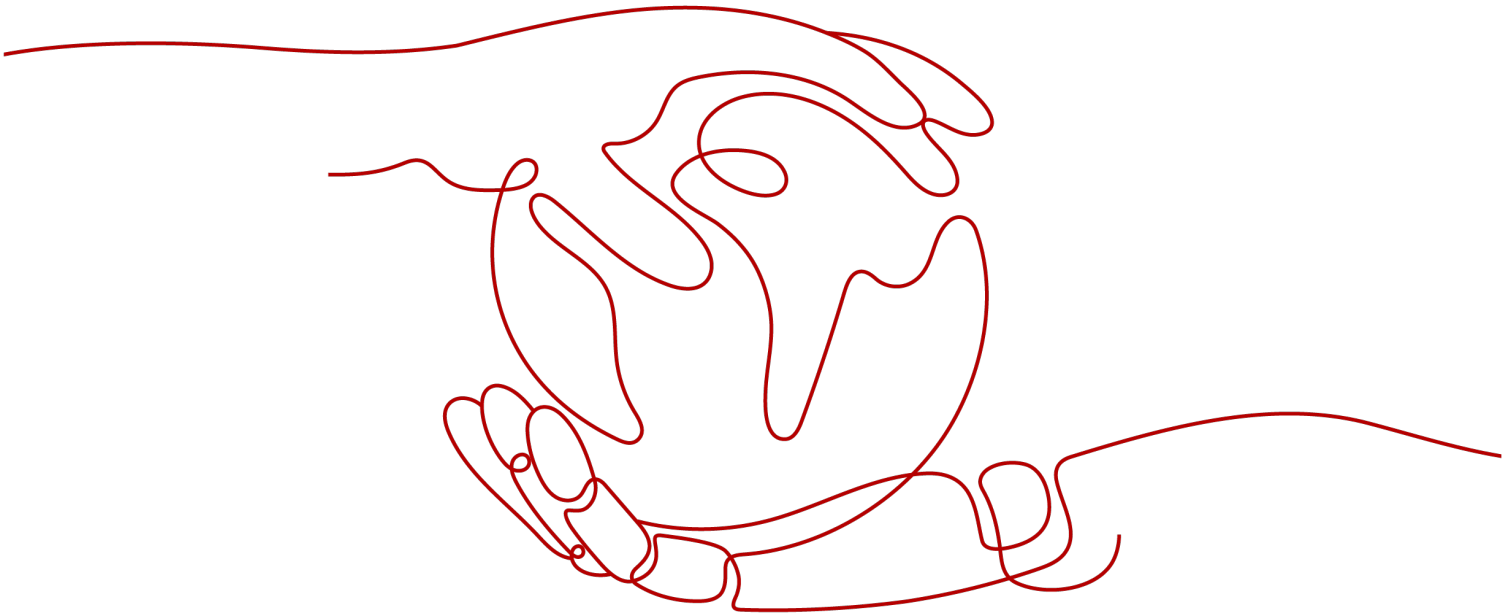
DataArts Fabric

最佳实践

文档版本 01

发布日期 2025-12-17



华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 使用 Ray 进行小模型推理

场景描述

小模型推理通常指的是在相对较小的机器学习模型上进行推理的过程。这些模型可能由于模型复杂度较低、参数量较少等原因，在单个计算节点上就能高效运行。然而，当需要处理的数据量非常大时，即使是小模型也可能面临性能瓶颈。此时，使用Ray进行并行和分布式推理能帮助您提升推理性能。

在DataArts Fabric上使用全托管Ray服务进行小模型推理时，您只需要将您的小模型推理过程定义为Ray可执行的任务，同时在DataArts Fabric中创建推理Job并运行，即可开启推理任务。

前提条件

- 已创建可用的OBS桶。如果未创建，请参考**创建桶**。
- 已创建DataArts Fabric工作空间。如果未创建，请参考**创建工作空间**。
- 已创建DataArts Fabric Ray资源。如果未创建，请参考**购买Ray资源**。
- 已创建DataArts Fabric Ray集群。如果未创建，请参考**创建Ray集群**。

步骤一：准备代码脚本

以下面Python代码脚本为例，创建一个推理作业，做简单的线性回归模型推理，再使用Ray的分布式调度能力计算出推理结果。您可以参照以下示例创建自己的脚本用于后续的推理任务。

- simple_model.py模型定义及启动模型脚本。脚本定义了一个线性回归的模型SimpleModel，并定义了模型部署serve。

```
# simple_model.py
from sklearn.linear_model import LinearRegression
import numpy as np
import pickle
import ray
from ray import serve
from fastapi import FastAPI, Request
from ray.serve.handle import DeploymentHandle

app = FastAPI()

class SimpleModel:
    def __init__(self):
        self.model = LinearRegression()
        X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
```



```
        y = np.dot(X, np.array([1, 2])) + 3
        self.model.fit(X, y)
    def predict(self, X):
        return self.model.predict(X).tolist()

model_instance = SimpleModel()

@serve.deployment(name="simple_model_deployment", ray_actor_options={"num_cpus": 1})
@serve.ingress(app)
class SimpleModelDeployment:
    def __init__(self, model: DeploymentHandle):
        self.model = model
    @app.post("/predict")
    async def predict(self, request: Request):
        request_data = await request.json()
        input_data = np.array(request_data).reshape(-1, 2)
        prediction = self.model.predict(input_data)
        return {"prediction": prediction}
deployment_instance = SimpleModelDeployment.bind(model=model_instance)
serve.run(deployment_instance)
```

- infer_client.py客户端调用主入口脚本。主要流程为调用模型脚本部署模型，然后将数据输入模型进行推理，最后将推理结果上传至OBS。脚本中需要传入必选的-ak、-sk、-ep、-dp参数。
  - -ak是OBS的AK，详情请参见**OBS的AK/SK**。
  - -sk是OBS的SK，详情请参见**OBS的AK/SK**。
  - -ep是OBS Endpoint，详情请参见**OBS Endpoint**。
  - -dp是推理输出文件存储在OBS的路径及文件名。

```
# client.py
import requests
import numpy as np
from obs import ObsClient
from urllib.parse import urlparse
import argparse
from dataclasses import dataclass
import os
from ray import serve
import subprocess
import multiprocessing
import time
import ray
input_file_path = './input.txt'
output_file_path = './output.txt'

def run_model():
    # serve run simple_model:deployment_instance
    subprocess.run(['python3', './simple_model.py'])
@dataclass(frozen=True)
class ParsedObsPath:
    bucket_id: str
    key_id: str
def do_inference():
    input_data = []
    with open(input_file_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            input_data.append([float(parts[0]), float(parts[1])])
    input_data_array = np.array(input_data).tolist()
    print(f"input_data_array={input_data_array}")
    response = requests.post("http://localhost:8000/predict", json=input_data_array)
    print(f"response: {response}")
    predictions = response.json()["prediction"]
    with open(output_file_path, 'w') as f:
        for prediction in predictions:
            f.write(f"{prediction}\n")
    print(f"result save in: {output_file_path}")
```

```
def parse_obs_uri(path: str) -> ParsedObsPath:
    if not path.startswith('obs://'):
        raise Exception(f'OBS path format incorrect: "{path}"')
    parsed = urlparse(path)
    return ParsedObsPath(bucket_id=parsed.netloc, key_id=parsed.path[1:])
def upload_file_to_obs(obs_client: ObsClient, obs_path: str, source_path: str):
    if not os.path.exists(source_path):
        raise Exception(
            f'Source file is not exist: source_path={source_path}')
    uri = parse_obs_uri(obs_path)
    # ObsClient.putFile(bucketName, objectKey, file_path, metadata, headers, progressCallback)
    print(f"bucket_id={uri.bucket_id}, key_id={uri.key_id}, source_path={source_path}")
    result = obs_client.putFile(
        bucketName=uri.bucket_id,
        objectKey=uri.key_id,
        file_path=source_path
    )
    return result
def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("-ak", "--access_key_id", help="OBS access key",
                        type=str, required=True)
    parser.add_argument("-sk", "--secret_access_key", help="OBS secret key",
                        type=str, required=True)
    parser.add_argument("-st", "--security_token", help="OBS security token",
                        type=str, required=False)
    parser.add_argument("-ep", "--endpoint", help="OBS entrypoint", type=str,
                        required=True)
    parser.add_argument("-dp", "--dst_path",
                        help="Local filesystem destination path", type=str,
                        required=True)
    args = parser.parse_args()
    obs_client = ObsClient(
        access_key_id=args.access_key_id,
        secret_access_key=args.secret_access_key,
        security_token=args.security_token,
        server=args.endpoint,
        signature='obs'
    )
    # run model
    print("start run model")
    background_process = multiprocessing.Process(target=run_model)
    background_process.start()
    # wait ray serve ready
    time.sleep(20)
    # do inference
    print("start do inference")
    do_inference()
    # upload infer result to obs
    print("start upload result to obs")
    upload_result = upload_file_to_obs(
        obs_client=obs_client,
        obs_path=args.dst_path,
        source_path=output_file_path
    )
    print(f"upload result={upload_result}")
    if not upload_result.status < 300:
        raise Exception('Error while uploading to OBS.'
                        f' upload status: {upload_result.status}')
    ray.shutdown()
if __name__ == "__main__":
    main()
```

- input.txt推理输入。

```
3.0 5.0
1.0 2.0
4.0 6.0
2.0 3.0
```

步骤二：将代码脚本上传至 OBS 桶

1. 登录华为云控制台，在页面左上角单击☰，选择“存储 > 对象存储服务 OBS”，进入OBS服务。
2. 将**步骤一**创建的代码脚本上传至OBS桶，详情请参见**使用OBS桶上传对象**。
3. 脚本上传成功后，可在OBS桶中查看到已上传的脚本。后续在创建Ray作业时可以选择脚本使用。

   例如，将Job脚本上传至obs://fabric-job-test/rayJob/ray-job/RayInferDemo2目录下。

步骤三：创建一个推理的 Job

1. 登录DataArts Fabric工作空间管理控制台，选择已创建的工作空间，单击“进入工作空间”。
2. 在左侧导航栏，单击“Job定义”，然后在右上角单击“创建作业”。具体操作，请参见**创建Ray Job**。

表 1-1 配置说明

配置项	说明
代码目录	选择**步骤二：将代码脚本上传至OBS桶**中上传的路径，例如obs://fabric-job-test/rayJob/ray-job/RayInferDemo2 。
Ray主文件	选择整个Job的主入口脚本，例如infer_client.py。
Ray作业参数	填写主入口脚本所需参数。ak/sk获取请参考**OBS的AK/SK**，ep获取请参考**OBS Endpoint**。示例如下： -ak XXXXXXXXXXXXXXX -sk xxxxxxxxxxxxxxx -ep obs.cn-north-7.huawei.com -dp obs://fabric-job-test/test_output/output.txt
依赖库	填写依赖及版本。多个依赖需要换行填写。示例如下： scikit-learn==1.5.2 numpy==1.19.5

步骤四：运行 Job

1. Job定义完后，确认Job已选择可用的Ray集群，在“Job定义”的“操作”列，单击目标Job对应的“启动”。
2. 在“Job定义”的“操作”列，单击目标Job对应的“运行详情”，然后单击“运行”页签，查看Job运行状态。

   您可以在运行参数中指定的输出OBS桶路径中查看输出。

2 使用推理服务的权限配置示例

场景描述

假设某个公司需要使用DataArts Fabric服务，有如下需求：

- 张三作为IAM权限管理员，需要在DataArts Fabric页面进行服务授权并为各种不同的角色配置好对应的IAM权限。
- 李四作为开发工程师，需要创建工作空间，搭建一套Ray集群。
- 王五作为算法工程师，使用储存在OBS中的推理文件建立个人推理模型或者使用公共推理模型进行试验。
- 赵六作为测试工程师，需要对于其他同事的工作成果的查询和使用权限。

前提条件

- 张三、李四、王五、赵六已有可正常使用的华为云账号。
- 张三已具有IAM权限管理员身份，且拥有DataArtsFabricFullPolicy权限。

身份与权限说明

表 2-1 身份与权限说明

姓名	身份	权限说明
张三	IAM权限管理员	需要拥有DataArtsFabricFullPolicy权限，并且需要IAM Agency Management FullAccess来创建为DataArts Fabric云服务创建委托。
李四	开发工程师	• 需要拥有DataArtsFabricFullPolicy权限、可以创建工作空间。如果创建工作空间时需要指定LakeFormation Metastore，则还需要LakeFormation ReadOnly Access。 • 搭建Ray集群时需要实际购买Ray资源，DataArtsFabricFullPolicy权限中包含了创建购买Ray资源订单的权限，但是不包含付费的权限，需要由客户指定的费用管理员来进行付费。

姓名	身份	权限说明
王五	算法工程师	需要拥有DataArtsFabricFullPolicy权限、和必须的OBS权限来在DataArts Fabric服务中使用OBS中的模型文件，OBS权限需要由用户权限管理员张三为其授权
赵六	测试工程师	需要拥有DataArtsFabricReadOnlyPolicy进行只读操作。

操作步骤

步骤1 张三进入DataArts Fabric服务的服务授权界面进行授权。具体操作，请参见**创建IAM用户并授权使用DataArts Fabric**。

- 授权李四DataArtsFabricFullPolicy权限和LakeFormation ReadOnly Access权限。
- 授权王五DataArtsFabricFullPolicy权限，并指定OBS桶“my-obs-bucket”的OBS OperateAccess权限。
- 授予赵六DataArtsFabricReadOnlyPolicy权限。

步骤2 李四进入DataArts Fabric页面创建工作空间，并指定MetaStore，具体操作，请参见**创建工作空间**。

步骤3 李四在新创建的工作空间中，购买Ray资源并生成订单。具体操作，请参见**购买Ray资源**。

步骤4 张三进入待支付订单页面，为订单完成付费，付费完成后，Ray集群自动创建。具体操作，请参见**购买Ray资源**。

步骤5 李四在Ray集群上执行Job。具体操作，请参见**创建Ray Job**。

步骤6 王五在模型界面创建模型，并指定模型文件的OBS地址。具体操作，请参见**创建模型**。

步骤7 王五使用该模型构建推理服务，并在试验场完成试验和调试。具体操作，请参见**使用推理服务进行推理**。

步骤8 赵六在Job运行界面检查RayJob的运行结果，在试验场测试王五构建的推理服务。具体操作，请参见**管理Ray Job**和**在试验场进行推理**。

----结束

3 使用 DataArts Fabric SQL 进行数据查询

场景描述

DataArts Fabric全新推出云原生Serverless版本DataArts Fabric SQL，利用云基础设施提供的资源池化和海量存储能力，结合并行执行、元数据解耦、计算持久化分离架构，实现了极致弹性和湖仓一体等特性。

本文介绍如何快速使用DataArts Fabric SQL并完成简单的数据查询。

前提条件

- 已注册账号并实名认证，且账号不能处于欠费或冻结状态。
- 已开通LakeFormation、OBS权限并进行了委托确认。
- 已存在可使用的工作空间。

步骤一：规划并创建 OBS 桶并导入数据

DataArts Fabric SQL通过OBS服务实现数据存储，需要先在OBS控制台进行桶及文件夹创建，并导入样例数据。

1. 登录管理控制台。
2. 在页面左上角单击图标，选择“存储 > 对象存储服务”，进入对象存储服务页面。
3. 以并行文件系统为例：

   选择“并行文件系统 > 创建并行文件系统”，进入创建页面，配置相关参数后单击“立即创建”。

   - 文件系统名称：根据界面要求设置并行文件系统名称，例如“fabric-serverless”。
   - 其他参数根据实际情况选择。
4. 在并行文件系统页面，单击已创建的文件系统名称，例如“fabric-serverless”。
5. 在左侧导航栏选择“文件”，单击“新建文件夹”，填写待创建的文件夹名称，单击“确定”。继续单击该文件夹名称，单击“新建文件夹”，可以创建其子文件夹。
6. 参考该步骤，依次创建用于存放元数据的路径，例如：
   - Catalog存储路径：fabric-serverless/catalog1

- 数据库存储路径：fabric-serverless/catalog1/database1
- 数据表存储路径：fabric-serverless/catalog1/database1/table1

步骤二：规划并创建 Lakeformation 实例、Catalog、数据库

DataArts Fabric SQL通过LakeFormation服务管理数据源，需要在LakeFormation购买实例，并配置该实例的Catalog、数据库、表信息。

1. 登录管理控制台。
2. 在页面左上角单击，选择“大数据 > 湖仓构建 DataArts Lake Formation”，进入LakeFormation页面。
3. 在“总览”购买实例。
4. 左上角选择切换到该实例。
5. 创建Catalog。
   a. 在左侧导航栏选择“元数据 > Catalog”。
   b. 单击“创建Catalog”，配置以下参数后，单击“提交”。
      - Catalog名称：catalog1
      - 选择位置：单击“ ”，选择存储位置，例如选择“obs://fabric-serverless/catalog1”，单击“确定”。
      - Catalog类型：DEFAULT
      - 其他参数保持默认。
   c. 创建完成后，即可在“Catalog”页面查看相关信息。
6. 创建数据库。
   a. 在左侧导航栏选择“元数据 > 数据库”。
   b. 在右上角“Catalog”后的下拉框中选择“catalog1”。
      如果当前已包含名称为“default”的数据库，则跳过数据库的创建操作。
   c. 单击“创建数据库”，配置相关参数后，单击“提交”。
      - 库名称：database1
      - 所属Catalog：catalog1
      - 选择位置：单击“ ”，选择位置，例如选择“obs://fabric-serverless/catalog1/database1”，单击“确定”。
      - 其他参数保持默认。
   d. 创建完成后，即可在“数据库”页面查看详细信息。

步骤三：使用 DataArts Fabric SQL

1. 登录华为云DataArts Fabric控制台，选择进入工作空间。
2. 左侧选择“开发与生产 > SQL编辑器”，选择LakeFormation实例、LakeFormation Catalog。
3. 选择SQL端点，运行SQL。

```
CREATE TABLE
 database1.iceberg_table (
```



```
  col_id INT,
  col_tinyint SMALLINT,
  col_smallint SMALLINT,
  col_int INTEGER,
  col_bigint BIGINT
) store AS iceberg;

SELECT * FROM database1.iceberg_table;
```