【社区】Fluid支持数据处理操作和数据流设计

一、概述

1、需求背景、概述及目标

需求背景

Fluid目前为用户提供了如缓存预热（DataLoad），数据迁移（DataMigrate）等面向运维侧人员的数据操作（Data Operation），在实际的使用场景中，平台运维人员根据自身业务情况，将一系列数据操作包装成任务提交的子步骤，以数据科学家为代表的平台用户在平台上仅需指定所需的数据源（Fluid Dataset中的mountPoint字段），并提交任务就可以通过Fluid访问数据源中的数据。

Fluid目前存在两个待解决的问题：

- **缺少自定义数据处理的数据操作抽象：**这些数据处理的具体逻辑由数据科学家定义。这些操作既可以是数据集中数据的预处理，也可以是类似模型训练任务这种数据消费的任务。Fluid当前无法满足数据科学家对于数据的自定义处理需求，缺少数据处理的抽象。
- **缺少带顺序依赖的数据操作自动化执行能力：**除了对数据处理过程的抽象，Fluid仍然缺少对数据操作流程自动化的功能：目前一个完整数据消费流程中的全部步骤都需要用户手动触发，或者自己编写复杂的脚本逻辑去监听触发，增加了实际使用场景下用户使用Fluid的复杂度。

需求案例

- 【模型训练场景】作为一个数据科学家，我希望能够简单地自动化我在Kubernetes环境中的模型训练过程，包括数据准备（可能包括远程数据源到指定FS的数据迁移、缓存预热步骤）、数据预处理，模型训练的步骤。并且，我可以在例如Jupyter Notebook或VSCode等开发环境中以代码方式定义该自动化流程并提交我的任务。
- 【推理服务场景】作为一个模型推理服务运维人员，我希望能够简单地自动化模型推理服务启动过程，包括原模型的下载、模型切分和转换（e.g. FasterTransformer模型convert, HuggingFace TGI safetensors conversion），多副本模型推理服务拉起等步骤。

目标和非目标

目标：

- 提供一种新的数据操作类型DataProcess，为数据科学家提供自定义数据处理逻辑的抽象。
- 提供DataFlow数据流功能，允许用户通过Fluid提供的API定义自动化的数据处理流程。DataFlow支持Fluid的全部数据操作，包括DataLoad、DataBackup、DataMigrate、DataProcess。

非目标（Non-Goal）：

- DataFlow数据流中的各个子步骤必须是Fluid提供的数据操作CR，**不支持定义不属于Fluid数据操作的任务的前后顺序关系。**
  - 比如，不支持在某个Job任务运行结束后启动DataLoad缓存预热任务，或者是在DataLoad结束后执行PyTorch训练任务这样的数据流定义。如果要实现类似效果，**必须将Job任务或者是PyTorch训练任务包装成DataProcess CR提交。**
- DataFlow数据流仅支持串联多个数据操作，根据定义的先后顺序一个一个执行。**不支持多步骤并行执行、循环执行、按条件选择性执行等高级语义**，如果用户有明确此类需求，推荐使用Argo Workflow或者Tekton。

- **类似作业启动前/结束后动态扩/缩容Dataset数据缓存的需求不在本设计的考虑范围。**
  - 如果需要实现上述效果，可以使用自定义的DataProcess包装 `kubectl scale` 满足需求。

术语声明

- Dataset：Fluid定义的数据集抽象概念，数据集是逻辑上相关的一组数据的集合，基于Dataset Fluid提供数据管理和编排能力
- Runtime：Fluid定义的运行时概念，支持Dataset的多维度能力的执行引擎，例如缓存引擎等。
- Data Operation：Fluid定义的数据操作抽象概念，包括DataLoad、DataMigrate、DataBackup以及本设计涉及的DataProcess都是一种具体的Data Operation。

二、关键设计与实现

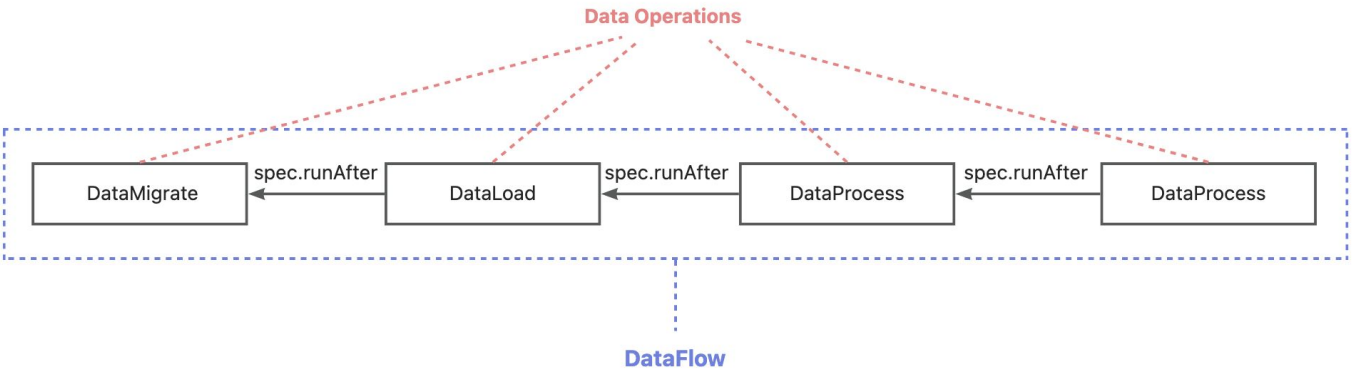
1、设计原则

- **覆盖常见用户场景：**本设计计划覆盖数据科学家对于数据的自定义处理/消费需求，API保持功能精简。
- **向前兼容：**用户可以选择性启用DataFlow功能。如果不启用，各个Data Operation应当保持原逻辑不变。

2、API设计

2.1、设计概述

- 新增的DataProcess CRD和目前已存在的DataLoad CRD，DataMigrate CRD同级，都是一种具体的Data Operation概念。DataProcess CRD的生命周期与DataLoad，DataMigrate一致，都包含None->Pending->Executing->Complete/Failed的生命周期转换流程。
- DataFlow不由CRD实现，而是由各个Data Operation CRD中新增的 `spec.runAfter` 字段实现，每个Data Operation CR的spec.runAfter可以指向另一个Data Operation，由此定义各个Data Operation的执行顺序。



2.2、Data Operation适配DataFlow改动设计

为支持DataFlow能力，所有Fluid Data Operation均需要添加以下两个字段：

- `spec.runAfter`：指定该Data Operation在另一个Data Operation后执行
- `status.waitingFor.operationComplete`：DataFlow Controller控制各个Data Operation的先后执行顺序。用户无需指定，默认值为 `false` 。初始化时，如果 `spec.runAfter` 不为空，对应的DataOperation Controller将 `status.waitingFor.operationComplete` 设置为 `true` 。当runAfter条件满足时，由DataFlow Controller设置为 `false` ，如果DataOperation Controller检查到 `status.waitingFor.operationComplete` 为 `false` ，则实际执行。

Data Operation（DataLoad） 修改后YAML示例：

YAML | 复制代码

```yaml
apiVersion: data.fluid.io/v1alpha1
kind: DataLoad
metadata:
  name: raw-data-warmup
spec:
  runAfter:
    operationKind: DataMigrate
    name: raw-data-migrate
    namespace: default
  ...
status:
  waitingFor:
    operationComplete: true
```

2.2、DataProcess CRD 设计

YAML | 复制代码

```yaml
apiVersion: data.fluid.io/v1alpha1
kind: DataProcess
metadata:
  name: sample-process
spec:
  dataset:
    name: raw-data
    namespace: default
    mountPath: /data
    subPath: path/to/samples
  runAfter:
    operationKind: DataLoad
    name: raw-data-warmup
    namespace: default
  processor:
    # shell script
    shell:
      scirpt: |
        set -ex
        python3 /root/code/process_data.py --data-dir /data
      image: mycode/pypreprocess:latest
      serviceAccountName: mysa
    # kubernetes Job
    job:
      template:
        spec:
          containers:
          - image: mycode/pypreprocess:latest
            imagePullPolicy: IfNotPresent
            name: sample-process
            command:
            - python3
            - /root/code/process_data.py
            - --data-path
            - /data
            volumeMounts:
              - mountPath: /root/code
                name: code
          volumes:
            - name: code
              persistentVolumeClaim:
                claimName: nas-pvc
```

- `dataset`：指定需要处理的目标Dataset。底层实现上将翻译为pod的volumes和

volumeMounts。

- `processor`：包含两种互斥的processor实现：
  - shell: 可使用任意的容器镜像，在容器内执行Shell脚本
  - job：Kubernetes Job，使用Job标准spec处理数据，支持更多扩展场景

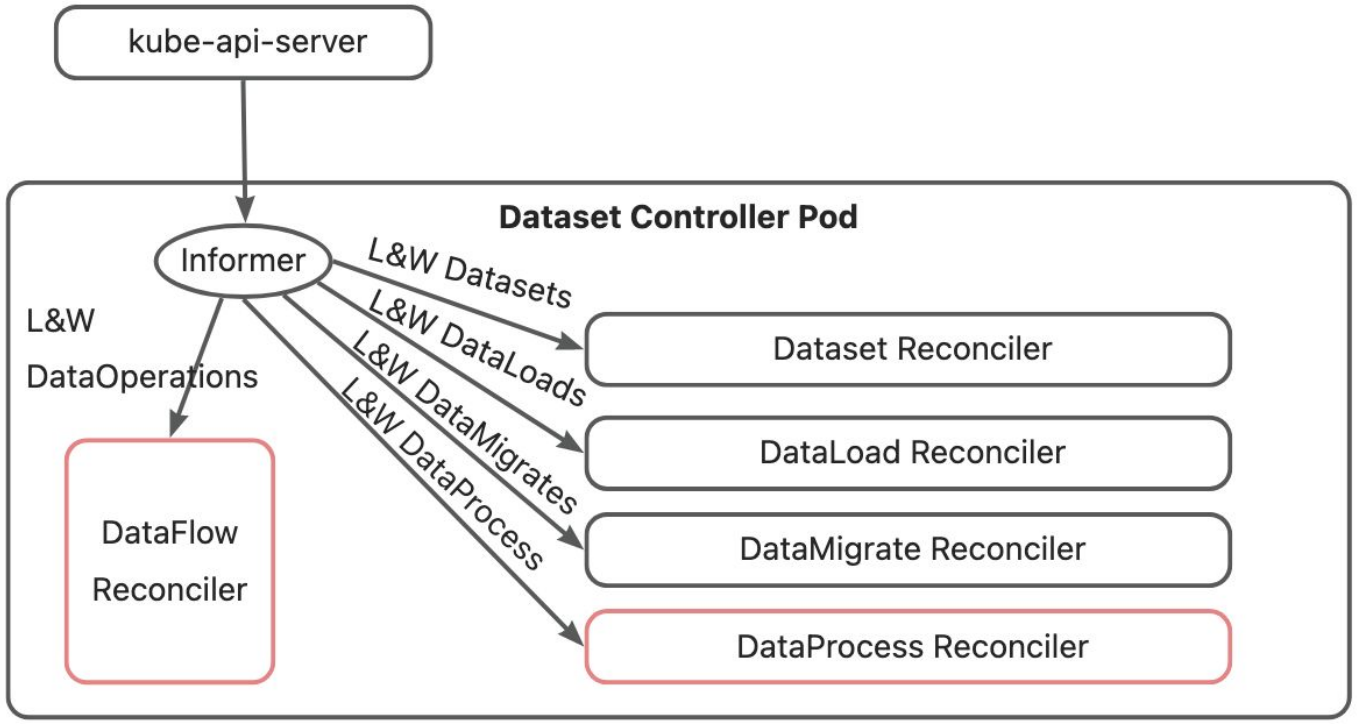
3、架构设计

新增两个组件：

- DataProcess Controller：管理DataProcess CRs的生命周期，提交用户定义的数据处理作业。
- DataFlow Controller：List&Watch所有定义了 `spec.runAfter` 字段的Data Operation CR，控制数据流中各个Data Operation的先后执行顺序。

在架构上，DataProcess Controller和DataFlow Controller在Dataset Controller Pod中以协程方式启动。这种架构的优势在于，DataFlow Controller需要List&Watch全部Data Operation，而Dataset Controller Pod中已经包含了各个Data Operation Controller协程，他们可以共享Informer，减少Fluid控制面整体的内存消耗。

架构示意图如下，红框标出的是本设计新增的组件。



4、组件工作流程

DataProcess工作流程

Fluid Data Operation目前的状态机都按照None –> Pending –> Executing –> Complete/Failed的状态进行转换，DataProcess状态机与现有的其他Data Operation状态机保持一致。

Go | 复制代码

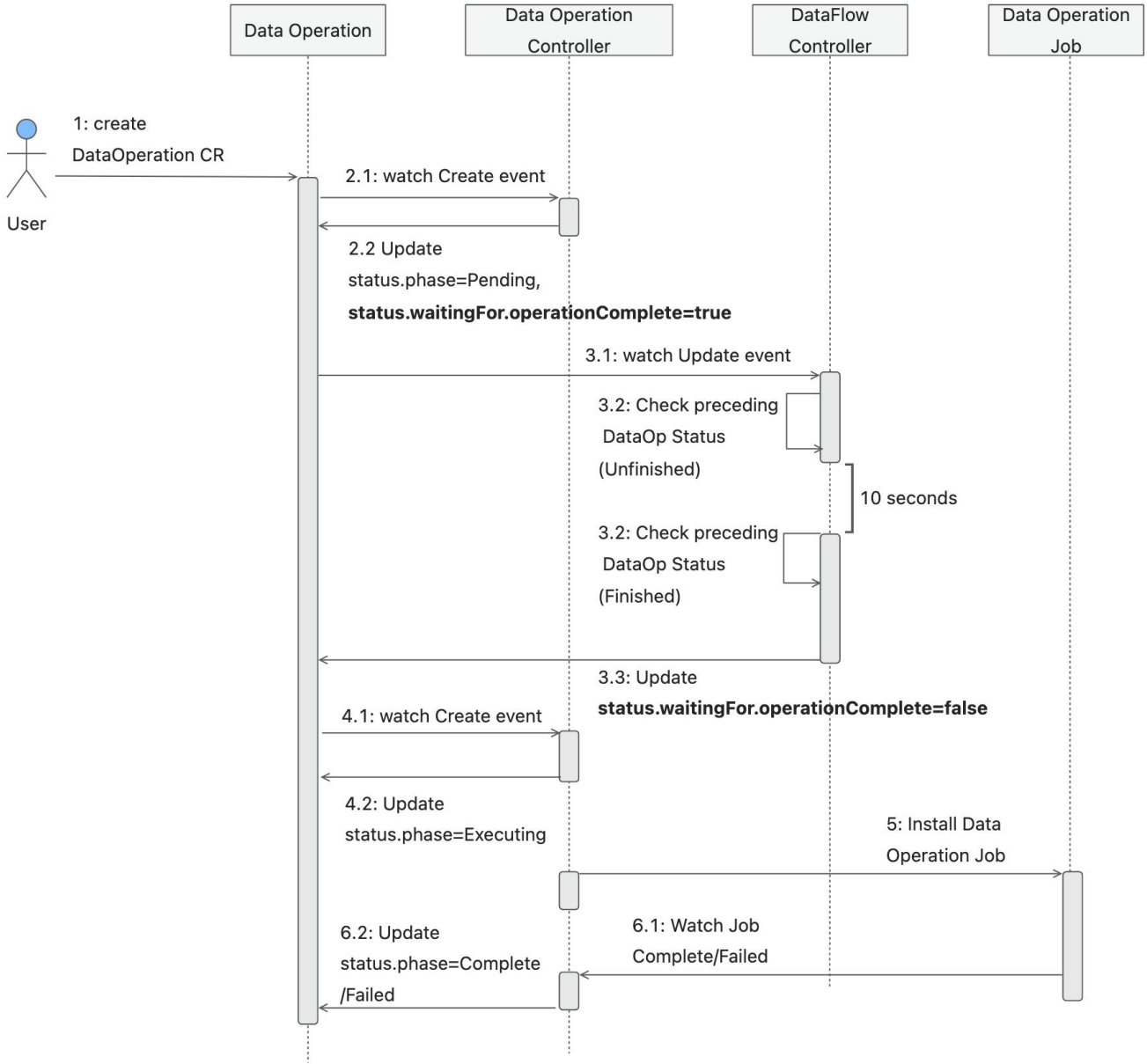
```go
func (t *TemplateEngine) Operate(ctx cruntime.ReconcileRequestContext, object client.Object, opStatus *datav1alpha1.OperationStatus,
    operation dataoperation.OperationInterface) (ctrl.Result, error) {

    // use default template engine
    switch opStatus.Phase {
    case common.PhaseNone:
        return t.reconcileNone(ctx, object, opStatus, operation)
    case common.PhasePending:
        return t.reconcilePending(ctx, object, opStatus, operation)
    case common.PhaseExecuting:
        return t.reconcileExecuting(ctx, object, opStatus, operation)
    case common.PhaseComplete:
        return t.reconcileComplete(ctx, object, opStatus, operation)
    case common.PhaseFailed:
        return t.reconcileFailed(ctx, object, opStatus, operation)
    default:
        ctx.Log.Error(fmt.Errorf("unknown phase"), "won't reconcile it", "phase", opStatus.Phase)
        return utils.NoRequeue()
    }
}
```

在reconcileExecuting逻辑中，根据用户指定的processor类型，DataProcess Controller创建不同的数据处理 Pod：

- 如果用户指定的是**DataProcess.spec.processor.job**，那么按照用户的jobTemplate创建Job任务。
- 如果用户指定的是**DataProcess.spec.processor.shell**，那么使用ConfigMap存储用户的script脚本，并创建Job任务，在Job创建的Pod种执行用户指定的script脚本。

DataFlow工作流程流程



DataOperation Controller工作流程

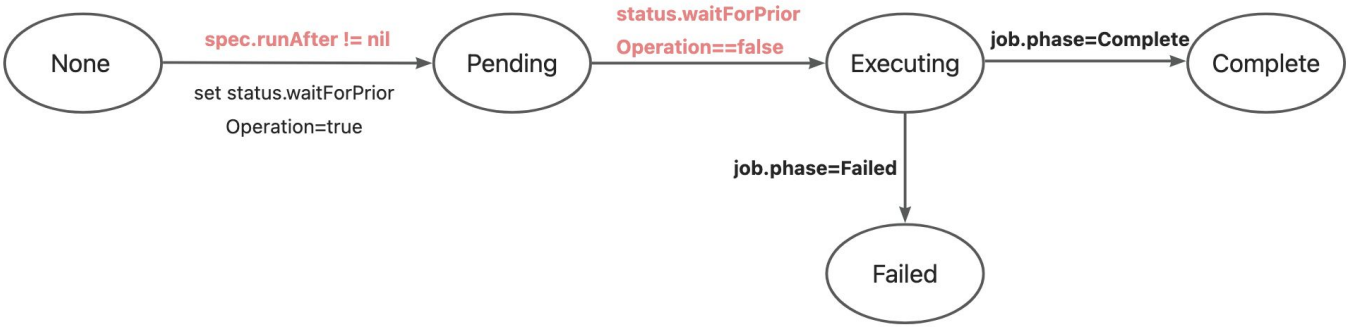
现在Data Operation CR生命周期为：

None –> Pending –> Executing –> Complete/Failed

在None–>Pending的状态转换过程中新增逻辑：

- 如果spec.runAfter为空，不做特殊处理，CR状态设置为Pending状态
- 如果spec.runAfter不为空，设置 `status.waitingFor.operationComplete=true` ，并将CR状态设置为Pending状态

在Pending -> Executing的状态转换中新增判断逻辑，控制Data Operation任务创建，只有在`status.waitingFor.operationComplete==false`时，才将Data Operation CR状态从Pending状态转换至Executing状态。



DataFlow Controller工作流程

1. 用户提交一系列Data Operation CRs，并在各个CR中以`spec.runAfter`字段指定各个Operation之间的前后执行顺序。Data Operation CR中的`status.waitingFor.operationComplete`默认设置为false。
2. DataFlow Controller list&watch所有Data Operation CR，但是仅对**处于Pending状态且`status.waitingFor.operationComplete != nil`的Data Operation CR对象**进行处理。DataFlow Controller按照以下逻辑判断此时该DataOperation是否已满足运行条件，如果已满足运行条件，更新`status.waitForPriorOperation = false`：
    a. 如果`spec.runAfter`为空，更新`status.waitForPriorOperation = false`【针对用户创建某Data Operation CR后，动态修改spec.runAfter的情况】
    b. 如果`spec.runAfter`中指定的对象的`status.phase == Complete`，更新`status.waitForPriorOperation = false`。
    c. 否则，认为不满足条件，10s后重新进入workqueue处理。