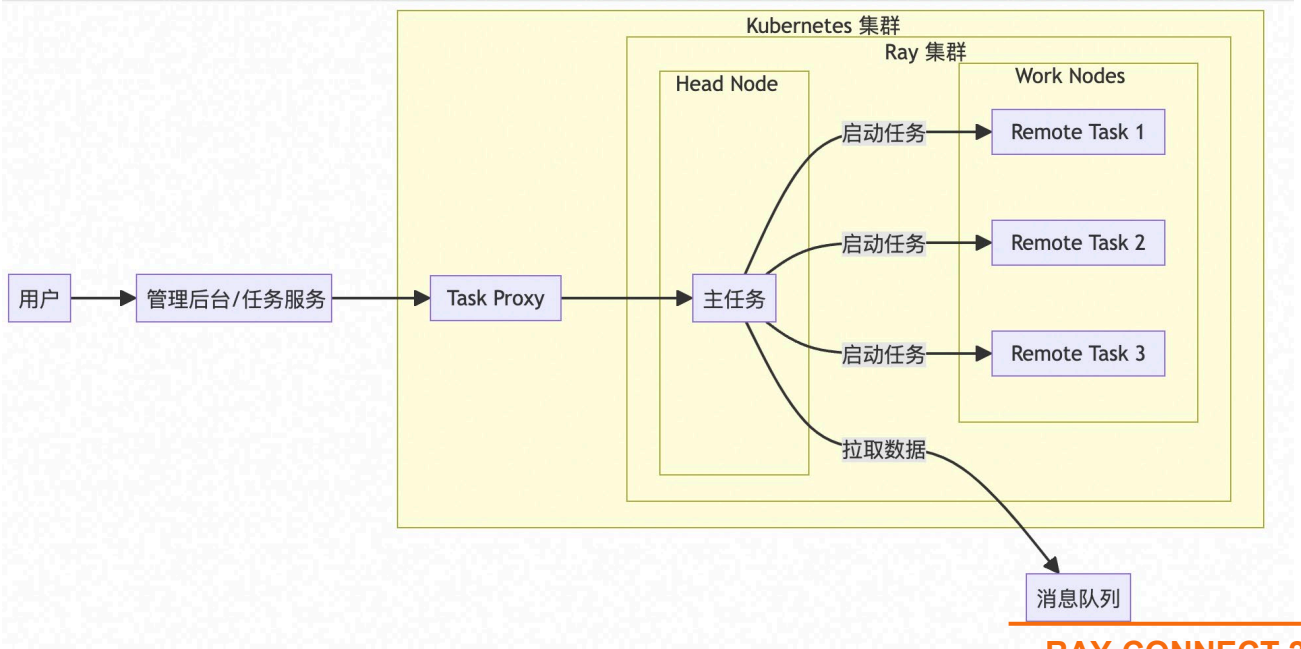
主任务 parallel_example.py 将数据 (i) 发送到各个远程任务

如何进行分布式的任务处理

典型的处理架构

主节点负责分发数据和启动任务，而任务节点负责具体的数据处理。

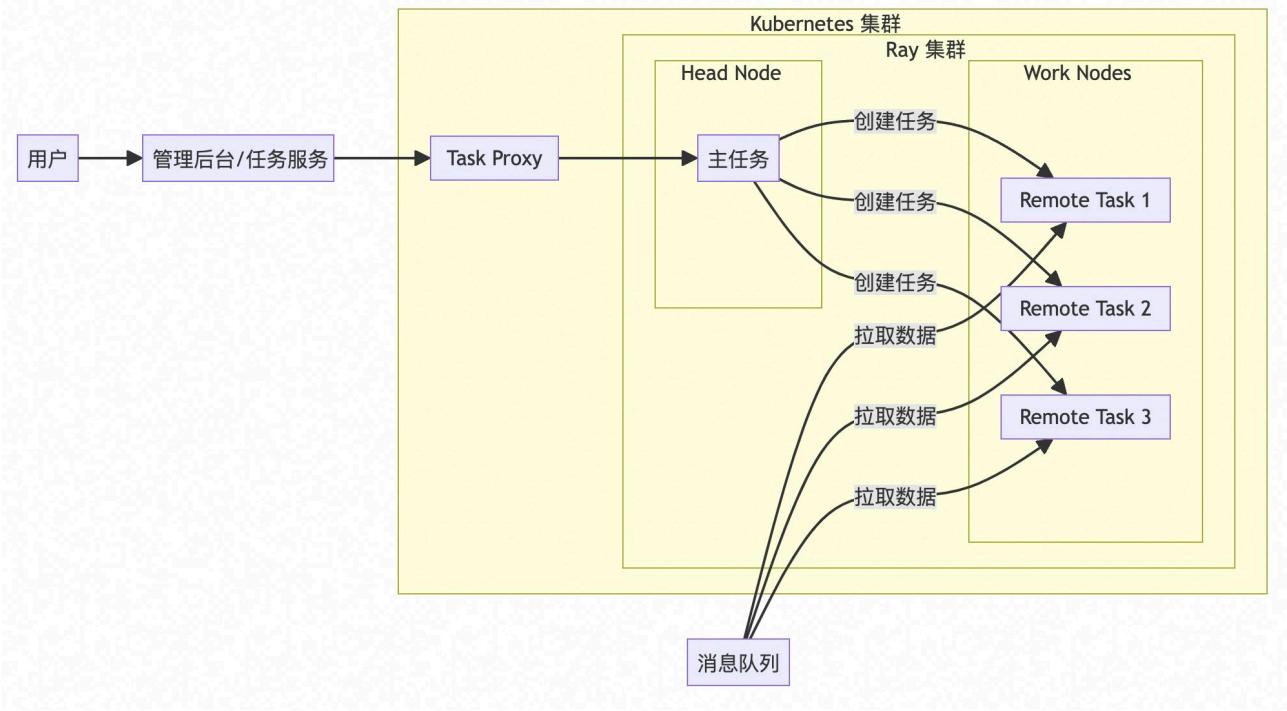
需要监听和管理任务状态，
不同节点的效率不一样，任务的失败重试/节点故障处理；



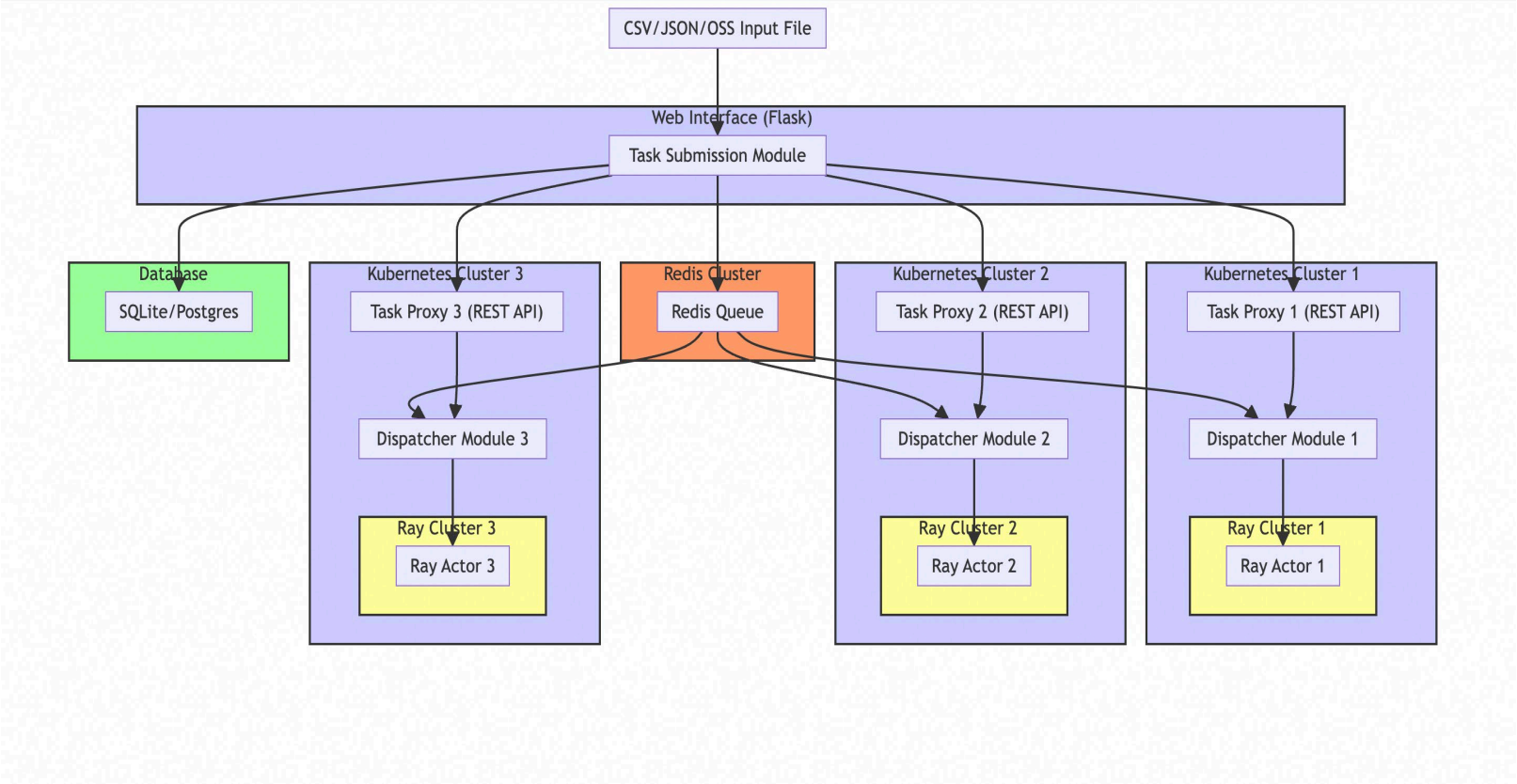
如何进行分布式的任务处理

状态管理与数据分发的优化方案

- 任务节点各自拉取待处理数据
- 只要数据队列中有待处理数据
- 任务节点始终处于工作状态；
- 充分利用多进程/多线程



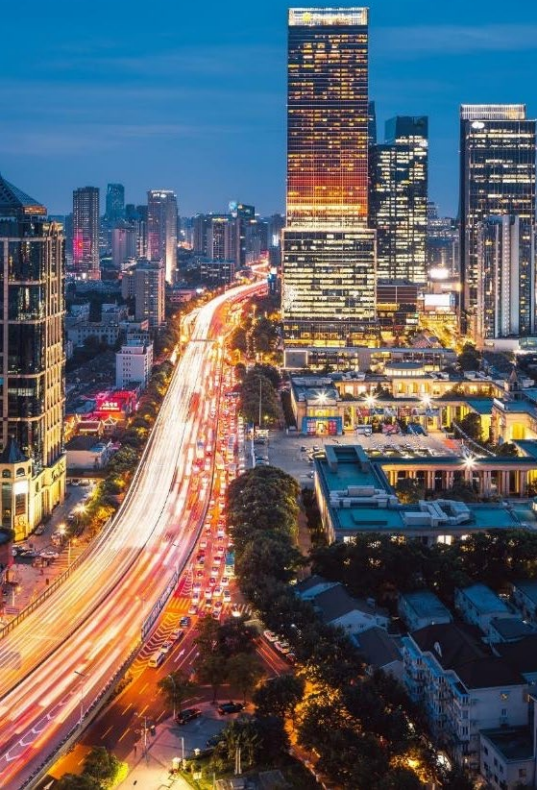
整体架构


CSV/JSON/OSS Input File
Web Interface (Flask)
Task Submission Module
Database
SQLite/Postgres
Kubernetes Cluster 3
Task Proxy 3 (REST API)
Redis Cluster
Redis Queue
Kubernetes Cluster 2
Task Proxy 2 (REST API)
Kubernetes Cluster 1
Task Proxy 1 (REST API)
Dispatcher Module 3
Dispatcher Module 2
Dispatcher Module 1
Ray Cluster 3
Ray Actor 3
Ray Cluster 2
Ray Actor 2
Ray Cluster 1
Ray Actor 1

算法的复用：Ray 集群中算子的抽象

依赖包、依赖环境、输入和输出

多算法处理步骤的任务定义与管理: Data Pipeline 的定义



使用运算符重载， Python 中实现链式的算子调用定义 -> ，
如下代码定义了 5个算法算子依次对输入数据进行处理

```
VideoLlama(oss_input="human-video-high-quality/")
 -> Translation(inherit_input_type="True",file_formats=".json")
 -> ShotStyle()
 -> SubtitleDetection()  ->  LipSyncCheck()
```

任务管理系统的开发和版本发布流程

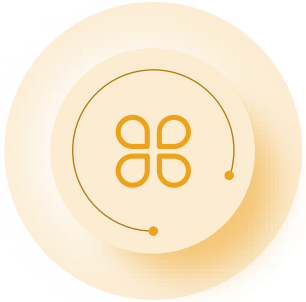
任务管理系统： Web前端管理后台 + 代理服务Task Proxy

使用同样的开发、发布流程

05

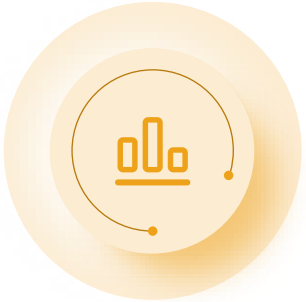
使用 Ray over K8S 的一些经验总结

使用消息队列来解耦



01

使用消息队列可以解耦任务的提交和执行



02

任务数据不放在消息体中传递，提高数据的可扩展性

使用 OSS 云存储来保障数据存储与访问的可扩展性

01

01

OSS 进行数据存储，扩展系统的存储能力，提高数据处理的灵活性

02

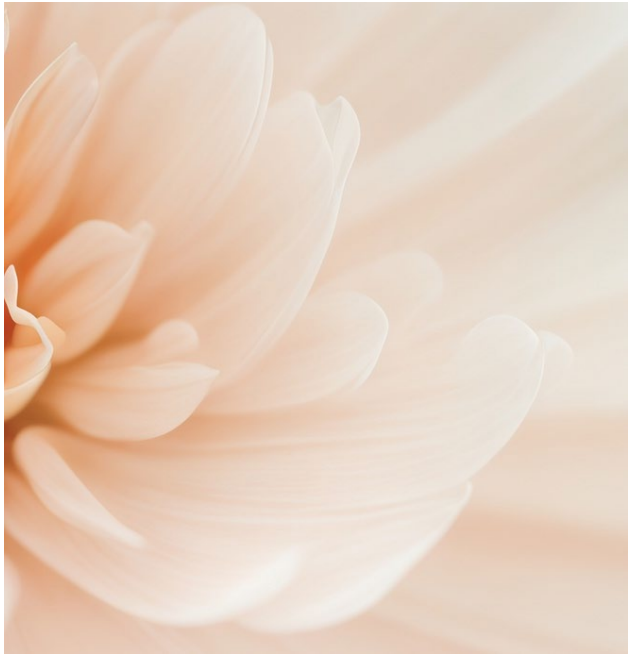
02

> 20 GB/s 的读写性能，几乎可以无限扩展

系统的可恢复性与效率

PART 01

任务设计为可恢复，提高系统的可靠性，减少任务失败带来的影响



PART 02

记录每条数据的状态，可以避免重复处理，提高数据处理的效率

处理效率

01

01

使用批处理来发送数据到消息系统（或数据系统），效率提升 20x

02

02

减少不必要的状态管理、数据传输，子任务独立拉取任务，最大化节点资源利用率（接近100%），最小化出错率

系统的可运维

PART 01

任务从 k8s/ray 内部提交

通过 Web 后台管理和提交任务

避免客户端提交的环境版本一致性问题（Ray，Python）

使用稳定的 Ray 版本

PART 02

结合pip包和work_dir的使用，可以提高系统的复用性和灵活性，方便算法的部署和切换。

镜像打包：
海外镜像打包，阿里云镜像跨区域自动同步

谢谢大家

演讲人：吕召刚

时间：2024.10

2024