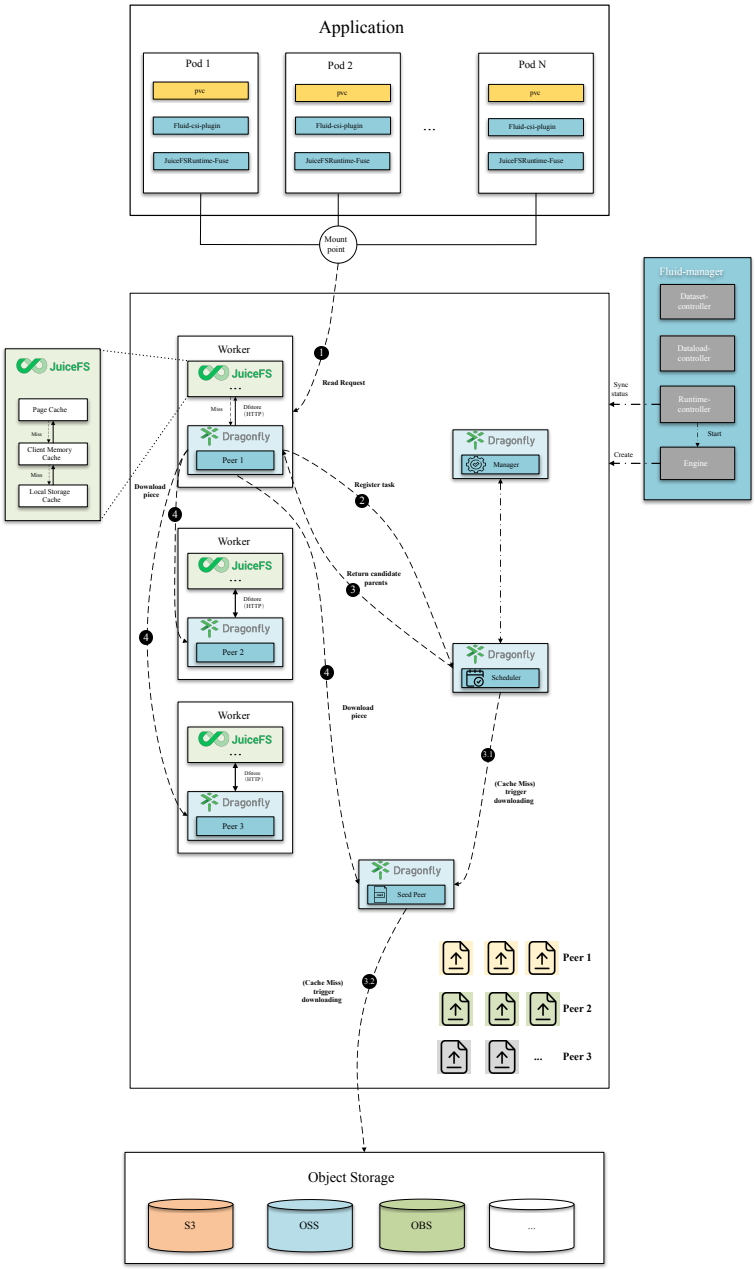
Fluid & JuiceFS & Dargonfly-ZH V2

本文描述了 Fluid & JuiceFS Runtime 与 Dragonfly 集成。介绍了如何搭建 Fluid & JuiceFS Runtime & Dragonfly 的环境，并对集成 Dragonfly 后的加速效果和读性能进行了测试。

Fluid 目前支持多种数据访问引擎，包括 Alluxio Runtime、 JuiceFS Runtime，Thin Runtime 等，使用者无需开发即可实现对通用存储的访问，其中 JuiceFS Runtime 基于 JuiceFS 实现。然而目前 Fluid 针对大文件并发下发的场景仍有优化空间。以 Facebook LLaMa 65B 为例， 其大小为 121.62 GB， 如果采用目前基于内存的缓存或者基于本地磁盘在单节点下的缓存，对于机器的存储和网络来说是很重的负担。而 Dragonfly 是一款基于 P2P 的智能镜像和文件分发工具，在解决大规模文件分发场景下有着无可比拟的优势。**为了解决 Fluid 在分发过程中可能遇到的瓶颈带宽问题，JuiceFS 依赖 Dragonfly 做对象存储，通过 JuiceFS Runtime 实现 Fluid 与 Dragonfly 的集成。**

架构



Dragonfly 作为新的缓存层引入到 JuiceFS 和对象存储之间。在读过程中有一定优化。在读的时候，当 JuiceFS 缓存都没有命中时，就会向 Dragonfly 的 Peer 发起请求通过 P2P 网络分发流量，可以缓解中心化的对象存储带宽压力从而达到加速效果。

JuiceFS 集成 Dragonfly

Dragonfly Kubernetes 集群搭建

准备 Kubernetes 集群

如果没有可用的 Kubernetes 集群进行测试，推荐使用 Kind。

创建 Kind 多节点集群配置文件 `kind-config.yaml`，配置如下：

```shell
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: worker
  - role: worker
```

使用配置文件创建 Kind 集群：

```shell
kind create cluster --config kind-config.yaml
```

Kind 加载 Dragonfly 镜像

下载 Dragonfly Latest 镜像：

```shell
docker pull dragonflyoss/scheduler:latest
docker pull dragonflyoss/manager:latest
docker pull dragonflyoss/dfdaemon:latest
```

Kind 集群加载 Dragonfly Latest 镜像：

```shell
kind load docker-image dragonflyoss/scheduler:latest
kind load docker-image dragonflyoss/manager:latest
kind load docker-image dragonflyoss/dfdaemon:latest
```

基于 Helm Charts 创建 Dragonfly 集群

创建 Helm Charts 配置文件 `charts-config.yaml`，基本的配置如下，完整的配置信息可见此处。设置 `dfdaemon.config.proxy.proxies` 和 `seedPeer.config.proxy.proxies` 参数，需要正则匹配对象存储 Endpoint 地址：

```yaml
scheduler:
  image: dragonflyoss/scheduler
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066

seedPeer:
  image: dragonflyoss/dfdaemon
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066
    proxy:
      defaultFilter: 'Expires&Signature&ns'
      security:
        insecure: true
        tlsVerify: false
      tcpListen:
        # # Listen address.
        # listen: 0.0.0.0
        # Listen port, daemon will try to listen,
        # when this port is not available, daemon will try next port.
        port: 65001
        namespace: ""
      proxies:
        # Proxy all http download requests of the s3.
        - regx: s3.*amazonaws.com.*
        # Proxy all http download requests of the oss.
        - regx: oss.*aliyuncs.com.*
        # Proxy all http download requests of the obs.
        - regx: obs.*myhuaweicloud.com.*

dfdaemon:
  image: dragonflyoss/dfdaemon
  tag: latest
  metrics:
    enable: true
  config:
```

```yaml
    verbose: true
    pprofPort: 18066
    proxy:
      defaultFilter: 'Expires&Signature&ns'
      security:
        insecure: true
        tlsVerify: false
      tcpListen:
        # # Listen address.
        # listen: 0.0.0.0
        # Listen port, daemon will try to listen,
        # when this port is not available, daemon will try next port.
        port: 65001
        namespace: ""
      proxies:
        # Proxy all http download requests of the s3.
        - regx: s3.*amazonaws.com.*
        # Proxy all http download requests of the oss.
        - regx: oss.*aliyuncs.com.*
        # Proxy all http download requests of the obs.
        - regx: obs.*myhuaweicloud.com.*

manager:
  image: dragonflyoss/manager
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066

jaeger:
  enable: true
```

使用配置文件部署 Dragonfly Helm Charts:

```shell
$ helm repo add dragonfly https://dragonflyoss.github.io/helm-charts/
$ helm install --wait --create-namespace --namespace dragonfly-system dragonfly dragonfly/dragonfly -f charts-config.yaml
NAME: dragonfly
LAST DEPLOYED: Thu Sep 28 17:35:49 2023
NAMESPACE: dragonfly-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Get the scheduler address by running these commands:
  export SCHEDULER_POD_NAME=$(kubectl get pods --namespace dragonfly-system -l "app=dragonfly,release=dragonfly,component=scheduler" -o jsonpath={.items[0].metadata.name})
  export SCHEDULER_CONTAINER_PORT=$(kubectl get pod --namespace dragonfly-system $SCHEDULER_POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  kubectl --namespace dragonfly-system port-forward $SCHEDULER_POD_NAME 8002:$SCHEDULER_CONTAINER_PORT
  echo "Visit http://127.0.0.1:8002 to use your scheduler"

2. Get the dfdaemon port by running these commands:
  export DFDAEMON_POD_NAME=$(kubectl get pods --namespace dragonfly-system -l "app=dragonfly,release=dragonfly,component=dfdaemon" -o jsonpath={.items[0].metadata.name})
  export DFDAEMON_CONTAINER_PORT=$(kubectl get pod --namespace dragonfly-system $DFDAEMON_POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  You can use $DFDAEMON_CONTAINER_PORT as a proxy port in Node.

3. Configure runtime to use dragonfly:
  https://d7y.io/docs/getting-started/quick-start/kubernetes/


4. Get Jaeger query URL by running these commands:
  export JAEGER_QUERY_PORT=$(kubectl --namespace dragonfly-system get services dragonfly-jaeger-query -o jsonpath="{.spec.ports[0].port}")
  kubectl --namespace dragonfly-system port-forward service/dragonfly-jaeger-query 16686:$JAEGER_QUERY_PORT
  echo "Visit http://127.0.0.1:16686/search?limit=20&lookback=1h&maxDuration&minDuration&service=dragonfly to query download events"
```

检查 Dragonfly 是否部署成功:

```shell
$ kubectl get po -n dragonfly-system
NAME                                 READY   STATUS    RESTARTS        AGE
dragonfly-dfdaemon-65rz7             1/1     Running   5 (6m17s ago)   8m43s
dragonfly-dfdaemon-rnvsj             1/1     Running   5 (6m23s ago)   8m43s
dragonfly-jaeger-7d58dfcfc8-qmn8c    1/1     Running   0               8m43s
dragonfly-manager-6f8b4f5c66-qq8sd   1/1     Running   0               8m43s
dragonfly-mysql-0                    1/1     Running   0               8m43s
dragonfly-redis-master-0             1/1     Running   0               8m43s
dragonfly-redis-replicas-0           1/1     Running   0               8m43s
dragonfly-redis-replicas-1           1/1     Running   0               7m33s
dragonfly-redis-replicas-2           1/1     Running   0               5m50s
dragonfly-scheduler-0                1/1     Running   0               8m43s
dragonfly-seed-peer-0                1/1     Running   3 (5m56s ago)   8m43s
```

暴露 Dragonfly Peer Proxy 服务端口

创建 `proxy.yaml` 配置文件，暴露 Peer Proxy 服务监听的端口，用于和 JuiceFS 交互。`targetPort` 如果未在 `charts-config.yaml` 中修改默认为 `65001`， port 可根据实际情况设定值，建议也使用 `65001` 。

```yaml
kind: Service
apiVersion: v1
metadata:
  name: proxy
spec:
  selector:
    app: dragonfly
    component: dfdaemon
    release: dragonfly

  ports:
  - protocol: TCP
    port: 65001
    targetPort: 65001

  type: NodePort
```

创建服务:

```
kubectl --namespace dragonfly-system apply -f proxy.yaml
```

将本地的 `65001` 端口流量转发至 Dragonfly 的 Peer Proxy 服务:

```shell
kubectl --namespace dragonfly-system port-forward service/proxy 65001:65001
```

安装 JuiceFS

参考官方文档进行安装。 Linux 和 macOS 系统可使用一键安装脚本，根据硬件架构自动下载安装最新版 JuiceFS 客户端。

```shell
# 默认安装到 /usr/local/bin
curl -sSL https://d.juicefs.com/install | sh -
```

安装完成后，可以在执行 `juicefs format` 和 `juicefs config` 等命令时指定使用的对象存储为 Dragonfly：

```shell
juicefs format \
    --storage dragonfly \
    --access-key ABCDEFGHIJKLMNopqXYZ \
    --secret-key ZYXwvutsrqpoNMLkJiHgfeDCBA \
    --bucket "https://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001&backendStorage=oss" \
    redis://192.168.1.6:6379/1 \
    myjfs
```

`--storage`：设置存储类型，这里为 `dragonfly`。

`--bucket`：设置对象存储的 Endpoint 地址。

`--access-key`：设置对象存储 API 访问密钥 Access Key ID。

`--secret-key`：设置对象存储 API 访问密钥 Access Key Secret。

Dragonfly Peer Proxy 地址参数和 Backend 对象存储类型参数，以 Query String 的形式传入 `--bucket`，参数详细解释如下：

参数	类型	描述	是否必填
backendStorage	string	Backend 对象存储，支持 s3、oss 以及 obs。	Y
proxy	string	Dragonfly Peer 的 Proxy 地址。	Y

验证创建的文件系统状态：

```shell
$ juicefs status redis://192.168.1.6:6379/1
2023/10/17 19:09:35.738635 juicefs[2273224] <INFO>: Meta address: redis://localhost:6379/1 [interface.go:498]
2023/10/17 19:09:35.739344 juicefs[2273224] <WARNING>: AOF is not enabled, you may lose data if Redis is not shutdown properly. [info.go:84]
2023/10/17 19:09:35.739407 juicefs[2273224] <INFO>: Ping redis latency: 22.384µs [redis.go:3572]
{
  "Setting": {
    "Name": "myjfs",
    "UUID": "316d39df-a7ba-4cde-8cc7-5568a7a0f745",
    "Storage": "dragonfly",
    "Bucket": "https://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001\u0026backendStorage=oss",
    "BlockSize": 4096,
    "Compression": "none",
    "EncryptAlgo": "aes256gcm-rsa",
    "TrashDays": 1,
    "MetaVersion": 1,
    "MinClientVersion": "1.1.0-A",
    "DirStats": true
  },
  "Sessions": [],
  "Statistic": {
    "UsedSpace": 0,
    "AvailableSpace": 1125899906842624,
    "UsedInodes": 0,
    "AvailableInodes": 10485760
  }
}

```

在使用其他 JuiceFS 命令的时候也可指定 Dragonfly 为对象存储。 关于 JuiceFS 命令的详细解释可见此处。

功能验证

执行命令：

```shell
juicefs objbench \
    --storage dragonfly \
    --access-key ABCDEFGHIJKLMNopqXYZ \
    --secret-key ZYXwvutsrqpoNMLkJiHgfeDCBA \
    https://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001&backendStorage=oss
```

传入的 Endpoint 形式为 `http://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001&backendStorage=oss`，详情可参照 JuiceFS 环境配置的 `format` 解释。测试通过 `teststorage` 的单元测试:

```shell
+----------+--------------------+-----------------------------------------+
| CATEGORY |         TEST       |                  RESULT                 |
+----------+--------------------+-----------------------------------------+
|    basic |     create a bucket |                                    pass |
|    basic |       put an object |                                    pass |
|    basic |       get an object |                                    pass |
|    basic |       get non-exist |                                    pass |
|    basic |  get partial object | failed to get object with the offset out of r... |
|    basic |      head an object |                                    pass |
|    basic |    delete an object |                                    pass |
|    basic |    delete non-exist |
```

多节点读性能测试

JuiceFS

首先测试 JuiceFS 本身的缓存加速效果，初始化文件系统，为了控制变量，此处配置的对象存储需要与 Dragonfly 中一致。

```shell
juicefs format \
    --storage oss \
    --bucket https://myjfs.oss-cn-shanghai.aliyuncs.com \
    --access-key ABCDEFGHIJKLMNopqXYZ \
    --secret-key ZYXwvutsrqpoNMLkJiHgfeDCBA \
    redis://192.168.1.6:6379/2 \
    myjfs
```

将格式化的文件系统挂载:

```shell
juicefs mount redis://192.168.1.6:6379/2  /mnt/jfs
```

首先在挂载的目录新建大小为 1G 的文件:

```shell
$ time dd if=/dev/zero of=/mnt/jfs/test.txt bs=1M count=1000
1000+0 records in
1000+0 records out
1048576000 bytes (1.0 GB, 1000 MiB) copied, 10.7013 s, 98.0 MB/s
dd if=/dev/zero of=/mnt/jfs/test.txt bs=1M count=1000  0.00s user 0.33s sys
tem 3% cpu 10.711 tota
```

进行第一次读， JuiceFS 触发**回源下载**，耗费 **11.356 秒**。

```shell
$ time cp /mnt/jfs/test.txt /dev/null

cp /mnt/jfs/test.txt /dev/null  0.00s user 0.29s system 2% cpu 11.356 total
```

清除页缓存，重新读命中 JuiceFS 的**本地缓存**，耗费 **0.347 秒**。

```shell
$ sync && echo 3 > /proc/sys/vm/drop_caches
$ time cp /mnt/jfs/test.txt /dev/null
cp /mnt/jfs/test.txt /dev/null  0.00s user 0.30s system 86% cpu 0.347 total
```

JuiceFS & Dragonfly

接下来测试 Dragonfly 的加速效果。分别测试命中 Local Peer 和 Remote Peer 缓存，首先暴露 Dragonfly Peer 的 65001 端口：

```shell
export dragonfly_dfdaemon_name=$(kubectl get po -n dragonfly-system | grep dragonfly-dfdaemon- | tail -n 1 | awk '{print $1}')
kubectl --namespace dragonfly-system port-forward $dragonfly_dfdaemon_name 65001:65001
```

初始化基于 Dragonfly 的 JuiceFS 文件系统：

```shell
juicefs format \
    --storage dragonfly \
    --access-key ABCDEFGHIJKLMNopqXYZ \
    --secret-key ZYXwvutsrqpoNMLkJiHgfeDCBA \
    --bucket "https://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001&backendStorage=oss" \
    redis://192.168.1.6:6379/1 \
    myjfs
```

将文件系统挂载，并禁用 JuiceFS 的本地缓存:

```shell
juicefs mount redis://192.168.1.6:6379/1  /mnt/jfs --cache-size=0
```

挂载的目录新建一个大小 1G 的文件：

```shell
$ time dd if=/dev/zero of=/mnt/jfs/test.txt bs=1M count=1000
1000+0 records in
1000+0 records out
1048576000 bytes (1.0 GB, 1000 MiB) copied, 10.2689 s, 102 MB/s
dd if=/dev/zero of=/mnt/jfs/test.txt bs=1M count=1000  0.00s user 0.38s system 3% cpu 10.271 total
```

然后进行第一次读，此时 JuiceFS 和 Dragonfly 都没有缓存，会触发**回源下载**，耗费 **11.147 秒**。

```shell
$ time cp /mnt/jfs/test.txt /dev/null
cp /mnt/jfs/test.txt /dev/null  0.00s user 0.30s system 2% cpu 11.147 total
```

清除文件系统的缓存。重新读命中 Dragonfly Local Peer 的缓存，耗费 **1.554 秒**。

```shell
$ sync && echo 3 > /proc/sys/vm/drop_caches
$ time cp /mnt/jfs/test.txt /dev/null
cp /mnt/jfs/test.txt /dev/null  0.00s user 0.32s system 20% cpu 1.554 total
```

为了测试命中 Dragonfly Remote Peer 的缓存速度，删除 Peer:

```shell
$ export dragonfly_dfdaemon_name=$(kubectl get po -n dragonfly-system --sort-by=.metadata.creationTimestamp | grep "dragonfly-dfdaemon-" | awk '{print $1}' | tail -n 1)
$ kubectl delete po $dragonfly_dfdaemon_name -n dragonfly-system
$ kubectl get po -n dragonfly-system
NAME                                 READY   STATUS    RESTARTS   AGE
dragonfly-dfdaemon-5q4r8             1/1     Running   0          30s # new pod
dragonfly-dfdaemon-nhzcc             1/1     Running   0          19m
dragonfly-jaeger-c7947b579-q4hr4     1/1     Running   0          19m
dragonfly-manager-5dc5fbf548-zrf7d   1/1     Running   0          19m
dragonfly-mysql-0                    1/1     Running   0          19m
dragonfly-redis-master-0             1/1     Running   0          19m
dragonfly-redis-replicas-0           1/1     Running   0          19m
dragonfly-redis-replicas-1           1/1     Running   0          18m
dragonfly-redis-replicas-2           1/1     Running   0          18m
dragonfly-scheduler-0                1/1     Running   0          19m
dragonfly-seed-peer-0                1/1     Running   0          19m
```

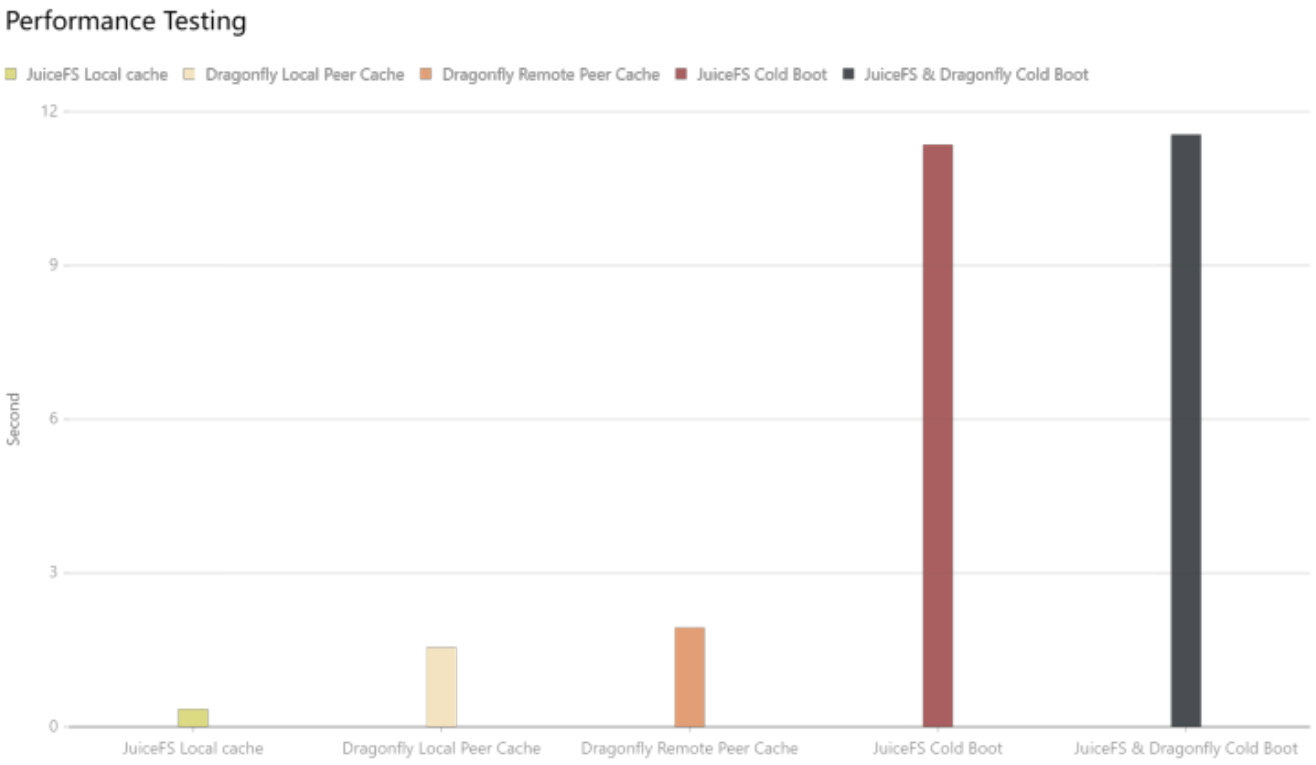
可见 dragonfly-dfdaemon-5q4r8 是重建的 Pod:

```shell
kubectl --namespace dragonfly-system port-forward $dragonfly_dfdaemon_name 65001:65001
```

清除文件系统缓存。重新读新建立的 Pod 没有缓存， 会命中 **Remote Peer** 的缓存，耗费 **1.937 秒**。

```shell
$ sync && echo 3 > /proc/sys/vm/drop_caches
$ time cp /mnt/jfs/test.txt /dev/null
cp /mnt/jfs/test.txt /dev/null  0.01s user 0.32s system 16% cpu 1.937 total
```

结果分析



可见整体读取速度从快到慢为 JuiceFS 本地缓存, Dragonfly Local Peer, Dragonfly Remote Peer , Dragonfly 回源与 JuiceFS 回源。由于机器本身网络环境、配置等影响，实际下载时间不具有参考价值， 但是不同场景下载时间所提升的比率是有重要意义的。

Fluid & JuiceFS Runtime 集成 Dragonfly

Dragonfly Kubernetes 集群搭建

准备 Kubernetes 集群

如果没有可用的 Kubernetes 集群进行测试，推荐使用 Kind。

创建 Kind 多节点集群配置文件 kind-config.yaml, 配置如下:

```shell
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: worker
  - role: worker
```

使用配置文件创建 Kind 集群:

```shell
kind create cluster --config kind-config.yaml
```

Kind 加载 Dragonfly 镜像

下载 Dragonfly Latest 镜像:

```shell
docker pull dragonflyoss/scheduler:latest
docker pull dragonflyoss/manager:latest
docker pull dragonflyoss/dfdaemon:latest
```

Kind 集群加载 Dragonfly Latest 镜像:

```shell
kind load docker-image dragonflyoss/scheduler:latest
kind load docker-image dragonflyoss/manager:latest
kind load docker-image dragonflyoss/dfdaemon:latest
```

基于 Helm Charts 创建 Dragonfly 集群

创建 Helm Charts 配置文件 `charts-config.yaml`, 基本的配置如下，完整的配置信息可见此处。设置对象存储需要修改 `dfdaemon.config.proxy.proxies` 和 `seedPeer.config.proxy.proxies` 参数，需要正则匹配对象存储 Endpoint 地址：

YAML

```yaml
scheduler:
  image: dragonflyoss/scheduler
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066

seedPeer:
  image: dragonflyoss/dfdaemon
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066
    proxy:
      defaultFilter: 'Expires&Signature&ns'
      security:
        insecure: true
        tlsVerify: false
      tcpListen:
        # # Listen address.
        # listen: 0.0.0.0
        # Listen port, daemon will try to listen,
        # when this port is not available, daemon will try next port.
        port: 65001
        namespace: ""
      proxies:
        # Proxy all http download requests of the s3.
        - regx: s3.*amazonaws.com.*
        # Proxy all http download requests of the oss.
        - regx: oss.*aliyuncs.com.*
        # Proxy all http download requests of the obs.
        - regx: obs.*myhuaweicloud.com.*

dfdaemon:
  image: dragonflyoss/dfdaemon
  tag: latest
  metrics:
    enable: true
  config:
```

```yaml
    verbose: true
    pprofPort: 18066
    proxy:
      defaultFilter: 'Expires&Signature&ns'
      security:
        insecure: true
        tlsVerify: false
      tcpListen:
        # # Listen address.
        # listen: 0.0.0.0
        # Listen port, daemon will try to listen,
        # when this port is not available, daemon will try next port.
        port: 65001
        namespace: ""
      proxies:
        # Proxy all http download requests of the s3.
        - regx: s3.*amazonaws.com.*
        # Proxy all http download requests of the oss.
        - regx: oss.*aliyuncs.com.*
        # Proxy all http download requests of the obs.
        - regx: obs.*myhuaweicloud.com.*

manager:
  image: dragonflyoss/manager
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066

jaeger:
  enable: true
```

使用配置文件部署 Dragonfly Helm Charts:

```shell
$ helm repo add dragonfly https://dragonflyoss.github.io/helm-charts/
$ helm install --wait --create-namespace --namespace dragonfly-system dragonfly dragonfly/dragonfly -f charts-config.yaml
NAME: dragonfly
LAST DEPLOYED: Thu Sep 28 17:35:49 2023
NAMESPACE: dragonfly-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Get the scheduler address by running these commands:
  export SCHEDULER_POD_NAME=$(kubectl get pods --namespace dragonfly-system -l "app=dragonfly,release=dragonfly,component=scheduler" -o jsonpath={.items[0].metadata.name})
  export SCHEDULER_CONTAINER_PORT=$(kubectl get pod --namespace dragonfly-system $SCHEDULER_POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  kubectl --namespace dragonfly-system port-forward $SCHEDULER_POD_NAME 8002:$SCHEDULER_CONTAINER_PORT
  echo "Visit http://127.0.0.1:8002 to use your scheduler"

2. Get the dfdaemon port by running these commands:
  export DFDAEMON_POD_NAME=$(kubectl get pods --namespace dragonfly-system -l "app=dragonfly,release=dragonfly,component=dfdaemon" -o jsonpath={.items[0].metadata.name})
  export DFDAEMON_CONTAINER_PORT=$(kubectl get pod --namespace dragonfly-system $DFDAEMON_POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  You can use $DFDAEMON_CONTAINER_PORT as a proxy port in Node.

3. Configure runtime to use dragonfly:
  https://d7y.io/docs/getting-started/quick-start/kubernetes/


4. Get Jaeger query URL by running these commands:
  export JAEGER_QUERY_PORT=$(kubectl --namespace dragonfly-system get services dragonfly-jaeger-query -o jsonpath="{.spec.ports[0].port}")
  kubectl --namespace dragonfly-system port-forward service/dragonfly-jaeger-query 16686:$JAEGER_QUERY_PORT
  echo "Visit http://127.0.0.1:16686/search?limit=20&lookback=1h&maxDuration&minDuration&service=dragonfly to query download events"
```

检查 Dragonfly 是否部署成功:

```shell
$ kubectl get po -n dragonfly-system
NAME                                    READY   STATUS    RESTARTS        AGE
dragonfly-dfdaemon-65rz7                1/1     Running   5 (6m17s ago)   8m43s
dragonfly-dfdaemon-rnvsj                1/1     Running   5 (6m23s ago)   8m43s
dragonfly-jaeger-7d58dfcfc8-qmn8c       1/1     Running   0               8m43s
dragonfly-manager-6f8b4f5c66-qq8sd      1/1     Running   0               8m43s
dragonfly-mysql-0                       1/1     Running   0               8m43s
dragonfly-redis-master-0                1/1     Running   0               8m43s
dragonfly-redis-replicas-0              1/1     Running   0               8m43s
dragonfly-redis-replicas-1              1/1     Running   0               7m33s
dragonfly-redis-replicas-2              1/1     Running   0               5m50s
dragonfly-scheduler-0                   1/1     Running   0               8m43s
dragonfly-seed-peer-0                   1/1     Running   3 (5m56s ago)   8m43s
```

暴露 Dragonfly Peer Proxy 服务端口

创建 `proxy.yaml` 配置文件，暴露 Peer Proxy 服务监听的端口，用于和 JuiceFS 交互。`targetPort` 如果未在 `charts-config.yaml` 中修改默认为 `65001`， port 可根据实际情况设定值，建议也使用 `65001` 。

```yaml
kind: Service
apiVersion: v1
metadata:
  name: proxy
spec:
  selector:
    app: dragonfly
    component: dfdaemon
    release: dragonfly

  ports:
  - protocol: TCP
    port: 65001
    targetPort: 65001

  type: NodePort
```

创建服务:

```
kubectl --namespace dragonfly-system apply -f proxy.yaml
```

将本地的 `65001` 端口流量转发至 Dragonfly Peer Proxy 服务:

```shell
kubectl --namespace dragonfly-system port-forward service/proxy 65001:65001
```

安装 Fluid

详细流程可参照文档，此处对 Dragonfly 相关参数进行了说明。

基于 Helm Chart 创建 Fluid 集群

可以参考安装文档完成 Fluid 的安装。

创建命名空间:

```shell
$ kubectl create ns fluid-system
```

部署 Fluid Helm Charts:

```shell
$ helm repo add fluid https://fluid-cloudnative.github.io/charts
$ helm repo update
$ helm install fluid fluid/fluid
NAME: fluid
LAST DEPLOYED: Thu Oct 12 21:54:34 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

验证是否安装成功：

```shell
$ kubectl get po -n fluid-system
NAME                                  READY   STATUS    RESTARTS   AGE
csi-nodeplugin-fluid-nq65p            2/2     Running   0          7m12s
csi-nodeplugin-fluid-nrwbt            2/2     Running   0          7m12s
csi-nodeplugin-fluid-q565r            2/2     Running   0          7m12s
dataset-controller-5f5f46d969-lpsc7   1/1     Running   0          7m11s
fluid-webhook-75f489c7b5-whzjz        1/1     Running   0          7m12s
fluidapp-controller-54975849ff-w272h  1/1     Running   0          7m12s
```

创建 Dataset 资源对象

提供元数据服务，并创建对应的 secret.yaml：

```shell
apiVersion: v1
kind: Secret
metadata:
  name: jfs-secret
type: Opaque
stringData:
  name: dragonfly
  metaurl: redis://127.0.0.1:6379/3
  token: ${JUICEFS_TOKEN}
  access-key: ${ACCESS_KEY}
  secret-key: ${SECRET_KEY}
```

使用上述配置文件创建 Secret：

```shell
$ kubectl create -f secret.yaml
```

创建 dataset.yaml：

```yaml
$ cat<<EOF >dataset.yaml
apiVersion: data.fluid.io/v1alpha1
kind: Dataset
metadata:
  name: jfsdemo
spec:
  mounts:
    - name: dragonfly
      mountPoint: "juicefs:///"
      options:
        bucket: "'https://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001&backendStorage=oss'"
        storage: "dragonfly"
      encryptOptions:
        - name: metaurl
          valueFrom:
            secretKeyRef:
              name: jfs-secret
              key: metaurl
        - name: token
          valueFrom:
            secretKeyRef:
              name: jfs-secret
              key: token
        - name: access-key
          valueFrom:
            secretKeyRef:
              name: jfs-secret
              key: access-key
        - name: secret-key
          valueFrom:
            secretKeyRef:
              name: jfs-secret
              key: secret-key
EOF
```

- mountPoint：用户在 JuiceFS 文件系统中存储数据的目录，以 `juicefs://` 开头。如 `juicefs:///demo` 为 JuiceFS 文件系统的 `/demo` 子目录。
- storage：对象存储类型，此处指定为 dragonfly。
- bucket: 设置对象存储的 Endpoint 地址。

Dragonfly Peer Proxy 地址参数和 Backend 对象存储类型参数，以 Query String 的形式传入 `--bucket`，参数详细解释如下：

参数	类型	描述	是否必填
backendStorage	string	Backend 对象存储，支持 s3、oss 以及 obs。	Y
proxy	string	Dragonfly Peer 的 Proxy 地址。	Y

创建 Dataset 资源对象:

```shell
$ kubectl create -f dataset.yaml
dataset.data.fluid.io/jfsdemo created
```

创建 JuiceFS Runtime 资源对象

创建 JuiceFS Runtime 的配置文件 runtime.yaml，其中 metadata 与 dataset 同名：

```shell
$ cat<<EOF >runtime.yaml
apiVersion: data.fluid.io/v1alpha1
kind: JuiceFSRuntime
metadata:
  name: jfsdemo
spec:
  replicas: 1
  fuse:
    image: dragonflyoss/juicefs-fuse
    imageTag: 0.1.0
    imagePullPolicy: IfNotPresent
  juicefsVersion:
    image: dragonflyoss/juicefs-fuse
    imageTag: 0.1.0
    imagePullPolicy: IfNotPresent
  tieredstore:
    levels:
      - mediumtype: MEM
        path: /dev/shm
        quota: 40Gi
        low: "0.1"
EOF
```

使用上述配置文件创建 JuiceFS Runtime：

```shell
$ kubectl create -f runtime.yaml
juicefsruntime.data.fluid.io/jfsdemo created
```

等待一段时间，让 JuiceFS Runtime 资源对象中的各个组件得以顺利启动：

```shell
$ kubectl get po |grep jfs
jfsdemo-worker-0                                                  1/1     Running
 0          4m2s
```

查看 dataset 状态，发现已经与 JuiceFS Runtime 绑定。

```shell
$ kubectl get dataset jfsdemo
NAME      UFS TOTAL SIZE   CACHED   CACHE CAPACITY   CACHED PERCENTAGE   PHASE   AGE
jfsdemo   0.00B            0.00B    40.00GiB         0.0%                Bound   21h
```

此时 Fluid 创建了默认的 PV 和与 dataset 同名的 PVC：

```shell
$ kubectl get pv | grep jfs
default-jfsdemo       100Pi      ROX            Retain           Bound    default/jfsdemo                                  fluid                   16h
$ kubectl get pvc
NAME      STATUS   VOLUME            CAPACITY   ACCESS MODES   STORAGECLASS   AGE
jfsdemo   Bound    default-jfsdemo   100Pi      ROX            fluid          21h
```

功能验证

创建一个应用容器来使用数据集:

```shell
$ cat<<EOF >app.yaml
apiVersion: v1
kind: Pod
metadata:
  name: demo-app
spec:
  containers:
    - name: demo
      image: nginx
      volumeMounts:
        - mountPath: /demo
          name: demo
  volumes:
    - name: demo
      persistentVolumeClaim:
        claimName: jfsdemo
EOF
```

创建应用:

```shell
$ kubectl create -f app.yaml
```

检查 Pod 资源对象是否已经创建:

```shell
$ kubectl get pod
NAME                READY   STATUS    RESTARTS   AGE
demo-app            1/1     Running   0          40s
jfsdemo-fuse-vfqgt  1/1     Running   0          40s
```

可以看到 Pod 已经创建成功，同时 JuiceFS 的 FUSE 组件也启动成功。

参考文档

1. JuiceFS 安装
2. JuiceFS 单机性能测试
3. JuiceFS 命令参考
4. Dragonfly 快速开始

5. Dragonfly Helm chart 配置文件
6. 如何在 Fluid 中使用 JuiceFS