数据加速器 GooseFS

实践教程

腾讯云

文档目录

实践教程
GooseFS 通过 MinIO 操作 COS

最近更新时间：2024-05-27 17:00:21

简介

本文提供了 GooseFS 集成 MinIO 的步骤。由于 GooseFS、MinIO 和 COS 三者均支持 AWS S3 协议，因此 GooseFS 可以很方便地与 MinIO 集成，下面将详细介绍操作步骤。

前提条件

1. 安装 golang1.17 以上。
2. 确保 make 命令可用。
3. 安装 Java 8。
4. 安装 SSH，确保能通过 SSH 连接到 LocalHost，并远程登录。
5. 安装 Git。
6. COS 服务上创建一个存储桶用于远端存储，操作指引请参见 控制台快速入门。

操作步骤

打通 MinIO 和 COS

如果在官网直接下载 MinIO 二进制包，并按照文档连接 COS，会发现非法 URI 的报错。这里的原因是 MinIO 会随机生成一个 BucketName，它不符合 COS 的 BucketName 规则（结尾是数字），因此 COS 无法正常响应。解决方法如下：

修改 MinIO 生成 BucketName 的逻辑

此方法的思路是修改 MinIO 生成 BucketName 的逻辑，使其生成符合规则的名称。

1. 从官方代码仓库拉取代码。

```
git clone https://github.com/minio/minio.git
```

2. 进入到 minio 所在目录，找到相应代码并修改。

```
cd minio
vim cmd/gateway/s3/gateway-s3.go
```

需要修改的函数是 randString，如下图所示。
修改部分已高亮显示，即在139行末尾添加了+"-000000"。

```
124 // randString generates random names and prepends them with a known prefix.
125 func randString(n int, src rand.Source, prefix string) string {
126         b := make([]byte, n)
127         // A rand.Int63() generates 63 random bits, enough for letterIdxMax letters!
128         for i, cache, remain := n-1, src.Int63(), letterIdxMax; i >= 0; {
129                 if remain == 0 {
130                         cache, remain = src.Int63(), letterIdxMax
131                 }
132                 if idx := int(cache & letterIdxMask); idx < len(letterBytes) {
133                         b[i] = letterBytes[idx]
134                         i--
135                 }
136                 cache >>= letterIdxBits
137                 remain--
138         }
139         return prefix + string(b[0:30-len(prefix)]) + "-000000"
140 }
```

3. 修改完成后，确认保存。
4. 编译 MinIO。

```
make
```

编译成功后 minio 目录下会出现 minio 可执行文件。

5. 设置环境变量，使得 MinIO 可以操作 COS。

```
export MINIO_ROOT_USER=SecretId
export MINIO_ROOT_PASSWORD=SecrectKey
```

可前往 API 密钥管理 页面获取 SecretId 和 SecretKey。

6. 启动 MinIO

```
./minio gateway s3 http://cos.ap-guangzhou.myqcloud.com
```

配置 GooseFS

1. 从官方仓库下载 GooseFS 安装包到本地。
   官方仓库下载链接：请前往 产品动态 下载最新版本。
2. 下载对应版本的S3插件**goosefs-underfs-s3a-1.2.0.jar**，将其放在 ${GOOSEFS_HOME}/lib目录下。
3. 执行如下命令，对安装包进行解压。

```
tar -zxvf goosefs-1.4.5-bin.tar.gz
cd goosefs-1.4.5
```

解压后，得到 goosefs-1.4.5，即 GooseFS 的主目录。下文将以 ${GOOSEFS_HOME} 表示该目录的绝对路径。

4. 在 ${GOOSEFS_HOME}/conf 的目录下，创建 goosefs-site.properties 配置文件。内容如下：

```
goosefs.master.mount.table.root.ufs=s3://{BucketName}/{DirectoryName}
goosefs.underfs.s3.endpoint={Endpoint}
goosefs.underfs.s3.disable.dns.buckets=true
goosefs.underfs.s3.inherit.acl=false
aws.accessKeyId={SecretId}
aws.secretKey={SecretKey}
```

5. 启用 GooseFS 前，执行如下命令，检查系统环境，确保 GooseFS 可以在本地环境中正确运行。

```
./bin/goosefs validateEnv local
```

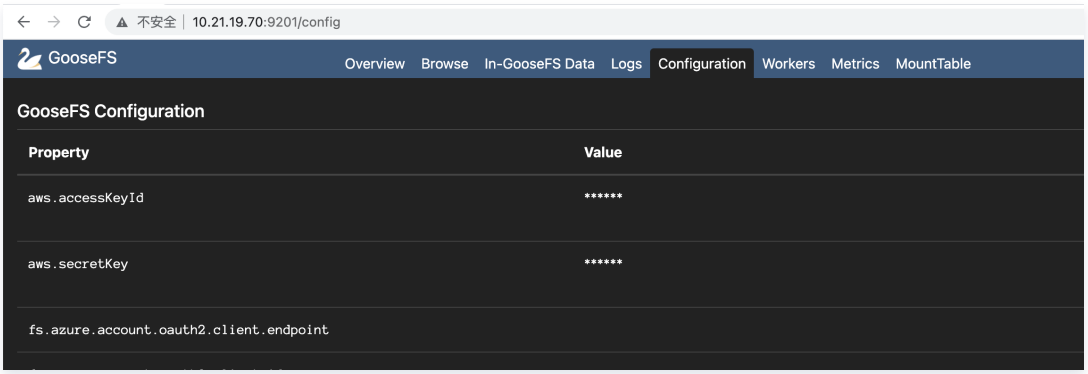
6. 执行如下命令，对 GooseFS 进行格式化。该命令将清除 GooseFS 的日志和 worker 存储目录下的内容。

```
./bin/goosefs format
```

7. 启动 GooseFS。

```
./bin/goosefs-start.sh local
```

该命令执行完毕后，可以访问 `http://localhost:9201` 和 `http://localhost:9204`，分别查看 Master 和 Worker 的运行状态，并且可以检查参数是否生效。如下图所示：

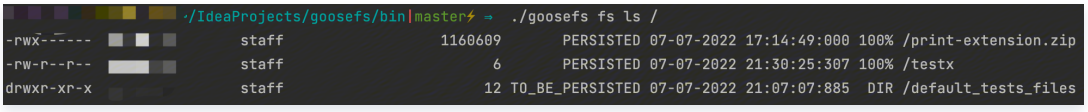


启动成功后，如下图所示：

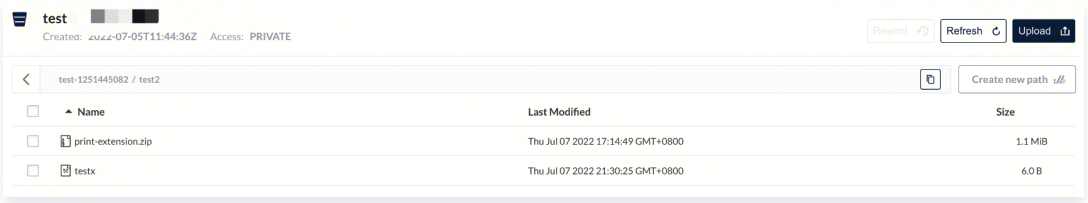
测试

我们通过 `goosefs fs` 命令对其进行简单的读写测试。

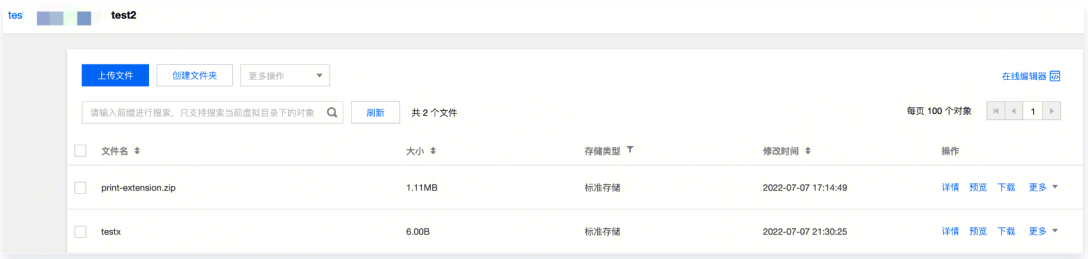
1. 查看当前根目录下有以下三个文件：



在 MinIO 中查看：

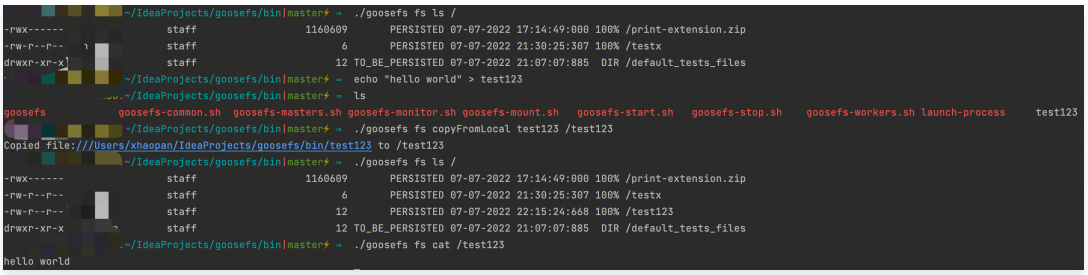


通过 COS 控制台查看：



可以看到它们是一致的。

2. 测试通过 goosefs fs 命令上传一个文件，并分别在 goosefs fs、MinIO 和 COS 控制台中验证结果。

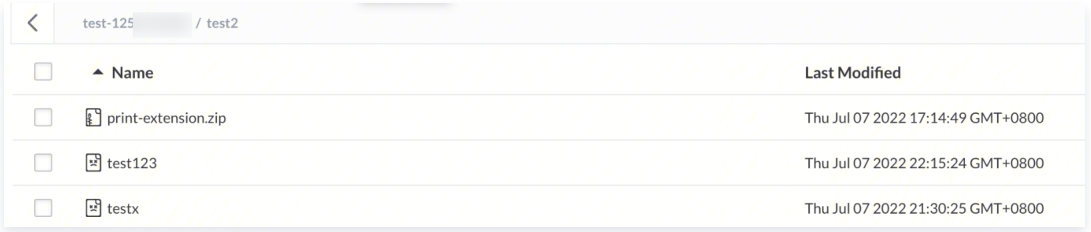


如上图所示，首先创建了一个名为 test123 的文件，内容是 hello world。

3. 通过 goosefs 将其上传，并利用 ls 和 cat 命令验证。
   在 MinIO 和 COS 控制台中可以看到这个文件，可以证明它们已经连通。
   - COS 控制台

文件名	大小	存储类型	修改时间
print-extension.zip	1.11MB	标准存储	2022-07-07 17:14:49
test123	12.00B	标准存储	2022-07-07 22:15:24
testx	6.00B	标准存储	2022-07-07 21:30:25

   - MinIO 控制台



在 Kubernetes 中使用 GooseFS 加速 Spark 数据访问

最近更新时间：2024-11-15 21:51:43

概述

在 Kubernetes 上运行的 Spark 可以将 GooseFS 用作数据访问层。本文将讲解如何在 Kubernetes 环境中使用 GooseFS 加速 Spark 的数据访问。

实践部署

环境与依赖版本

- CentOS 7.4+
- Kubernetes version 1.18.0+
- Docker 20.10.0
- Spark version 2.4.8+
- GooseFS 1.2.0+

Kubernetes 的部署

详细的 Kubernetes 部署可参见 Kubernetes 的官方文档 。

使用 GooseFS 加速 Spark 数据访问

目前，在 Kuberntes 中使用 GooseFS 加速 Spark 的数据访问主要有两种方式：

- 基于 Fluid 分布式数据编排与加速引擎（Fluid Operator 架构）部署运行 GooseFS Runtime Pods 和 Spark Runtime 加速 Spark 计算应用。
- 在 Kubernetes 中运行 Spark on GooseFS（Kubernetes Native 部署架构）。

基于 Fluid 部署运行 GooseFS on Kubernetes

Fluid 是 CNCF 基金会下的一个开源 Kubernetes 原生的分布式数据集编排和加速引擎，主要服务于云原生场景下的数据密集型应用，例如大数据应用、AI 应用等，详细介绍可参见 fluid 。
Fluid 也在 0.6.0 版本中正式集成了 GooseFS 的 Runtime（详见 更新日志 ），因此基于 Fluid 部署 GooseFS 加速 Spark 应用详情可参见 腾讯云 GooseFS 官网文档，这里也推荐 Fluid-0.7.0 以上版本部署运行 GooseFS。

在 Kubernetes 中运行 Spark on GooseFS

前置条件

1. Spark on Kubernetes 采用的是 Spark 官网推荐 Kubernetes Native 部署运行架构，详细的部署运行方法可参见 Spark 的官网文档 。
2. 已部署 GooseFS 集群。GooseFS 的集群部署可参见 使用自建集群部署 。

> ⚠ **注意：**
> 部署 GooseFS Worker 的时候，需要配置 goosefs.worker.hostname=$(hostname -i)，否则 Spark pod 中的 client 会无法解析 GooseFS 的 Worker 地址。

基本步骤

1. 首先，下载解压 spark-2.4.8-bin-hadoop2.7.tgz 。
2. 将 GooseFS client 从 GooseFS 的 Docker 镜像中解压出来，然后编译到 Spark 镜像中，如下所示：

```
# 将 GooseFS client 从 GooseFS 的 Docker 镜像中解压出来
$ id=$(docker create goosefs/goosefs:v1.2.0)
$ docker cp $id:/opt/alluxio/client/goosefs-1.2.0-client.jar - > goosefs-1.2.0-client.jar
$ docker rm -v $id 1>/dev/null
# 然后，copy 到 spark 的目录中
$ cp goosefs-1.2.0-client.jar /path/to/spark-2.4.8-bin-hadoop2.7/jars
# 然后，重新编译 spark 的 docker 镜像
$ docker build -t spark-goosefs:2.4.8 -f kubernetes/dockerfiles/spark/Dockerfile .
```

```
# 查看编译好的 docker image
$ docker image ls
```

```
Run 'docker image COMMAND --help' for more information on a command.
[hadoop@master ~]$ sudo docker image ls
REPOSITORY                                                TAG              IMAGE ID       CREATED          SIZE
spark-goosefs                                             2.4.8            2b897c83a9d7   39 minutes ago   599MB
fluidcloudnative/fluid-csi                                v0.7.0-3d66068   cb49a4c35af7   5 days ago       108MB
fluidcloudnative/alluxioruntime-controller                v0.7.0-3d66068   76e0a6d43df4   5 days ago       174MB
openjdk                                                   8-jdk-slim       ea05d9a6032a   6 days ago       296MB
registry.aliyuncs.com/google_containers/kube-apiserver    v1.23.4          62930710c963   2 weeks ago      135MB
registry.aliyuncs.com/google_containers/kube-proxy        v1.23.4          2114245ec4d6   2 weeks ago      112MB
registry.aliyuncs.com/google_containers/kube-scheduler    v1.23.4          aceacb6244f9   2 weeks ago      53.5MB
registry.aliyuncs.com/google_containers/kube-controller-manager   v1.23.4          25444908517a   2 weeks ago      125MB
ccr.ccs.tencentyun.com/qcloud/goosefs                     v1.2.0           6da0c30cb20e   3 weeks ago      2.28GB
rancher/mirrored-flannelcni-flannel                       v0.16.3          8cb5de74f107   5 weeks ago      59.7MB
rancher/mirrored-flannelcni-flannel-cni-plugin            v1.0.1           ac40ce625740   6 weeks ago      8.1MB
registry.aliyuncs.com/google_containers/etcd              3.5.1-0          25f8c7f3da61   4 months ago     293MB
registry.aliyuncs.com/google_containers/coredns           v1.8.6           a4ca41631cc7   5 months ago     46.8MB
registry.aliyuncs.com/google_containers/pause             3.6              6270bb605e12   6 months ago     683kB
registry.aliyuncs.com/acs/csi-node-driver-registrar       v1.2.0           c2103589e99f   2 years ago      17.1MB
[hadoop@master ~]$
```

测试步骤

首先，需要保证 GooseFS 集群已经启动运行，并且容器能够访问到 GooseFS Master/Worker 的 IP 和端口，然后按照以下步骤进行测试验证：

1. 在 GooseFS 中创建一个用于测试的 namespace，例如这里创建一个 /spark-cosntest 的namespace，并放入测试数据文件。

> **注意：**
> - 建议用户尽量避免在配置中使用永久密钥，采取配置子账号密钥或者临时密钥的方式有助于提升业务安全性。为子账号授权时建议按需授权子账号可执行的操作和资源，避免发生预期外的数据泄露。
> - 如果您一定要使用永久密钥，建议对永久密钥的权限范围进行限制，可通过限制永久密钥的可执行操作、资源范围和条件（访问 IP 等），提升使用安全性。

```
# 建议使用子账号密钥或者临时密钥的方式完成配置，提升配置安全性。为子账号授权时建议按需授权子账号可执行的操作和资源
$ goosefs ns create spark-cosntest cosn://goosefs-test-125000000/ --secret
fs.cosn.userinfo.secretId=********************************** --secret
fs.cosn.userinfo.secretKey=********************************** --attribute fs.cosn.bucket.region=ap-
xxxx
# 放入一个测试数据文件
$ goosefs fs copyFromLocal LICENSE /spark-cosntest
```

2. （可选）创建一个用于运行 spark 作业的服务账号。

```
$ kubectl create serviceaccount spark
$ kubectl create clusterrolebinding spark-role --clusterrole=edit \
  --serviceaccount=default:spark --namespace=default
```

3. 提交一个 spark 作业。

```
  --master k8s://http://127.0.0.1:8001 \
  --deploy-mode cluster \
  --name spark-goosefs \
  --class org.apache.spark.examples.JavaWordCount \
  --conf spark.executor.instance=2 \
  --conf spark.kubernetes.container.image=spark-goosefs/spark:2.4.8 \
  --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
  --conf spark.hadoop.fs.gfs.impl=com.qcloud.cos.goosefs.hadoop.GooseFileSystem \
  --conf spark.driver.extraClassPath=local:///opt/spark/jars/goosefs-1.2.0-client.jar \
  local:///opt/spark/examples/jars/spark-examples_2.11-2.4.8.jar \
  gfs://172.16.64.32:9200/spark-cosntest/LICENSE
```

4. 等待执行完成即可。

```
[root@master ~]# kubectl get pods
NAME                                READY   STATUS      RESTARTS   AGE
spark-goosefs-1646905692480-driver   0/1     Completed   0          105s
spark-pi-1646654681148-driver        0/1     Completed   0          2d21h
```

执行 `kubectl logs spark-goosefs-1646905692480-driver` 查看作业执行结果即可。

```
distribution: 5
warranties,: 1
depends: 1
hardware): 1
7.: 2
mailing: 1
continue: 1
with: 26
Notwithstanding: 1
January: 1
2: 1
CONTRACT,: 3
cross-claim: 2
(C): 1
provide: 1
(such: 1
lawsuit): 2
arising: 2
publish,: 1
based: 2
publish: 1
met:: 1
jQuery: 2
medium,: 1
Software,: 1
must:: 1
authorship.: 1
files.: 1
every: 1
claim,: 1
hardware: 1
herein,: 1
copies: 4
import: 1
remove: 1
statement: 1
their: 3
(excluding: 1
work: 5
state: 1
example,: 2
by,: 3
actions: 1
appear.: 1
attached: 1
INDIRECT,: 2
("Commercial: 1
Your: 9
```

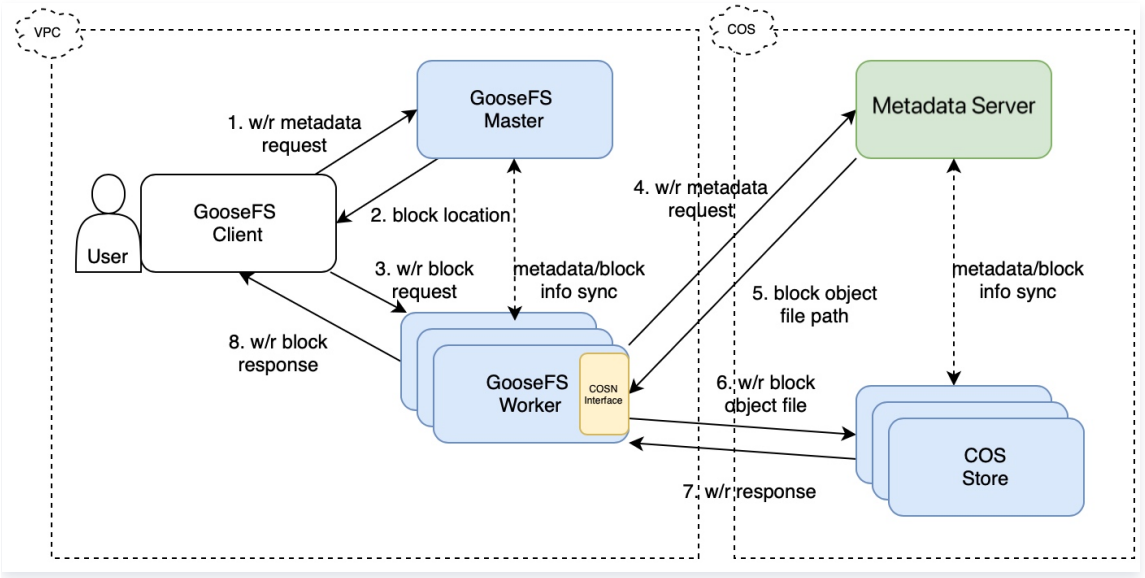
使用 GooseFS 以原生 POSIX 语义访问存储桶

最近更新时间：2024-11-15 21:51:43

背景

对象存储服务通过 元数据加速能力 提供了原生的 POSIX 语义接口，支持用户通过文件系统语义访问对象存储服务，提供原生的元数据操作能力。系统设计指标可以达到 Gb 级单链接带宽、10万级 QPS 以及 ms 级延迟。启用元数据加速功能后，可以提升集群对元数据的操作性能，例如 List、Rename 等操作，并且支持原生的 Append、Truncate 操作，可以广泛应用于大数据、高性能计算、机器学习、AI 等场景。

GooseFS 在 GooseFS V1.3+ 版本集成了最新版本的 COSN interface（COSN V8.14+ 版本），支持了原生 POSIX 语义访问对象存储服务。整体的文件读写流程框架如下：



区别于原生 COS 协议，通过原生 POSIX 语义访问存储桶具有如下优势：

1. 更全面的 POSIX 语义兼容，提供原生的 Append 、Truncate 操作支持；
2. 更高性能的文件元数据操作，支持 10 万级的List/Rename QPS；
3. 更低延迟的文件数据操作，大数据场景下，能有效降低大文件的读写延迟。

> ⚠ **注意**
> GooseFS 暂时不支持通过 Append、Truncate 接口操作对象存储服务，GooseFS-Lite 客户端支持。如有需要，可下载使用。

前提条件

通过 GooseFS 以原生 POSIX 语义访问对象存储服务的前提条件如下：

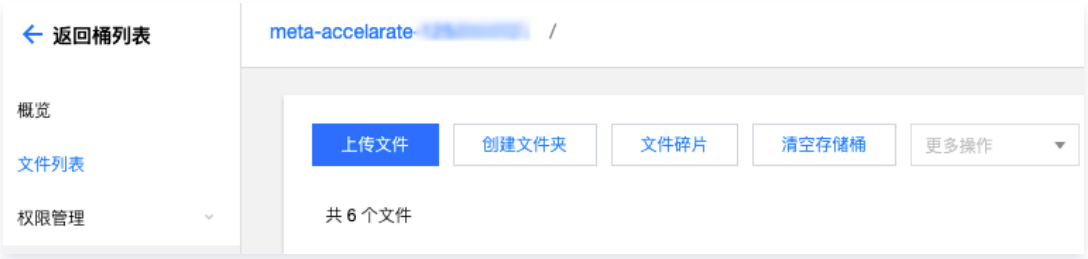
1. 在 COS 服务上创建一个存储桶以作为远端存储，操作指引请参见 控制台快速入门 。
2. 确保您的存储桶已开启元数据加速服务能力，元数据加速能力只能在创建存储桶时开启。可参见 使用 HDFS 协议访问已开启元数据加速器的存储桶 。
3. 安装 GooseFS V1.3+ 以上版本的 GooseFS 客户端和服务端安装包。可前往 产品动态 下载最新版本的 GooseFS 软件。
   - 安装 GooseFS 前，必须先安装 Java 8 或者更高的版本 。
   - 安装 GooseFS 前，必须先安装 SSH，确保能通过 SSH 连接到 LocalHost，并远程登录。
4. 安装完成后，在 `core-site.properties` 文件中修改访问协议的配置，即可通过原生的 POSIX 协议访问指定存储桶。

操作步骤

创建存储桶并配置 HDFS 协议

1. 创建 COS 存储桶，并且开启元数据加速器。如下图所示：

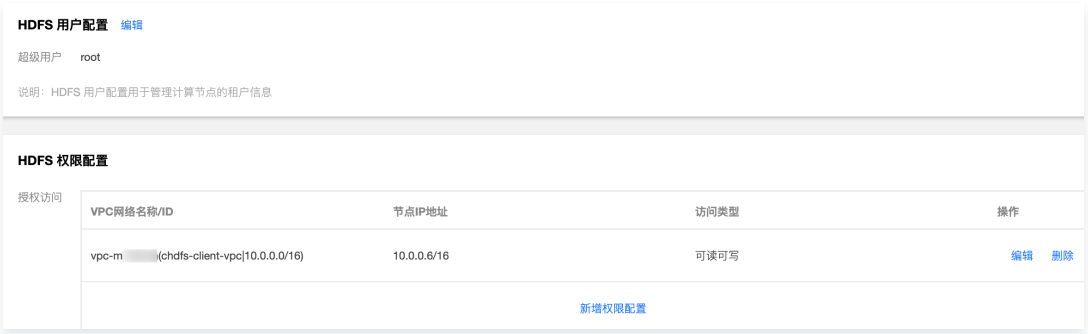
当存储桶创建完成后，进入存储桶的**文件列表**页面，您可在该页面进行文件上传和下载操作。如下图所示：



2. 在左侧菜单栏中，单击**性能配置 > 元数据加速能力**，可以看到元数据加速能力已开启。
如果是第一次创建**需开启元数据加速**的存储桶，需要按照提示进行相应的**授权**操作，单击授权完成后，将自动开启 HDFS 协议，并且看到默认的存储桶挂载点信息。如下图所示：

> **说明**
> 如果提示未找到对应的 HDFS 文件系统，请单击 提交工单 联系我们获取帮助。

3. 在 HDFS 权限配置栏中，单击**新增权限配置**。



4. 在 VPC 网络名称列选择计算集群所在的 VPC 网络地址，在节点 IP 地址列填写 VPC 网段下需要放通的 IP 地址或者 IP 段，访问类型选择读写或者只读，配置完成后，单击**保存**即可。

> **说明**
> HDFS 权限配置与原生 COS 权限体系存在差异。当您使用 HDFS 协议访问时，推荐通过配置 HDFS 权限授权指定 VPC 内机器访问 COS 存储桶，以便获取和原生 HDFS 一致的权限体验。

下载并安装好 GooseFS

1. 务必按照上方的**前提条件**，安装好对应的 JDK、SSH 以及依赖的 JAR 包。
   JAR 包需要放到各节点 classpath 下保证任务启动能正常加载，例如 `$HADOOP_HOME/share/hadoop/common/lib/` 下。
2. 可参见 产品动态，从官方仓库下载 GooseFS 安装包到本地。
3. 执行如下命令，对安装包进行解压。

```
$ tar -zxvf goosefs-1.3.0-bin.tar.gz
$ cd goosefs-1.3.0
```

   解压后，得到 goosefs-1.3.0，即 GooseFS 的主目录。下文将以 `${GOOSEFS_HOME}` 代指该目录的绝对路径。
4. 在 `${GOOSEFS_HOME}/conf` 的目录下，创建 `conf/goosefs-site.properties` 的配置文件。可以使用内置的配置模板：

```
$ cp conf/goosefs-site.properties.template conf/goosefs-site.properties
```

5. 在配置文件 `conf/goosefs-site.properties` 中，将 `goosefs.master.hostname` 设置为 `localhost`：

```
$ echo  "goosefs.master.hostname=localhost">> conf/goosefs-site.properties
```

修改配置文件以支持通过 POSIX 语义访问 COS

1. 在完成 GooseFS 初步安装后，编辑 `core-site.xml` 文件，新增以下基本配置：

> **注意**
> - 建议用户尽量避免在配置中使用永久密钥，采取配置子账号密钥或者临时密钥的方式有助于提升业务安全性。为子账号授权时建议按需授权子账号可执行的操作和资源，避免发生预期外的数据泄露。
> - 如果您一定要使用永久密钥，建议对永久密钥的权限范围进行限制，可通过限制永久密钥的可执行操作、资源范围和条件（访问 IP 等），提升使用安全性。

```
<!--账户的 API 密钥信息。可登录 [访问管理控制台](https://console.cloud.tencent.com/capi) 查看云 API 密钥。-->
<!--建议使用子账号密钥或者临时密钥的方式完成配置，提升配置安全性。为子账号授权时建议按需授权子账号可执行的操作和资源-->
<property>
      <name>fs.cosn.userinfo.secretId/secretKey</name>
      <value>**********************************</value>
</property>
```

```
<!--cosn 的实现类-->
<property>
      <name>fs.AbstractFileSystem.cosn.impl</name>
      <value>org.apache.hadoop.fs.CosN</value>
</property>

<!--cosn 的实现类-->
<property>
      <name>fs.cosn.impl</name>
      <value>org.apache.hadoop.fs.CosFileSystem</value>
</property>

<!--用户存储桶的地域信息，格式形如 ap-guangzhou-->
<property>
      <name>fs.cosn.bucket.region</name>
      <value>ap-guangzhou</value>
</property>

<!--本地临时目录，用于存放运行过程中产生的临时文件-->
<property>
      <name>fs.cosn.tmp.dir</name>
      <value>/tmp/hadoop_cos</value>
</property>
```

2. 将 `core-site.xml` 同步到所有 `hadoop` 节点上。
   完成这一步骤后，即可通过 POSIX 语义访问 COS 存储桶。

GooseFS Distributedload 调优实践

最近更新时间：2024-10-12 11:37:31

简介

GooseFS 为用户提供了一个就近计算端的缓存文件系统，当文件存储在远端存储系统时（例如对象存储），用户可以通过 distributedload 指令将需要使用的文件数据和元数据加载到 GooseFS 集群中，起到减少访问延迟、加速计算作业的效果。

GooseFS distributedLoad 指令如下：

```
distributedLoad [-A] [--replication <num>] [--active-jobs <num>] [--expire-time] <path>
-A 是否启用原子性的distributed load 能力
--active-jobs <active job count> 可以同时启用的数据加载任务数量，默认上限值为3000，如果超过该数值，新增任务需要等候当前任务完成后再执行。
--expire-time <arg> 设置清除用于数据加载的临时目录的过期时间，默认为24小时（单位默认ms，支持s，min，hour，如100s）
--replication <replicas> 每个加载任务加载的Block数据副本数，默认为1。
```

实践步骤

流程说明

distributedLoad 指令的完整执行流程涉及到 GooseFS client、JobMaster、JobWorker、Worker 模块，分环节说明如下：

1. 由 GooseFS Client 发起任务。
2. 由 GooseFS JobMaster 将每个文件转换为 Job，并按照 Block 的集合拆分为不同的 Task。
3. 由 GooseFS JobMaster 将不同的 Task 拆分给 JobWorker ，有不同的 JobWorker 发起 Load 操作。
4. GooseFS JobWorker 并发执行 Tasks，每个 Task 都会向 Worker 执行请求读操作，并发起写操作，其中每个读操作会向 Cosn 发起一个 Read request。

根据上述流程，可以推断出可能存在的瓶颈点如下：

1. GooseFS Client 按照 file 粒度去发起任务的并发。
2. GooseFS JobMaster 组织并分发任务的速度。
3. GooseFS JobWorker 执行任务的并发度。
4. GooseFS Worker 线程执行读操作与写操作的并发度。
5. Cosn 的处理 Read Request 的速度。

其中，第2项组织任务是内存操作，基本为信令流操作，分发任务的周期为 Worker 的心跳间隔（1s），基本上不会产生瓶颈。因此，主要调优方向集中在 JobWorker、Worker 和 Cosn 模块，涉及的关键参数如下：

1. JobWorker 模块：

```
goosefs.user.block.worker.client.pool.max: 建立读写流的时候，会从 client pool 中去获取，过小会阻塞获取 client
goosefs.job.worker.max.active.task.num : 同时允许执行的 task 数目
goosefs.job.worker.threadpool.size: 处理 task 的线程数目
goosefs.user.block.worker.client.pool.max: 设置 jobworker worker client 数量，过小会阻塞获取 client
```

2. Worker 模块：

```
goosefs.worker.network.reader.buffer.size:会影响 worker 的内存占用
goosefs.worker.network.block.reader.threads.max:决定了 worker 用于 read 的最大线程数目
```

3. Cosn 模块：

```
fs.cosn.block.size: load 上来的 block 的大小
```

```
fs.cosn.upload_thread_pool： read thread 的大小，每个 worker 会共用一个
fs.cosn.read.ahead.block.size：cosn 会按照这个粒度去请求 cos
fs.cosn.read.ahead.queue.size：cosn 是有预读功能的，主要针对大文件顺序读场景，而 load 刚好是顺序读，但是不一定是大文件。
```

配置调优

Cosn 配置

GooseFS 缓存集群提供的吞吐规模取决于集群和对象存储之间的吞吐情况，依赖于 Cosn 的性能，一般情况下可以根据业务需要调整 Cosn 的配置：
在大数据场景下，文件平均大小比较大，推荐配置如下：

1. `fs.cosn.block.size`：推荐设置为128MB。
2. `fs.cosn.upload_thread_pool`：推荐设置为 CPU 数目的2 - 3倍，也需要根据 CPU 的使用率适当增加或降低线程池数量。
3. `fs.cosn.read.ahead.block.size`：需要根据 block 大小来调整：
   - 如果文件平均大小为几十 MB 量级，则可以设置为4MB。
   - 如果为 MB 级或者 KB 级文件，推荐设置为文件大小的平均值或者中位值。尽量一次 rpc 可以读回来，其原因是数据长度占用了 HTTP 请求较小的 body，rpc 的消耗可能会大于本身读数据的消耗，所以尽量减少 rpc 次数。
4. `fs.cosn.read.ahead.queue.size`：推荐设置为8 - 32，需要根据内存容量数值来设置。一般情况下，一个文件输入流（inputstream）的内存占用等于 block.size*queue.size。由于 block 大小等于 `fs.cosn.block.size` / `fs.cosn.read.ahead.block.size`，以推荐设置为例，该数值等于32，所以设置值不需要大于32，如果设置超过32反而会浪费资源。

Worker 节点配置

确定 Cosn 配置之后，可以进一步明确 Worker 节点配置：
Worker 节点配置：`goosefs.worker.network.reader.buffer.size`，这个值也需要根据内存评估，读操作总共占用的内存大小等于 `worker read` 并发数量 x（`buffer.size` + 一个 `inputstream` 的内存占用 + 单次 `readRequest` 的长度（默认是1MB））。

JobWorker 配置

接下来明确 JobWorker 的配置：

1. `goosefs.job.worker.max.active.task.num`：这一数值可以略大于 `fs.cosn.upload_thread_pool` 的数值，从而充分利用 cosn 的能力。
2. `goosefs.user.block.worker.client.pool.max`：默认是1024，原则上要保证这个配置值需要等于 `goosefs.job.worker.max.active.task.num` 的2倍。
3. `goosefs.job.worker.threadpool.size`：默认是10，这一数值的最大值需要小于等于 `goosefs.job.worker.max.active.task.num`。

使用 GooseFS Client 发起操作时，可以根据 `goosefs.job.worker.max.active.task.num` 的值调整客户端的活跃任务数量，原则上需要保证 `active-jobs` 的值大于 `goosefs.job.worker.max.active.task.num` 的值。