阿里云
E-MapReduce

产品内核增强

文档版本：20200708

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止：** 重置操作将丢失用户配置数据。
	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告：** 重启操作将导致业务中断，恢复业务时间约十分钟。
	用于警示信息、补充说明等，是用户必须了解的内容。	 **注意：** 权重设置为0，该服务器不会再接受新请求。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明：** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	单击**设置** > **网络** > **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令。	执行cd /d C:/window命令，进入Windows系统文件夹。
斜体	表示参数、变量。	bae log list --instanceid Instance_ID
[]或者[a|b]	表示可选项，至多选择一个。	ipconfig [-all|-t]
{}或者{a|b}	表示必选项，至多选择一个。	switch {active|stand}

目录

1 JindoFS基础使用（EMR-3.27.0之前版本）

1.1 JindoFS使用说明（EMR-3.20.0~3.22.0版本）

本文主要介绍JindoFS的配置使用方式，以及一些典型的应用场景。

概述

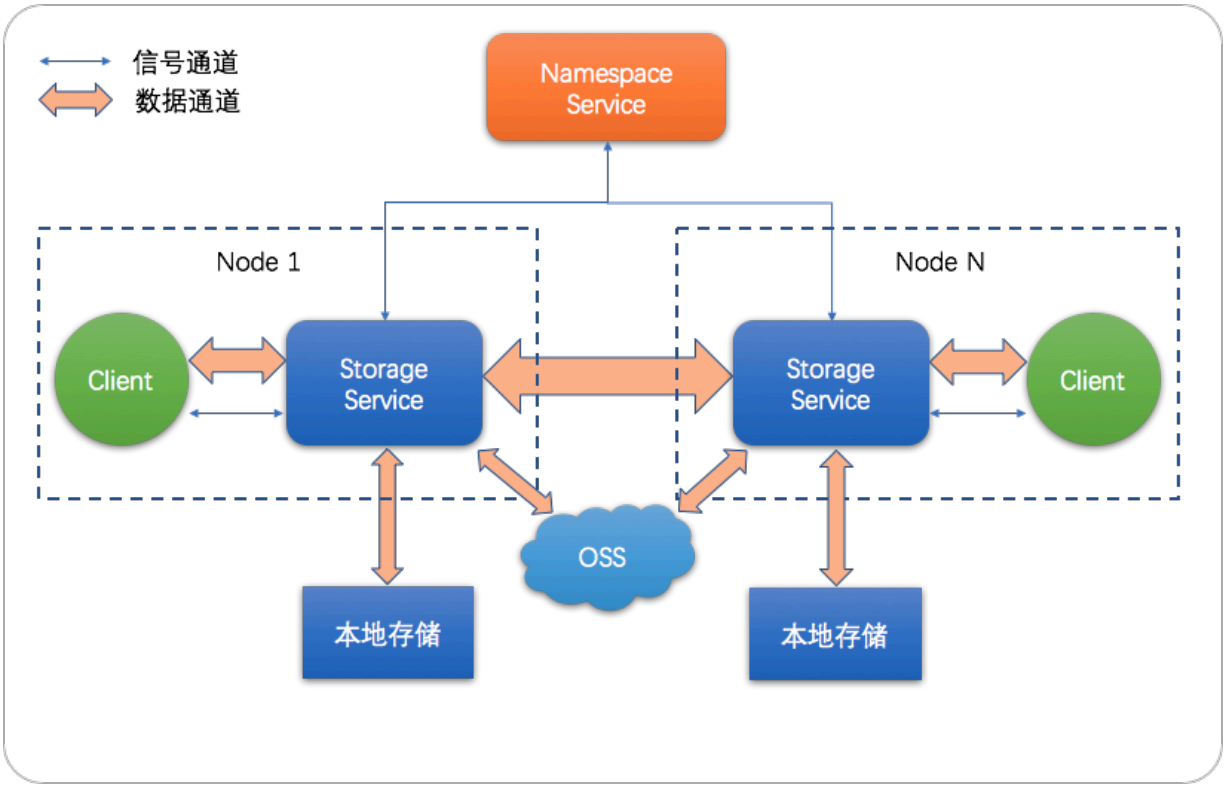
JindoFS是一种云原生的文件系统，结合OSS和本地存储，成为E-MapReduce产品的新一代存储系统，为上层计算提供了高效可靠的存储。

JindoFS 提供了块存储模式（Block）和缓存模式（Cache）的存储模式。

JindoFS 采用了本地存储和OSS的异构多备份机制，Storage Service提供了数据存储能力，首先使用OSS作为存储后端，保证数据的高可靠性，同时利用本地存储实现冗余备份，利用本地的备份，可以加速数据读取；另外，JindoFS 的元数据通过本地服务Namespace Service管理，从而保证了元数据操作的性能（和HDFS元数据操作性能相似）。

说明：

- E-MapReduce-3.20.0及以上版本支持Jindo FS，您可以在创建集群时勾选相关服务来使用JindoFS。
- 本文主要是E-MapReduce-3.20.0及以上版本至E-MapReduce-3.22.0（但不包括）版本的介绍；E-MapReduce-3.22.0 及以上版本的JindoFS 使用说明，请参见JindoFS使用说明（EMR-3.22.0~3.26.3版本）。



应用场景

E-MapReduce目前提供了三种大数据存储系统，E-MapReduce OssFileSystem、E-MapReduce HDFS和E-MapReduce JindoFS，其中OssFileSystem和JindoFS都是云上存储的解决方案，下表为这三种存储系统和开源OSS各自的特点。

特点	开源OSS	E-MapReduce OssFileSystem	E-MapReduce HDFS	E-MapReduce JindoFS
存储空间	海量	海量	取决于集群规模	海量
可靠性	高	高	高	高
吞吐率因素	服务端	集群内磁盘缓存	集群内磁盘	集群内磁盘
元数据效率	慢	中	快	快
扩容操作	容易	容易	容易	容易
缩容操作	容易	容易	需Decommission	容易
数据本地化	无	弱	强	较强

JindoFS块存储模式具有以下几个特点：

- 海量弹性的存储空间，基于OSS作为存储后端，存储不受限于本地集群，而且本地集群能够自由弹性伸缩。

- 能够利用本地集群的存储资源加速数据读取，适合具有一定本地存储能力的集群，能够利用有限的本地存储提升吞吐率，特别对于一写多读的场景效果显著。
- 元数据操作效率高，能够与HDFS相当，能够有效规避OSS文件系统元数据操作耗时以及高频访问下可能引发不稳定的问题。
- 能够最大限度保证执行作业时的数据本地化，减少网络传输的压力，进一步提升读取性能。

环境准备

- 创建集群

  选择E-MapReduce-3.20.0及以上版本至E-MapReduce-3.22.0（但不包括）版本，勾选可选服务中的**SmartData**和**Bigboot**，创建集群详情请参见#unique_6。Bigboot 服务提供了E-MapReduce平台上的基础的分布式数据管理交互服务以及一些组件管理监控和支持性服务，SmartData服务基于Bigboot之上对应用层提供了JindoFS文件系统。

- 配置集群

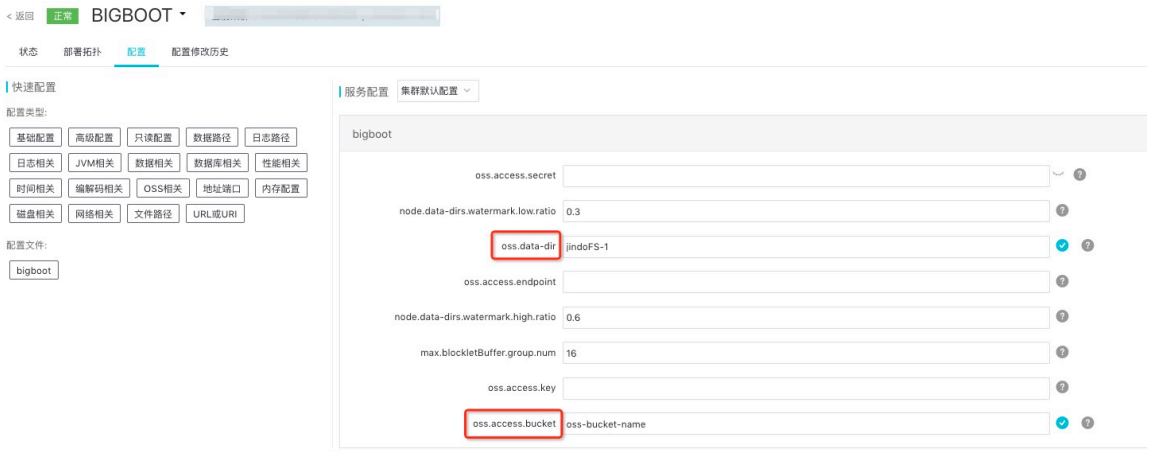
  SmartData提供的JindoFS文件系统使用OSS作为存储后端，因此在使用JindoFS之前需配置一些OSS相关参数。下面提供两种配置方式，第一种是先创建好集群，修改Bigboot相关参数，需重

启SmartData服务生效；第二种是创建集群过程中添加自定义配置，这样集群创建好后相关服务就能按照自定义参数启动。

- 集群创建好后参数初始化
  - oss.access.bucket为OSS bucket的名称。
  - oss.data-dir为JindoFS在OSS bucket中所使用的目录（注：该目录为 JindoFS后端存储目录，生成的数据不能人为破坏，并且保证该目录仅用于JindoFS后端存储，JindoFS在写入数据时会自动创建用户所配置的目录，无需在OSS上事先创建）。
  - oss.access.endpoint为bucket所在的区域。
  - oss.access.key为存储后端OSS的AccessKey ID。
  - oss.access.secret为存储后端OSS的AccessKey Secret。

  考虑到性能和稳定性，推荐使用同region下的OSS bucket作为存储后端，此时，E-MapReduce集群能够免密访问OSS，无需配置AccessKey ID和AccessKey Secret。

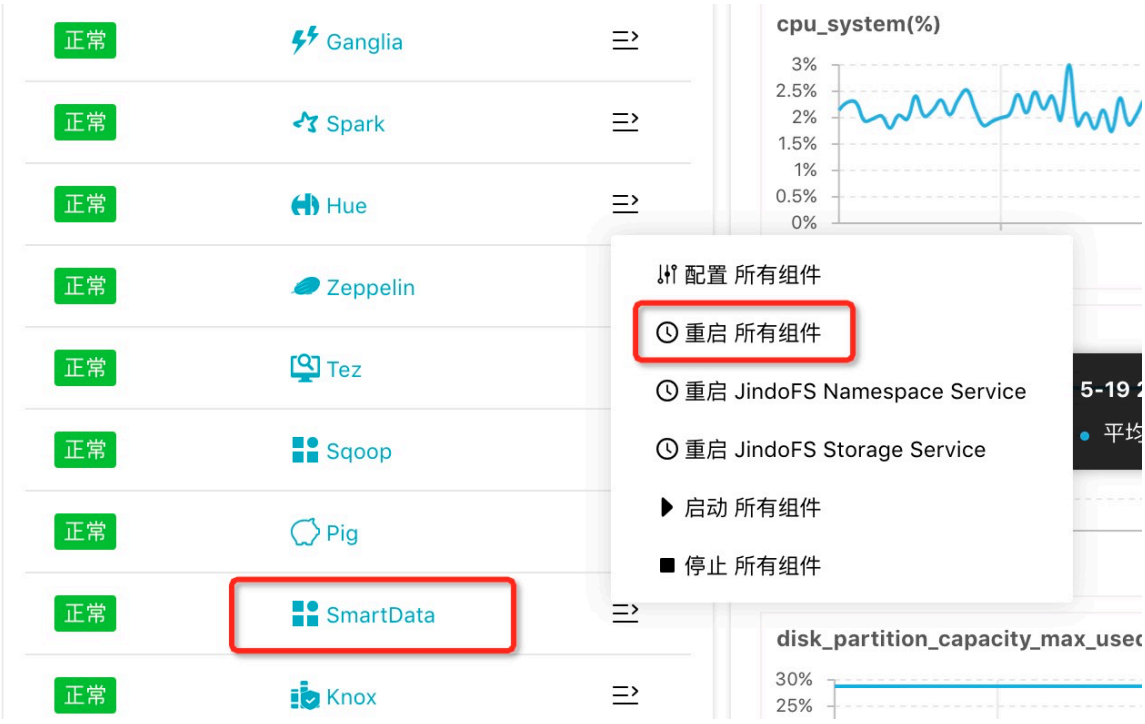
  所有JindoFS相关配置都在Bigboot组件中，配置如下图所示，红框中为必填的配置项。



说明：

JindoFS支持多命名空间，本文命名空间以test为例。

配置完成后保存并部署，然后在SmartData服务中重启所有组件，即开始使用JindoFS。



- 创建集群时添加自定义配置

E-MapReduce集群在创建集群时支持添加自定义配置，以同region下免密访问OSS为例，如下图勾选**软件自定义配置**，添加如下配置，配置oss.data-dir和oss.access.bucket。

```
[
  {
  "ServiceName":"BIGBOOT",
  "FileName":"bigboot",
  "ConfigKey":"oss.data-dir",
  "ConfigValue":"jindoFS-1"
  },
  {
  "ServiceName":"BIGBOOT",
  "FileName":"bigboot",
  "ConfigKey":"oss.access.bucket",
  "ConfigValue":"oss-bucket-name"
  }
```

```
]
```

使用JindoFS

JindoFS使用上与HDFS类似，提供jfs前缀，将jfs替代hdfs即可使用。简单示例：

```
hadoop fs -ls jfs:/// hadoop fs -mkdir jfs:///test-dirhadoop fs -put test.log jfs:///test-dir/
```

目前，JindoFS能够支持 E-MapReduce 集群上的 Hadoop、Hive、Spark的作业进行访问，其余组件尚未完全支持。

磁盘空间水位控制

JindoFS后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFS会自动淘汰本地较冷的数据备份。我们提供了**node.data-dirs.watermark.high.ratio**和**node.data-dirs**

.watermark.low.ratio这两个参数用来调节本地存储的使用容量，值均为0～1的小数表示使用比例，JindoFS默认使用所有数据盘，每块盘的使用容量默认即为数据盘大小。前者表示使用量上水位比例，每块数据盘的JindoFS占用的空间到达上水位即会开始清理淘汰；后者表示使用量下水位比例，触发清理后会将JindoFS的占用空间清理到下水位。用户可以通过设置上水位比例调节期望分给JindoFS的磁盘空间，下水位必须小于上水位，设置合理的值即可。

存储策略

JindoFS提供了Storage Policy功能，提供更加灵活的存储策略适应不同的存储需求，可以对目录设置以下四种存储策略。

策略	策略说明
COLD	表示数据仅在OSS上有一个备份，没有本地备份，适用于冷数据存储。
WARM	默认策略。 表示数据在OSS和本地分别有一个备份， 本地备份能够有效的提供后续的读取加速。
HOT	表示数据在OSS上有一个备份，本地有多个备份，针对一些最热的数据提供更进一步的加速效果。
TEMP	表示数据仅有一个本地备份，针对一些临时性数据，提供高性能的读写，但降低了数据的高可靠性，适用于一些临时数据的存取。

JindoFS提供了Admin工具设置目录的Storage Policy（默认为 WARM），新增的文件将会以父目录所指定的Storage Policy进行存储，使用方式如下所示。

```
jindo dfsadmin -R -setStoragePolicy [path] [policy]
```

通过以下命令，获取某个目录的存储策略。

```
jindo dfsadmin -getStoragePolicy [path]
```

说明：

其中[path] 为设置policy的路径名称， -R表示递归设置该路径下的所有路径。

Admin工具还提供archive命令，实现对冷数据的归档。

此命令提供了一种用户显式淘汰本地数据块的方式。Hive分区表按天分区，假如业务上对一周前的分区数据认为不会再经常访问，那么就可以定期将一周前的分区目录执行archive，淘汰本地备份，文件备份将仅仅保留在后端OSS上。

Archive命令的使用方式如下：

```
jindo dfsadmin -archive [path]
```

说明：

[path]为需要归档文件的所在目录路径。

1.2 JindoFS使用说明（EMR-3.22.0~3.26.3版本）

JindoFS是一种云原生的文件系统，结合OSS和本地存储，成为E-MapReduce产品的新一代存储系统，为上层计算提供了高效可靠的存储。本文主要说明JindoFS的配置使用方式，以及介绍一些典型的应用场景。

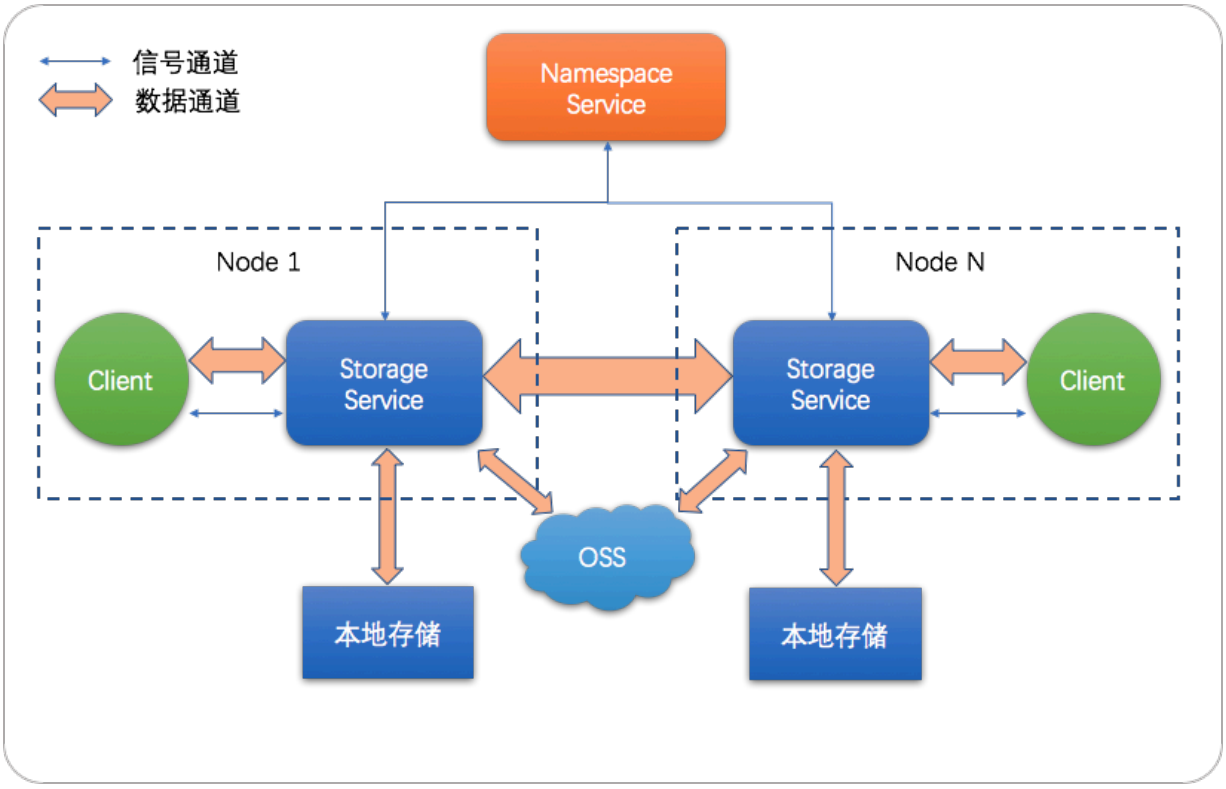
概述

JindoFS提供了块存储模式（Block）和缓存模式（Cache）的存储模式。

JindoFS采用了本地存储和OSS的异构多备份机制，Storage Service提供了数据存储能力，首先使用OSS作为存储后端，保证数据的高可靠性，同时利用本地存储实现冗余备份，利用本地的备份，可以加速数据读取；另外，JindoFS的元数据通过本地服务Namespace Service管理，从而保证了元数据操作的性能（和HDFS元数据操作性能相似）。

说明：

- E-MapReduce-3.20.0及以上版本支持Jindo FS，您可以在创建集群时勾选相关服务来使用JindoFS。
- 本文主要是E-MapReduce-3.22.0及以上版本的介绍；E-MapReduce-3.20.0及以上版本至E-MapReduce-3.22.0（但不包括）版本的JindoFS使用说明，请参见JindoFS使用说明（EMR-3.20.0~3.22.0版本）。



环境准备

- 创建集群

  选择E-MapReduce-3.22.0及以上版本，勾选可选服务中的**SmartData**，创建集群详情请参见#unique_6。

- 配置集群

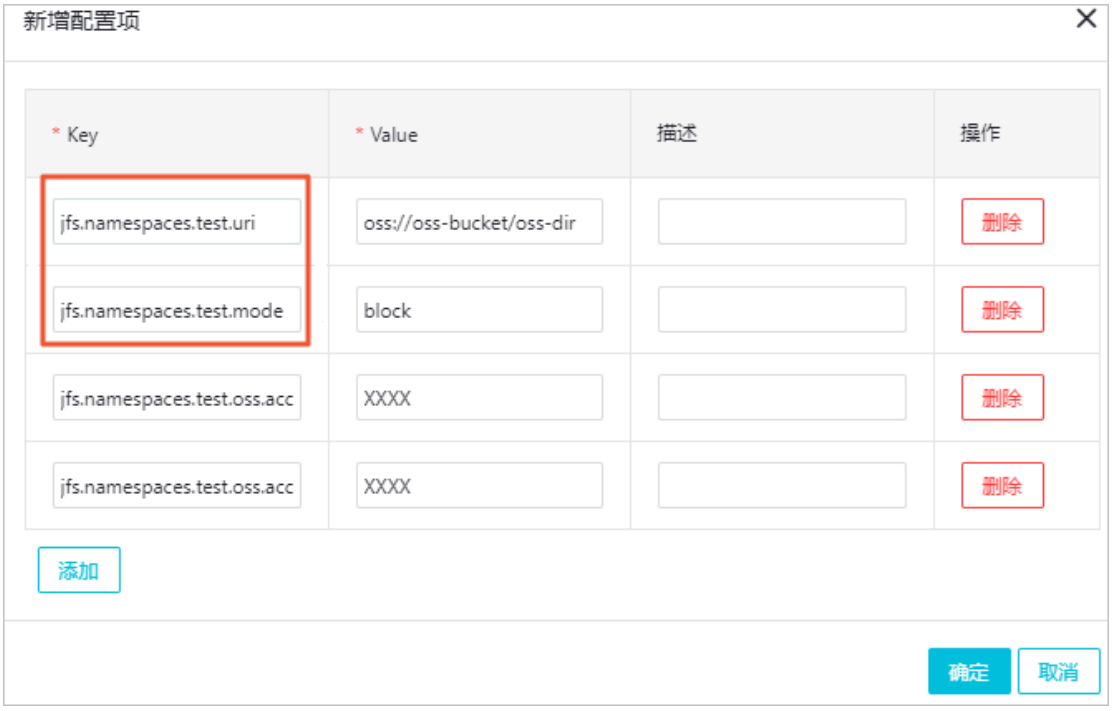
  SmartData提供的JindoFS文件系统使用OSS作为存储后端，因此在使用JindoFS之前需配置一些OSS相关参数。下面提供两种配置方式，第一种是先创建好集群，修改Bigboot相关参数，需重启SmartData服务生效；第二种是创建集群过程中添加自定义配置，这样集群创建好后相关服务就能按照自定义参数启动：

  - 集群创建好后初始化参数

    所有JindoFS相关配置都在Bigboot组件中，配置如下所示：

    1. 在**服务配置**页面，单击**bigboot**页签。

    2. 单击**自定义配置**。



说明：

■ 红框中为必填的配置项。

■ JindoFS支持多命名空间，本文命名空间以test为例。

参数	参数说明	示例
jfs.namespaces	表示当前JindoFS支持的命名空间，多个命名空间时以逗号隔开。	test
jfs.namespaces.test.uri	表示test命名空间的后端存储。	oss://oss-bucket/oss-dir **说明：** 该配置也可以配置到OSS bucket下的具体目录，该命名空间即以该目录作为根目录来读写数据。
jfs.namespaces.test.mode	表示test命名空间为块存储模式。	block **说明：** JindoFS支持block和cache两种存储模式。

参数	参数说明	示例
jfs.namespaces.test.oss.access.key	表示存储后端OSS的AccessKey ID。	xxxx **说明：** 考虑到性能和稳定性，推荐使用同账户、同region下的OSS bucket作为存储后端，此时，E-MapReduce集群能够免密访问OSS，无需配置AccessKey ID和AccessKey Secret。
jfs.namespaces.test.oss.access.secret	表示存储后端OSS的AccessKey Secret。	

配置完成后保存并部署，然后在SmartData服务中重启所有组件，即开始使用JindoFS。

- 创建集群时添加自定义配置

E-MapReduce集群在创建集群时支持添加自定义配置，以同region下免密访问OSS为例，如下图勾选**软件自定义配置**，配置命名空间test的相关配置，详情如下。

```
[
{
"ServiceName":"BIGBOOT",
"FileName":"bigboot",
"ConfigKey":"jfs.namespaces","ConfigValue":"test"
},{
"ServiceName":"BIGBOOT",
"FileName":"bigboot",
"ConfigKey":"jfs.namespaces.test.uri",
```

```
"ConfigValue":"oss://oss-bucket/oss-dir"
},{
"ServiceName":"BIGBOOT",
"FileName":"bigboot",
"ConfigKey":"jfs.namespaces.test.mode",
"ConfigValue":"block"
}
]
```

使用JindoFS

JindoFS使用上与HDFS类似，提供jfs前缀，将jfs替代hdfs即可使用。

目前，JindoFS能够支持EMR集群上的大部分计算组件，包括Hadoop、Hive、Spark、Flink、Presto和Impala。

简单示例：

- Shell命令

```
hadoop fs -ls jfs://your-namespace/
hadoop fs -mkdir jfs://your-namespace/test-dir
hadoop fs -put test.log jfs://your-namespace/test-dir/
```

```
hadoop fs -get jfs://your-namespace/test-dir/test.log ./
```

- MapReduce作业

```
hadoop jar /usr/lib/hadoop-current/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.5.jar teragen -Dmapred.map.tasks=1000 10737418240 jfs://your-namespace/terasort/input
hadoop jar /usr/lib/hadoop-current/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.5.jar terasort -Dmapred.reduce.tasks=1000 jfs://your-namespace/terasort/input jfs://your-namespace/terasort/output
```

- Spark-SQL

```
CREATE EXTERNAL TABLE IF NOT EXISTS src_jfs (key INT, value STRING) location 'jfs://your-namespace/Spark_sql_test/';
```

磁盘空间水位控制

JindoFS后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFS会自动淘汰本地较冷的数据备份。我们提供了**node.data-dirs.watermark.high.ratio**和**node.data-dirs.watermark.low.ratio**这两个参数用来调节本地存储的使用容量，值均为0～1的小数表示使用比例，JindoFS默认使用所有数据盘，每块盘的使用容量默认即为数据盘大小。前者表示使用量上水位比例，每块数据盘的JindoFS占用的空间到达上水位即会开始清理淘汰；后者表示使用量下水位比例，触发清理后会将JindoFS的占用空间清理到下水位。用户可以通过设置上水位比例调节期望分给JindoFS的磁盘空间，下水位必须小于上水位，设置合理的值即可。

存储策略

JindoFS提供了Storage Policy功能，提供更加灵活的存储策略适应不同的存储需求，可以对目录设置以下四种存储策略。

策略	策略说明
COLD	表示数据仅在OSS上有一个备份，没有本地备份，适用于冷数据存储。
WARM	默认策略。 表示数据在OSS和本地分别有一个备份， 本地备份能够有效的提供后续的读取加速。
HOT	表示数据在OSS上有一个备份，本地有多个备份，针对一些最热的数据提供更进一步的加速效果。
TEMP	表示数据仅有一个本地备份，针对一些临时性数据，提供高性能的读写，但降低了数据的高可靠性，适用于一些临时数据的存取。

JindoFS提供了Admin工具设置目录的Storage Policy（默认为 WARM），新增的文件将会以父目录所指定的Storage Policy进行存储，使用方式如下。

```
jindo dfsadmin -R -setStoragePolicy [path] [policy]
```

通过以下命令，获取某个目录的存储策略：

```
jindo dfsadmin -getStoragePolicy [path]
```

说明：

其中[path] 为设置policy的路径名称， -R表示递归设置该路径下的所有路径。

Admin工具

JindoFS提供了Admin工具的archive和jindo命令。

- Admin工具提供archive命令，实现对冷数据的归档。

  此命令提供了一种用户显式淘汰本地数据块的方式。Hive分区表按天分区，假如业务上对一周前的分区数据认为不会再经常访问，那么就可以定期将一周前的分区目录执行archive，淘汰本地备份，文件备份将仅仅保留在后端OSS上。

  Archive命令的使用方式如下。

  ```
  jindo dfsadmin -archive [path]
  ```

  说明：

  [path]为需要归档文件的所在目录路径。

- Admin工具提供jindo命令，为Namespace Service提供了一些管理员功能命令。

  ```
  jindo dfsadmin [-options]
  ```

  说明：

  可以通过jindo dfsadmin --help命令获取帮助信息。

Admin工具对Cache模式提供了diff和sync命令。

- diff命令主要用来显示本地数据与后端存储系统数据之间的差异。

  ```
  jindo dfsadmin -R -diff [path]
  ```

  说明：

  默认情况下比较[path]目录的子目录中元数据之间的差异，-R选项表示递归比较[path]目录下所有的路径。

- sync命令用于同步本地与后端存储之前的元数据。

```
jindo dfsadmin -R -sync [path]
```

> **说明：**
> [path]表示需要同步元数据的路径，默认只会同步[path]的下一级目录，-R选项表示递归比较[path]目录下所有的路径。

1.3 使用JindoFS SDK免密功能

本节介绍使用JindoFS SDK时，E-MapReduce（简称EMR）集群外如何以免密方式访问E-MapReduce JindoFS的文件系统。

前提条件

适用环境：ECS（EMR环境外）+Hadoop+JavaSDK。

背景信息

使用JindoFS SDK时，需要把环境中相关Jindo的包从环境中移除，如jboot.jar、smartdata-aliyun-jfs-*.jar。如果要使用Spark则需要把/opt/apps/spark-current/jars/里面的包也删除，从而可以正常使用。

步骤一：创建实例RAM角色

按以下步骤在访问控制RAM控制台创建一个实例RAM角色：

1. 使用云账号登录RAM的控制台。
2. 单击左侧导航栏的**RAM角色管理**。
3. 单击**创建 RAM 角色**，选择当前可信实体类型为**阿里云服务**。
4. 单击**下一步**。
5. 输入**角色名称**，从**选择授信服务**列表中，选择**云服务器**。
6. 单击**完成**。

步骤二：为RAM角色授予权限

按以下步骤在访问控制RAM控制台授权实例RAM角色一个系统权限或者自定义权限：

1. 使用云账号登录RAM的控制台。
2. （可选）如果您不使用系统权限，可以参见账号访问控制创建自定义权限策略章节创建一个自定义策略。
3. 单击左侧导航栏的**RAM角色管理**。

4. 单击新创建RAM角色名称所在行的**精确授权**。
5. 选择权限类型为**系统策略**或**自定义策略**。
6. 输入策略名称。
7. 单击**确定**。

步骤三：为实例授予RAM角色

按以下步骤在ECS控制台为一台ECS实例授予实例RAM角色：

1. 登录ECS管理控制台。
2. 在左侧导航栏，单击**实例与镜像** > **实例**。
3. 在顶部状态栏左上角处，选择地域。
4. 找到要操作的ECS实例，选择**更多** > **实例设置** > **授予/收回RAM角色**。

5. 在弹窗中，选择创建好的实例RAM角色，单击**确定**完成授予。

步骤四：在ECS上设置环境变量

按以下命令在ECS上设置环境变量：

```
export CLASSPATH=/xx/xx/jindofs-2.5.0-sdk.jar
```

或者

```
HADOOP_CLASSPATH=$HADOOP_CLASSPATH:/xx/xx/jindofs-2.5.0-sdk.jar
```

步骤五：测试免密方式访问的方法

1. 使用Shell访问OSS。

   ```
   hdfs dfs -ls/-mkdir/-put/..... oss://<ossPath>
   ```

2. 使用Hadoop FileSystem访问OSS。

   JindoFS SDK支持使用Hadoop FileSystem访问OSS，示例代码如下：

   ```
   import org.apache.hadoop.conf.Configuration;
   import org.apache.hadoop.fs.FileSystem;
   import org.apache.hadoop.fs.LocatedFileStatus;
   import org.apache.hadoop.fs.Path;
   import org.apache.hadoop.fs.RemoteIterator;

   import java.net.URI;

   public class test {
       public static void main(String[] args) throws Exception {
           FileSystem fs = FileSystem.get(new URI("ossPath"), new Configuration());
           RemoteIterator<LocatedFileStatus> iterator = fs.listFiles(new Path("ossPath"), false
   );
           while (iterator.hasNext()){
               LocatedFileStatus fileStatus = iterator.next();
               Path fullPath = fileStatus.getPath();
               System.out.println(fullPath);
           }
       }
   }
   ```

1.4 JindoFS块存储模式

本文主要介绍JindoFS的块存储模式（Block），以及一些典型的应用场景。

概念

块存储模式提供了最为高效的数据读写能力和元数据访问能力，并且能够支持更加全面的Hadoop文件系统语义。同时，JindoFS也提供了外部客户端，能够从集群外部访问建立在E-MapReduce集群内的JindoFS文件系统。

数据以Block形式存储在后端存储OSS上，本地Namespace服务维护元数据信息，该模式在性能上较优，无论是数据性能还是元数据性能。

应用场景

E-MapReduce目前提供了三种大数据存储系统，E-MapReduce OssFileSystem、E-MapReduce HDFS和E-MapReduce JindoFS，其中OssFileSystem和JindoFS都是云上存储的解决方案，下表为这三种存储系统和开源OSS各自的特点。

特点	开源OSS	E-MapReduce OssFileSystem	E-MapReduce HDFS	E-MapReduce JindoFS
存储空间	海量	海量	取决于集群规模	海量
可靠性	高	高	高	高
吞吐率因素	服务端	集群内磁盘缓存	集群内磁盘	集群内磁盘
元数据效率	慢	中	快	快
扩容操作	容易	容易	容易	容易
缩容操作	容易	容易	需Decommission	容易
数据本地化	无	弱	强	较强

JindoFS块存储模式具有以下几个特点：

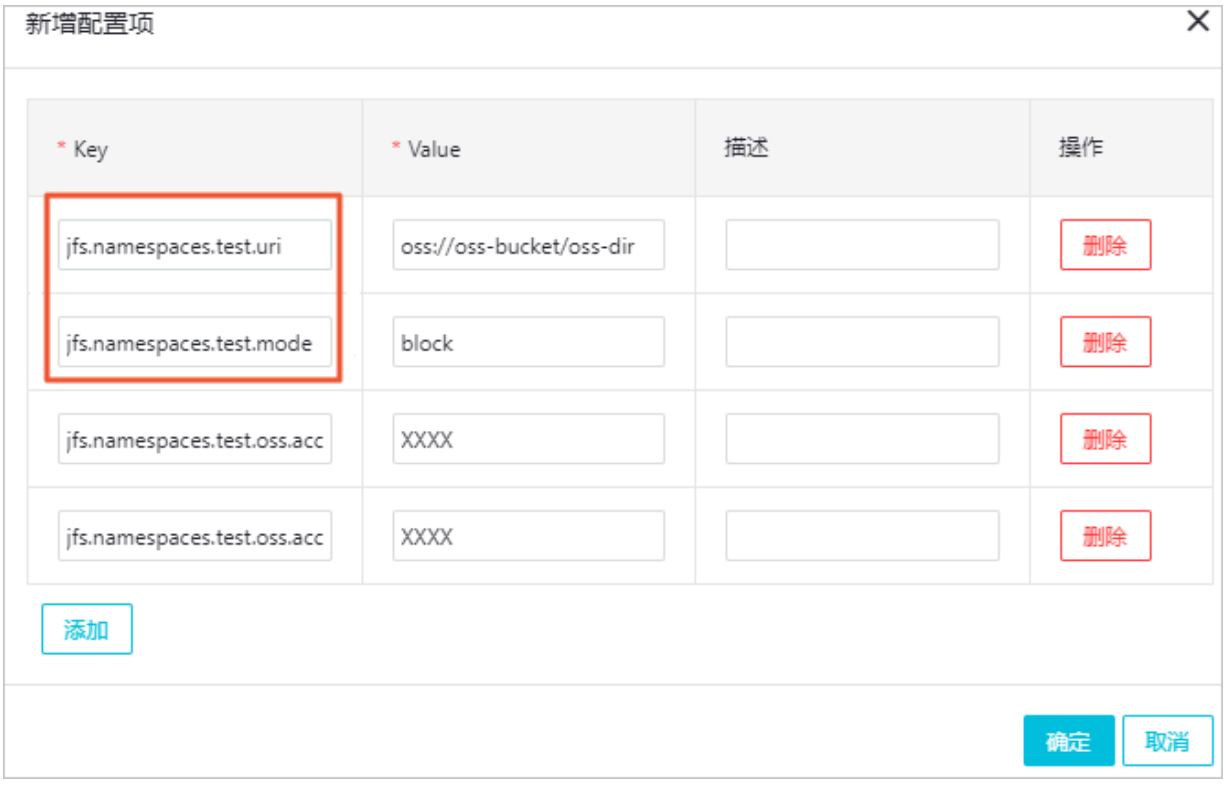
- 海量弹性的存储空间，基于OSS作为存储后端，存储不受限于本地集群，而且本地集群能够自由弹性伸缩。
- 能够利用本地集群的存储资源加速数据读取，适合具有一定本地存储能力的集群，能够利用有限的本地存储提升吞吐率，特别对于一写多读的场景效果显著。
- 元数据操作效率高，能够与HDFS相当，能够有效规避OSS文件系统元数据操作耗时以及高频访问下可能引发不稳定的问题。
- 能够最大限度保证执行作业时的数据本地化，减少网络传输的压力，进一步提升读取性能。

配置集群

所有JindoFS相关配置都在Bigboot组件中，配置如下图所示。

图 1-1: 修改配置项

图 1-2: 新增配置项



说明:

- 红框中为必填的配置项。
- JindoFS支持多命名空间，本文命名空间以test为例。

参数	参数说明	示例
jfs.namespaces	表示当前JindoFS支持的命名空间，多个命名空间时以逗号隔开。	test
jfs.namespaces.test.uri	表示test命名空间的后端存储。	oss://oss-bucket/oss-dir **说明：** 该配置也可以配置到OSS bucket下的具体目录，该命名空间即以该目录作为根目录来读写数据。
jfs.namespaces.test.mode	表示test命名空间为块存储模式。	block
jfs.namespaces.test.oss.access.key	表示存储后端OSS的AccessKey ID。	xxxx **说明：** 考虑到性能和稳定性，推荐使用同账户、同region下的OSS bucket作为存储后端，此时，E-MapReduce集群能够免密访问OSS，无需配置AccessKey ID和AccessKey Secret。
jfs.namespaces.test.oss.access.secret	表示存储后端OSS的AccessKey Secret。	

配置完成后保存并部署，然后在SmartData服务中重启Namespace Service，即可开始使用JindoFS。

存储策略

JindoFS提供了Storage Policy功能，提供更加灵活的存储策略适应不同的存储需求，可以对目录设置以下四种存储策略。

策略	策略说明
COLD	表示数据仅在OSS上有一个备份，没有本地备份，适用于冷数据存储。
WARM	默认策略。 表示数据在OSS和本地分别有一个备份， 本地备份能够有效的提供后续的读取加速。
HOT	表示数据在OSS上有一个备份，本地有多个备份，针对一些最热的数据提供更进一步的加速效果。
TEMP	表示数据仅有一个本地备份，针对一些临时性数据，提供高性能的读写，但降低了数据的高可靠性，适用于一些临时数据的存取。

JindoFS提供了Admin工具设置目录的Storage Policy（默认为 WARM），新增的文件将会以父目录所指定的Storage Policy进行存储，使用方式如下所示。

```
jindo dfsadmin -R -setStoragePolicy [path] [policy]
```

通过以下命令，获取某个目录的存储策略。

```
jindo dfsadmin -getStoragePolicy [path]
```

说明：
其中[path] 为设置policy的路径名称， -R表示递归设置该路径下的所有路径。

Admin工具还提供archive命令，实现对冷数据的归档。

此命令提供了一种用户显式淘汰本地数据块的方式。Hive分区表按天分区，假如业务上对一周前的分区数据认为不会再经常访问，那么就可以定期将一周前的分区目录执行archive，淘汰本地备份，文件备份将仅仅保留在后端OSS上。

Archive命令的使用方式如下：

```
jindo dfsadmin -archive [path]
```

说明：
[path]为需要归档文件的所在目录路径。

1.5 JindoFS缓存模式

本文主要介绍JindoFS的缓存模式（Cache），以及一些典型的应用场景。

概述

缓存模式兼容现有OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行数据和元数据的缓存，从而提高访问数据以及元数据的性能，Cache模式提供不同元数据同步策略以满足您在不同场景下的需求。

应用场景

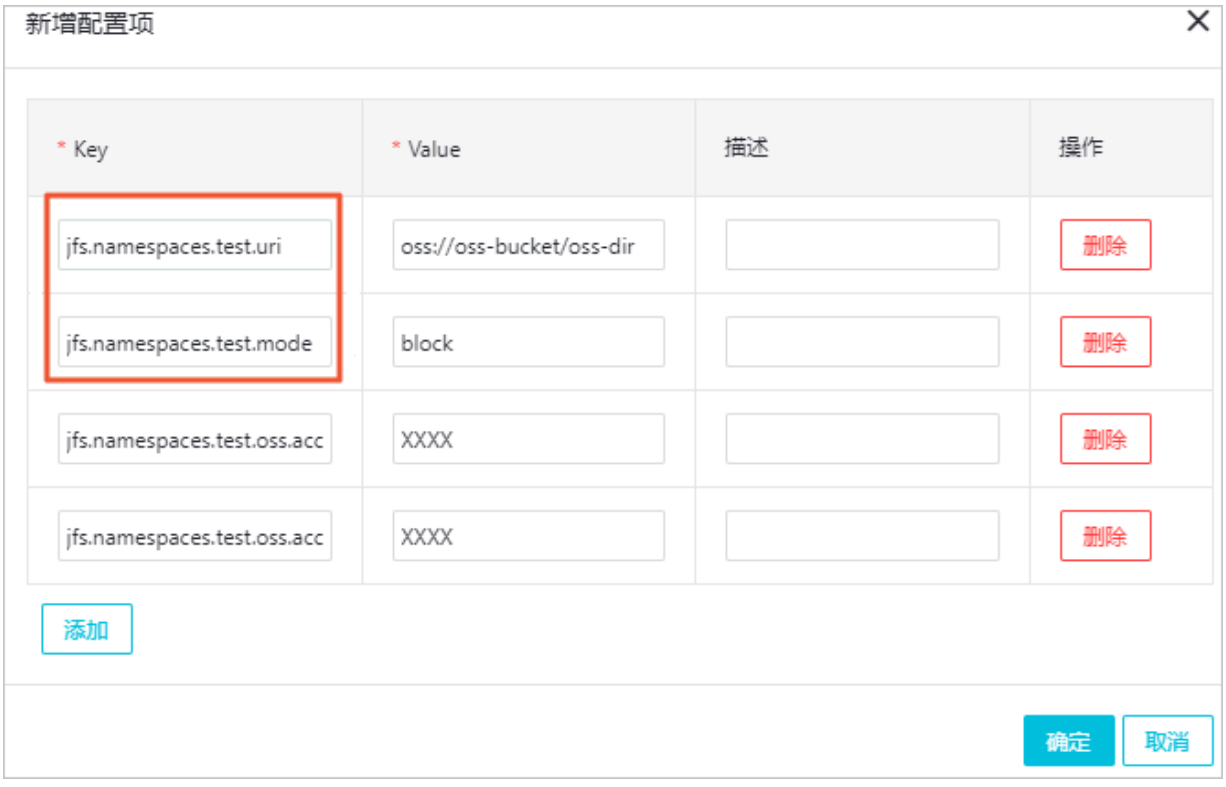
缓存模式最大的特点就是兼容性，保持了OSS原有的对象语义，集群中仅做缓存，因此JindoFS和OSS客户端、OssFileSystem等，或者其他的各种OSS的交互程序是完全兼容的，对原有OSS上的存量数据也不需要做任何的迁移、转换工作即可使用。同时集群中的数据和元数据缓存也能一定程度上提升数据访问性能。

配置集群

所有JindoFS相关配置都在Bigboot组件中，配置如下图所示。

图 1-3: 修改配置项

图 1-4: 新增配置项



说明：

- 红框中为必填的配置项。
- JindoFS支持多命名空间，本文命名空间以test为例。

参数	参数说明	示例
jfs.namespaces	表示当前JindoFS支持的命名空间，多个命名空间时以逗号隔开。	test
jfs.namespaces.test.uri	表示test命名空间的后端存储。	oss://oss-bucket/ **说明：** 该配置也可以配置到OSS bucket下的具体目录，该命名空间即以该目录作为根目录来读写数据，但一般情况下配置bucket即可，这样路径就和原生OSS保持一致。
jfs.namespaces.test.mode	表示test命名空间为缓存模式。	cache
jfs.namespaces.test.oss.access.key	表示存储后端OSS的AccessKey ID。	xxxx **说明：** 考虑到性能和稳定性，推荐使用同账户、同region下的OSS bucket作为存储后端，此时，E-MapReduce集群能够免密访问OSS，无需配置AccessKey ID和AccessKey Secret。
jfs.namespaces.test.oss.access.secret	表示存储后端OSS的AccessKey Secret。	

配置完成后保存并部署，然后在SmartData服务中重启Namespace Service，即可开始使用JindoFS。

元数据同步策略

缓存模式下可能存在JindoFS集群构建之前，您已经在OSS上保存了大量数据的场景，对于这种场景，后续的数据访问会同步数据和元数据到JindoFS集群，数据同步策略为了访问数据都会在本地保留一份；元数据同步策略分为两部分，包括元数据同步间隔策略和元数据load策略：

- 元数据同步间隔策略：

  配置参数为**namespace.sync.interval**， 该参数默认值为**-1**， 表示不会同步OSS上的元数据。

  - 当**namespace.sync.interval=0**时，表示每次操作都会同步OSS上的元数据。
  - 当**namespace.sync.interval>0**时，表示会以固定的时间间隔来同步OSS上的元数据。

    说明：
    例如当**namespace.sync.interval=5**时， 表示每隔5秒会去同步OSS上的元数据。

- 元数据Load策略：

  配置参数为**namespace.sync.loadtype**，该配置参数为枚举类型**{never, once, always}**，**never**表示从不同步OSS上的元数据；**once**为默认配置，表示只从OSS同步一次元数据；**always**表示每次操作都会同步OSS上的元数据。

  说明：
  当不配置**namespace.sync.interval**参数时，才会去使用Load策略；如果已配置**namespace.sync.interval**参数，则Load策略配置不生效。

1.6 JindoFS外部客户端

本文主要介绍JindoFS的外部客户端，以及一些典型的应用场景。

概述

JindoFS外部客户端，主要是为E-MapReduce集群外部访问JindoFS集群提供一种可行的方法。现在JindoFS外部客户端只能访问块存储模式下的JindoFS，不支持访问缓存模式下的JindoFS。实际上，缓存模式兼容OSS原始语义，因此外部访问仅需用普通OSS客户端即可。

应用场景

JindoFS外部客户端实现了Hadoop文件系统的接口，在用户程序跟E-MapReduce JindoFS Namespace服务网络相通的情况下， 用户可以通过JindoFS外部客户端去访问JindoFS上存储的数据， 但外部客户端不能利用E-MapReduce JindoFS的数据缓存能力，相比E-MapReduce集群内部访问JindoFS集群，性能有所损失。

配置外部客户端

已配置JindoFS块存储模式的Namespace，详情请参见JindoFS块存储模式。

1. 获取Bigboot程序包。

   在E-MapReduce集群内部/usr/lib/bigboot-current路径下，获取Bigboot程序包。

   > **说明：**
   > 一般情况下，程序使用Native开发，若实际系统类型与E-MapReduce集群差别较大，相关的程序需重新编译，可以通过联系我们处理。

2. 配置环境。

   设置环境变量**BIGBOOT_HOME**为程序安装根目录，将程序根目录下ext和lib的路径，添加到用户使用的大数据组件（Hadoop或Spark等）的**Classpath**中。

3. 从E-MapReduce集群内部拷贝配置文件/usr/lib/bigboot-current/conf/bigboot.cfg.external，到用户客户机上对应的安装目录conf/bigboot.cfg。

4. 配置Namespace Service。

   - client.namespace.rpc.port：配置JindoFS Namespace Service的监听端口。
   - client.namespace.rpc.address：配置JindoFS Namespace Service的监听地址。

     > **说明：**
     > 默认E-MapReduce集群中的配置文件已经配置好这两项。

5. 配置数据访问相关的配置项。

   - client.namespaces.{YourNamespace}.oss.access.bucket：配置OSS bucket选项。
   - client.namespaces.{YourNamespace}.oss.access.endpoint：配置OSS endpoint选项。
   - client.namespaces.{YourNamespace}.oss.access.key：配置OSS的AccessKey ID。
   - client.namespaces.{YourNamespace}.oss.access.secret：配置OSS的AccessKey Secret。

     > **说明：**
     > 其中{YourNamespace}为外部客户端要访问的Namespace的名称，本文Namespace的名称以test为例。

     配置示例如下。

     ```
     client.namespace.rpc.port = 8101
     client.namespace.rpc.address = {RPC_Address}
     client.namespaces.test.oss.access.bucket = {YourOssBucket}
     client.namespaces.test.oss.access.endpoint = {YourOssEndpoint}
     ```

```
client.namespaces.test.oss.access.key = {YourOssAccessKeyID}
client.namespaces.test.oss.access.secret = {YourOssAccessKeySecret}
```

配置验证

验证如下信息：

- 通过以下命令，验证Namespace是否正确。

```
hdfs dfs -ls jfs://test/
```

- 通过以下命令，验证数据是否可以上传或者下载。

```
hdfs dfs -put /etc/hosts  jfs://test/

hdfs dfs -get jfs://test/hosts
```

2 JindoFS基础使用（EMR-3.27.0及以上版本）

2.1 SmartData 2.6.x版本简介

从EMR 3.26.3版本开始，支持SmartData的2.6.x版本，包含多个重大特性的发布以及大幅的性能优化，包括Namespace服务后端存储支持Tablestore（OTS）以及Raft、Namespace服务支持HA、块存储模式和缓存模式使用方式优化和读写性能优化等。

元数据服务后端存储方案升级

在原有RocksDB方案的基础上，新版本推出了Tablestore以及Raft的后端存储方案可供选择，实现元数据上云，加上数据块本身已经通过OSS持久化，从而整个Jindo存储实现了一种完全云原生的解决方案，各个模块都不再强依赖于集群内部的状态，用户集群可以随时销毁，重建之后能够通过维护在云上的数据快速地恢复。同时，基于Tablestore或者Raft的方案，实现了元数据服务的高可用，通过多个Namespace服务提供HA方案。针对默认方案RocksDB，特别是使用Cache模式等对于元数据存储以及HA没有很高要求的场景下，RocksDB依然是一种简单、实用而且高效的方案。

各方案详情请参见：

- 使用Tablestore作为存储后端
- 使用Raft-RocksDB-Tablestore作为存储后端
- 使用RocksDB作为元数据后端

使用模式优化

使用模式支持块存储模式和缓存模式两种。

- 块存储模式（Block）使用方式与EMR 3.26.3之前版本基本没有变化，详情请参见JindoFS块存储模式使用说明。
- 缓存模式（Cache ）支持了多种使用方式，既支持与Block模式一致的使用方式，也支持了原有OSS文件系统的使用方式，以满足用户不同的需要，详情请参见JindoFS缓存模式使用说明。

支持权限

Block模式支持文件系统权限功能。没有权限的用户将禁止读（写）文件。包含Unix权限和Ranger权限两种使用方式。使用Unix权限，可以使用文件的777权限进行管理。使用Ranger权限，可以利用Ranger路径通配符等高级配置进行权限管理。权限功能详细请参见JindoFS权限功能。

2.2 JindoFS块存储模式使用说明

块存储模式（Block）提供了最为高效的数据读写能力和元数据访问能力。数据以Block形式存储在后端存储OSS上，本地提供缓存加速，元数据则由本地Namespace服务维护，提供高效的元数据访问性能。本文主要介绍JindoFS的块存储模式及其使用方式。

背景信息

JindoFS块存储模式具有以下几个特点：

- 海量弹性的存储空间，基于OSS作为存储后端，存储不受限于本地集群，而且本地集群能够自由弹性伸缩。
- 能够利用本地集群的存储资源加速数据读取，适合具有一定本地存储能力的集群，能够利用有限的本地存储提升吞吐率，特别对于一写多读的场景效果显著。
- 元数据操作效率高，能够与HDFS相当，能够有效规避OSS文件系统元数据操作耗时以及高频访问下可能引发不稳定的问题。
- 能够最大限度保证执行作业时的数据本地化，减少网络传输的压力，进一步提升读取性能。

配置使用方式

1. 进入SmartData服务。
   a) 登录阿里云E-MapReduce控制台。
   b) 在顶部菜单栏处，根据实际情况选择地域（Region）和资源组。
   c) 单击上方的**集群管理**页签。
   d) 在**集群管理**页面，单击相应集群所在行的**详情**。
   e) 在左侧导航栏单击**集群服务** > **SmartData**。
2. 进入bigboot服务配置。
   a) 单击**配置**页签。
   b) 单击**bigboot**。

3. 配置以下参数。

   JindoFS支持多命名空间，本文命名空间以test为例。

   a) 修改**jfs.namespaces**为**test**。

      test表示当前JindoFS支持的命名空间，多个命名空间时以逗号隔开。

   b) 单击**自定义配置**，在**新增配置项**对话框中增加以下参数，单击**确定**。

参数	参数说明	示例
jfs.namespaces.test.oss.uri	表示test命名空间的后端存储。	oss://<oss_bucket>/<oss_dir>/ **说明：** 推荐配置到OSS bucket下的某一个具体目录，该命名空间即会将Block模式的数据块存放在该目录下。
jfs.namespaces.test.mode	表示test命名空间为块存储模式。	block
jfs.namespaces.test.oss.access.key	表示存储后端OSS的AccessKey ID。	xxxx **说明：** 考虑到性能和稳定性，推荐使用同账户、同Region下的OSS bucket作为存储后端，此时，E-MapReduce集群能够免密访问OSS，无需配置AccessKey ID和AccessKey Secret。
jfs.namespaces.test.oss.access.secret	表示存储后端OSS的AccessKey Secret。	

   c) 单击**确定**。

4. 单击右上角的**保存**。

5. 单击右上角的**操作** > **重启 Jindo Namespace Service**。

   重启后即可通过 jfs://test/<path_of_file>的形式访问JindoFS上的文件。

磁盘空间水位控制

JindoFS后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFS会自动淘汰本地较冷的数据备份。我们提供了storage.watermark.high.ratio和storage.watermark.low.ratio两个参数来调节本地存储的使用容量，值均为0～1的小数，表示使用磁盘空间的比例。

1. 修改磁盘水位配置。

   可在**服务配置**区域的**storage**页签，修改以下参数。

参数	描述
storage.watermark.high.ratio	表示磁盘使用量的上水位比例，每块数据盘的JindoFS数据目录占用的磁盘空间到达上水位即会触发清理。默认值：0.4。
storage.watermark.low.ratio	表示使用量的下水位比例，触发清理后会自动清理冷数据，将JindoFS数据目录占用空间清理到下水位。默认值：0.2。

   说明：
   您可以通过设置上水位比例调节期望分给JindoFS的磁盘空间，下水位必须小于上水位，设置合理的值即可。

2. 保存配置。

   a. 单击右上角的**保存**。

   b. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。

   c. 单击**确定**。

3. 重启Jindo Storage Service使配置生效。

   a. 单击右上角的**操作** > **重启Jindo Storage Service**。

   b. 在**执行集群操作**对话框中，设置相关参数。

   c. 单击**确定**。

   d. 在**确认**对话框中，单击**确定**。

2.3 JindoFS缓存模式使用说明

缓存模式（Cache）主要兼容原生OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行缓存，提升EMR集群内访问OSS的效率，同时兼容了原有OSS原有文件形式，数据访问上能够与其他OSS客户端完全兼容。本文主要介绍JindoFS的缓存模式及其使用方式。

背景信息

缓存模式最大的特点就是兼容性，保持了OSS原有的对象语义，集群中仅做缓存，因此和其他的各种OSS客户端是完全兼容的，对原有OSS上的存量数据也不需要做任何的迁移、转换工作即可使用。同时集群中的缓存也能一定程度上提升数据访问性能，缓解读写 OSS 的带宽压力。

配置使用方式

JindoFS缓存模式提供了以下两种基本使用方式，以满足不同的使用需求。

- OSS Scheme

  详情请参见配置OSS Scheme（推荐）。

- JFS Scheme

  详情请参见配置JFS Scheme。

配置OSS Scheme（推荐）

OSS Scheme保留了原有 OSS 文件系统的使用习惯，即直接通过oss://<bucket_name>/<path_of_your_file>的形式访问OSS上的文件。使用该方式访问OSS，无需进行额外的配置，创建EMR集群后即可使用，对于原有读写OSS的作业也无需做任何修改即可运行。

配置JFS Scheme

1. 进入SmartData服务。

   a)

   b)

   c)

   d)

   e) 在左侧导航栏单击**集群服务** > **SmartData**。

2. 进入bigboot服务配置。

   a) 单击**配置**页签。

   b) 单击**bigboot**。

3. 配置以下参数。

   JindoFS支持多命名空间，本文命名空间以test为例。

   a) 修改**jfs.namespaces**为**test**。

      test表示当前JindoFS支持的命名空间，多个命名空间时以逗号隔开。

   b) 单击**自定义配置**，在**新增配置项**对话框中增加以下参数。

参数	参数说明	示例
jfs.namespaces.test.oss.uri	表示test命名空间的后端存储。	oss://<oss_bucket>/<oss_dir>/ **说明：** 该配置必须配置到OSS bucket下的具体目录，也可以直接使用根目录。
jfs.namespaces.test.mode	表示test命名空间为缓存模式。	cache

4. 单击右上角的**保存**。
5. 单击右上角的**操作** > **重启 Jindo Namespace Service**。

   重启后即可通过jfs://test/<path_to_your_file>的形式访问，该命名空间下的文件会以**jfs.namespaces.test.oss.uri**所配置的目录作为根目录进行组织，例如jfs://test/hello.txt对应实际OSS上的文件为oss://<oss_bucket>/<oss_dir>/hello.txt。

启用缓存

启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为禁用，即所有OSS读取都直接访问OSS上的数据。

1. 在**集群服务** > **SmartData**的**配置**页面，单击**client**页签。
2. 修改**jfs.cache.data-cache.enable**为**1**，表示启用缓存。

   此配置为客户端配置，不需要重启SmartData服务。

缓存启用后，Jindo服务会自动管理本地缓存备份，通过水位清理本地缓存，请您根据需求配置一定的比例用于缓存，详情请参见磁盘空间水位控制。

磁盘空间水位控制

JindoFS后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFS会自动淘汰本地较冷的数据备份。我们提供了storage.watermark.high.ratio和storage.watermark.low.ratio两个参数来调节本地存储的使用容量，值均为0 ~ 1的小数，表示使用磁盘空间的比例。

1. 修改磁盘水位配置。

   可在**服务配置**区域的**storage**页签，修改以下参数。

参数	描述
storage.watermark.high.ratio	表示磁盘使用量的上水位比例，每块数据盘的JindoFS数据目录占用的磁盘空间到达上水位即会触发清理。默认值：0.4。
storage.watermark.low.ratio	表示使用量的下水位比例，触发清理后会自动清理冷数据，将JindoFS数据目录占用空间清理到下水位。默认值：0.2。

   说明：
   您可以通过设置上水位比例调节期望分给JindoFS的磁盘空间，下水位必须小于上水位，设置合理的值即可。

2. 保存配置。

   a. 单击右上角的**保存**。

   b. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。

   c. 单击**确定**。

3. 重启Jindo Storage Service使配置生效。

   a. 单击右上角的**操作** > **重启Jindo Storage Service**。

   b. 在**执行集群操作**对话框中，设置相关参数。

   c. 单击**确定**。

   d. 在**确认**对话框中，单击**确定**。

访问OSS bucket

在EMR集群中访问同账号、同区域的OSS bucket时，默认支持免密访问，即无需配置任何AccessKey即可访问。如果访问非以上情况的OSS bucket需要配置相应的AccessKey ID、AccessKey Secret以及Endpoint，针对两种使用方式相应的配置分别如下：

- OSS Scheme

  1. 在**集群服务** > **SmartData**的**配置**页面，单击**smartdata-site**页签。
  2. 单击**自定义配置**，在**新增配置项**对话框中增加以下参数，单击**确定**。

参数	参数说明
fs.jfs.cache.oss-accessKeyId	表示存储后端OSS的AccessKey ID。
fs.jfs.cache.oss-accessKeySecret	表示存储后端OSS的AccessKey Secret。
fs.jfs.cache.oss-endpoint	表示存储后端OSS的endpoint。

- JFS Scheme

  1. 在**集群服务** > **SmartData**的**配置**页面，单击**bigboot**页签。
  2. 修改**jfs.namespaces**为**test**。
  3. 单击**自定义配置**，在**新增配置项**对话框中增加以下参数，单击**确定**。

参数	参数说明
jfs.namespaces.test.oss.uri	表示test命名空间的后端存储。示例：oss://<oss_bucket.endpoint>/<oss_dir>。 endpoint信息直接配置在oss.uri中。
jfs.namespaces.test.oss.access.key	表示存储后端OSS的AccessKey ID。
jfs.namespaces.test.oss.access.secret	表示存储后端OSS的AccessKey Secret。

高级配置

Cache模式还包含一些高级配置，用于性能调优，以下配置均为客户端配置，修改后无需重启SmartData服务。

- 在**服务配置**区域的**client**页签，配置以下参数。

参数	参数说明
client.oss.upload.threads	每个文件写入流的OSS上传线程数。默认值：4。
client.oss.upload.max.parallelism	进程级别OSS上传总并发度上限，防止过多上传线程造成过大的带宽压力以及过大的内存消耗。默认值：16。

- 在**服务配置**区域的**smartdata-site**页签，配置以下参数。

参数	参数说明
fs.jfs.cache.copy.simple.max.byte	rename过程使用普通copy接口的文件大小上限（小于阈值的使用普通 copy接口，大于阈值的使用multipart copy接口以提高copy效率）。 **说明：** 如果确认已开通OSS fast copy功能，参数值设为**-1**，表示所有大小均使用普通copy接口，从而有效利用fast copy获得最优的rename性能。
fs.jfs.cache.write.buffer.size	文件写入流的buffer大小，参数值必须为2的幂次，最大为8MB，如果作业同时打开的写入流较多导致内存使用过大，可以适当调小此参数。默认值：1048576。
fs.oss.committer.magic.enabled	启用Jindo Job Committer，避免Job Committer的rename操作，来提升性能。默认值：true。 **说明：** 针对Cache模式下，由于OSS这类对象存储rename操作性能较差的问题，推出了Jindo Job Committer。

2.4 JindoFS元数据服务

2.4.1 使用Tablestore作为存储后端

JindoFS元数据服务支持不同的存储后端，本文介绍使用Tablestore（OTS）作为元数据后端时需要进行的配置。

前提条件

- 已创建EMR集群。

  详情请参见#unique_6。

- 已创建Tablestore实例，推荐使用高性能实例。

  详情请参见#unique_21。

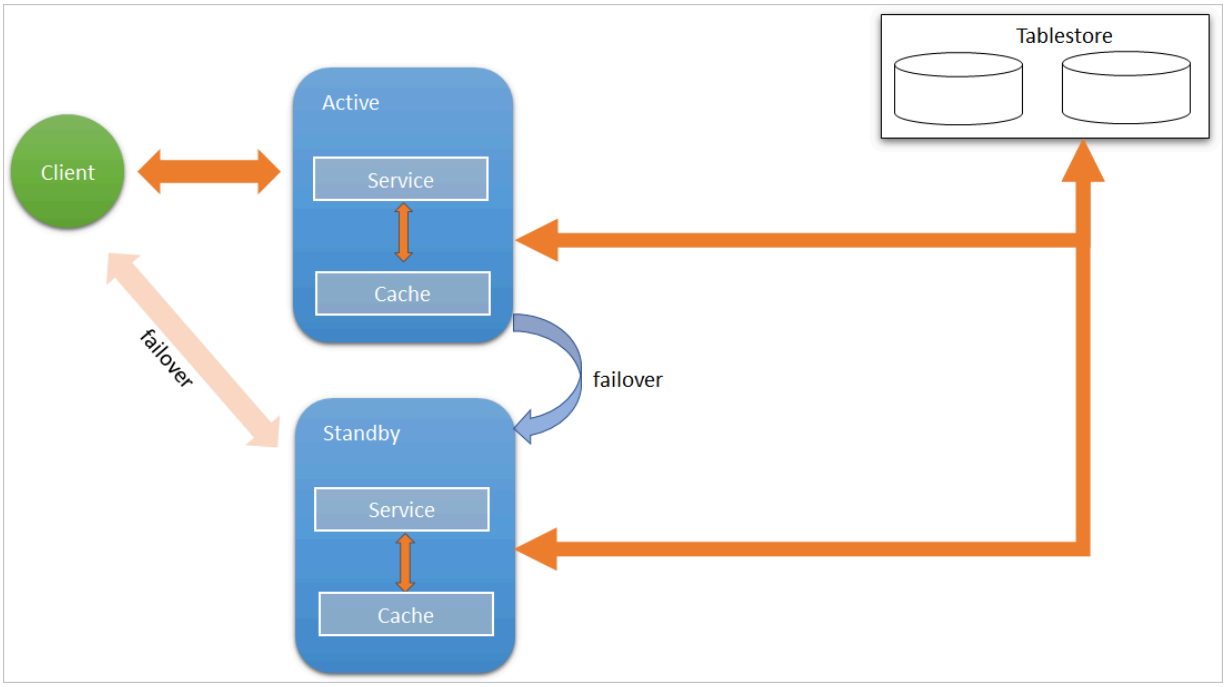
说明：

需要开启事务功能。

背景信息

JindoFS在新版本中，支持使用Tablestore作为JindoFS元数据服务（Namespace Service）的存储。一个EMR JindoFS集群可以绑定一个Tablestore实例（Instance）作为JindoFS元数据服务的存储介质，元数据服务会自动为每个Namespace创建独立的Tablestore表进行管理和存储元数据信息。

元数据服务（双机Tablestore和HA）架构图如下所示。



配置Tablestore

使用Tablestore功能，需要把创建的Tablestore实例和JindoFS的Namespace服务进行绑定，详细步骤如下：

1. 进入SmartData服务。

   a)

   b)

   c)

   d)

   e) 在左侧导航栏单击**集群服务** > **SmartData**。

2. 进入bigboot服务配置。

   a) 单击**配置**页签。

   b) 单击**bigboot**。

3. 配置以下参数。

   例如，在华东1（杭州）地域下，创建了emr-jfs的Tablestore实例，EMR集群使用VPC网络，访问Tablestore的AccessKey ID为kkkkkk，Access Secret为XXXXXX。

参数	参数说明	示例
namespace.backend.type	设置namespace后端存储类型，支持： • rocksdb • ots • raft 默认为rocksdb。	ots
namespace.ots.instance	Tablestore实例名称。	emr-jfs
namespace.ots.accessKey	Tablestore实例的AccessKey ID。	kkkkkk
namespace.ots.accessSecret	Tablestore实例的AccessKey Secret。	XXXXXX
namespace.ots.endpoint	Tablestore实例的Endpoint地址，普通EMR集群，推荐使用VPC地址。	http://emr-jfs.cn-hangzhou.vpc.tablestore.aliyuncs.com

4. 保存配置。

   a) 单击右上角的**保存**。

   b) 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。

   c) 单击**确定**。

5. 单击右上角的**操作** > **重启 Jindo Namespace Service**。

配置免密功能

1. 配置以下参数。

   在EMR集群中已实现免密访问Tablestore功能，使用免密功能的情况下您不需要指定**namespaces.ots.accessKey**和**namespace.ots.accessSecret**。需要指定的参数列表如下。

参数	参数说明	示例
namespace.ots.instance	Tablestore实例名称。	emr-jfs
namespace.ots.endpoint	Tablestore实例的Endpoint地址，普通EMR集群内，推荐使用VPC地址。	http://emr-jfs.cn-hangzhou.vpc.tablestore.aliyuncs.com
namespace.backend.type	设置namespace后端存储类型，支持rocksdb、ots和raft，默认为rocksdb，需要设置为ots。	ots

2. 保存配置。

   a) 单击右上角的**保存**。

   b) 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。

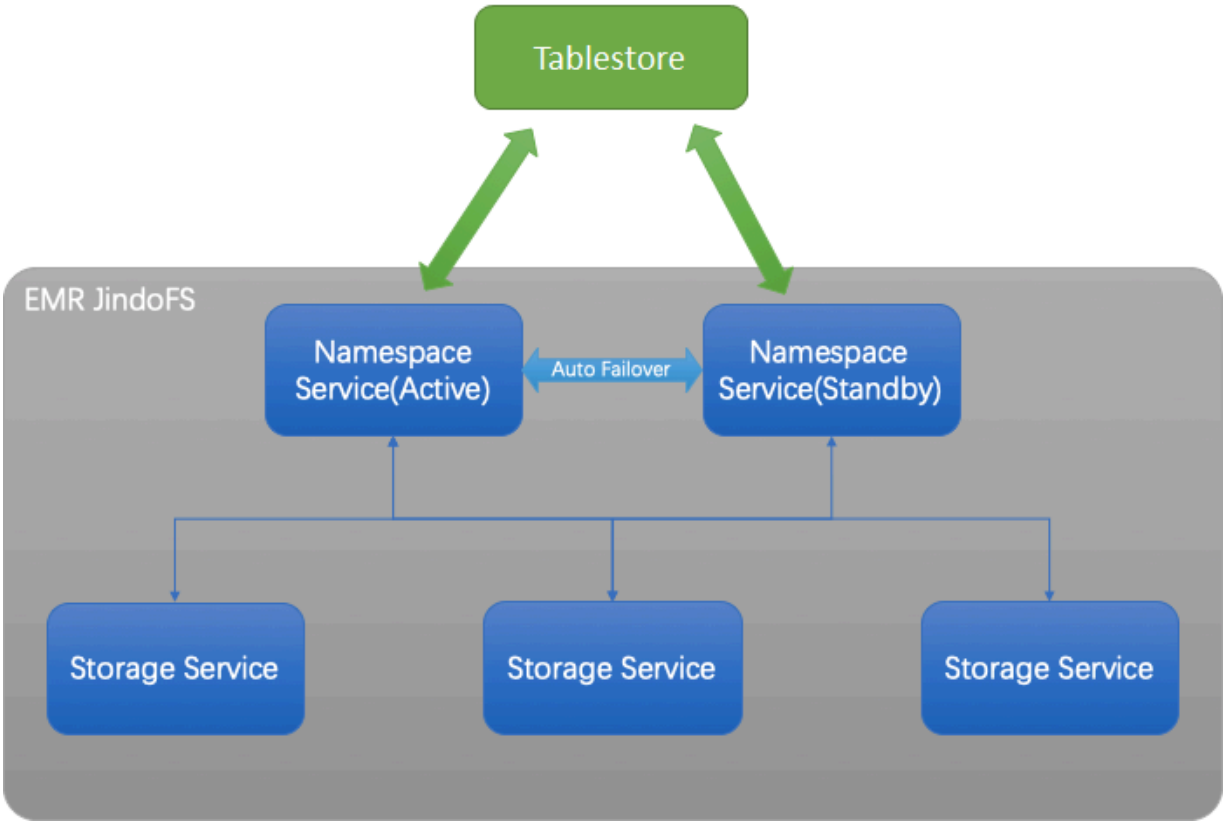
   c) 单击**确定**。

3. 单击右上角的**操作** > **重启 Jindo Namespace Service**。

配置Tablestore（高可用方案）

针对EMR的高可用集群，可以通过配置开启Namespace高可用模式。

Namespace高可用模式采用Active和Standby互备方式，支持自动故障转移，当Active Namespace出现异常或者挂掉时，客户端可以请求自动切换到新的Active节点。



1. 进入SmartData的bigboot服务配置，配置以下参数。
   a) 修改**jfs.namespace.server.rpc-address**值为**emr-header-1:8101,emr-header-2:8101**。
   b) 单击右上角的**自定义配置**，添加**namespace.backend.ots.ha**为**true**。
   c) 单击**确定**。
   d) 保存配置。
      A. 单击右上角的**保存**。
      B. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。
      C. 单击**确定**。
2. 单击右上角的**操作** > **重启Jindo Namespace Service**。
3. 单击右上角的**操作** > **重启Jindo Storage Service**。

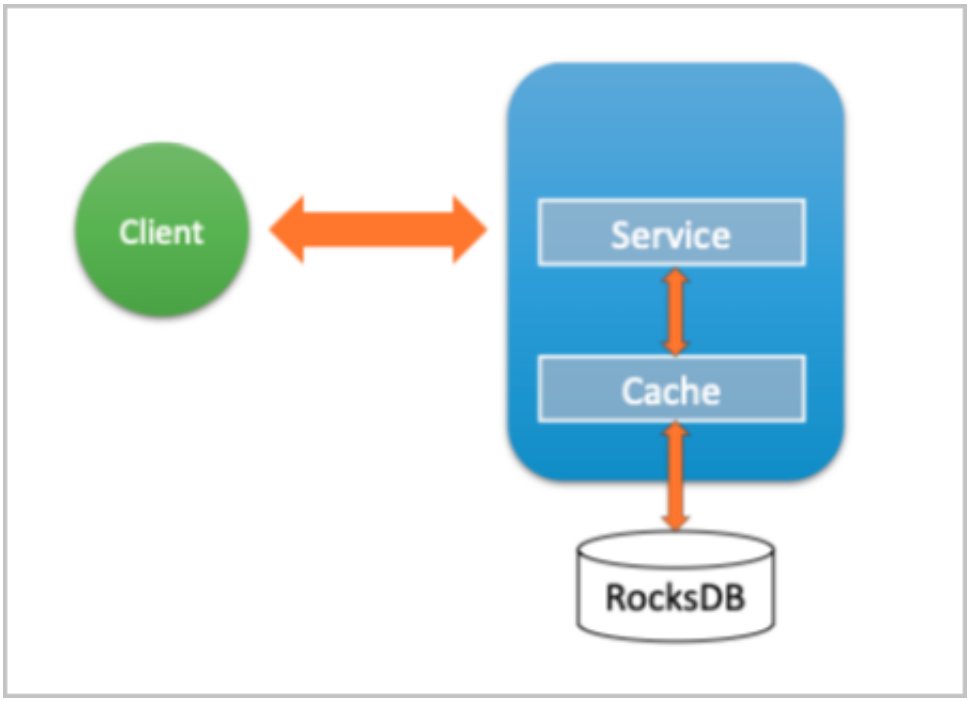
2.4.2 使用RocksDB作为元数据后端

JindoFS元数据服务支持不同的存储后端，默认配置RocksDB为元数据存储后端。本文介绍使用RocksDB作为元数据后端时需要进行的相关配置。

背景信息

RocksDB作为元数据后端时不支持高可用，如果需要高可用，推荐配置Tablestore（OTS）或者Raft作为元数据后端，详情请参见使用Tablestore作为存储后端和使用Raft-RocksDB-Tablestore作为存储后端。

单机RocksDB作为元数据服务如下图所示。



配置RocksDB

1. 进入SmartData服务。
   a)
   b)
   c)
   d)
   e) 在左侧导航栏单击**集群服务** > **SmartData**。
2. 进入bigboot服务配置。
   a) 单击**配置**页签。
   b) 单击**bigboot**。

3. 设置**namespace.backend.type**为**rocksdb**。
4. 保存配置。
   a) 单击右上角的**保存**。
   b) 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。
   c) 单击**确定**。
5. 单击右上角的**操作** > **重启 Jindo Namespace Service**。

2.4.3 使用Raft-RocksDB-Tablestore作为存储后端

JindoFS在EMR-3.27.0及之后版本中支持使用Raft-RocksDB-OTS作为Jindo元数据服务（Namespace Service）的存储。1个EMR JindoFS集群创建3个Master节点组成1个Raft实例，实例的每个Peer节点使用本地RocksDB存储元数据信息。

前提条件

- 创建Tablestore实例，推荐使用高性能实例，详情请参见#unique_21。

  说明:
  需要开启事务功能。

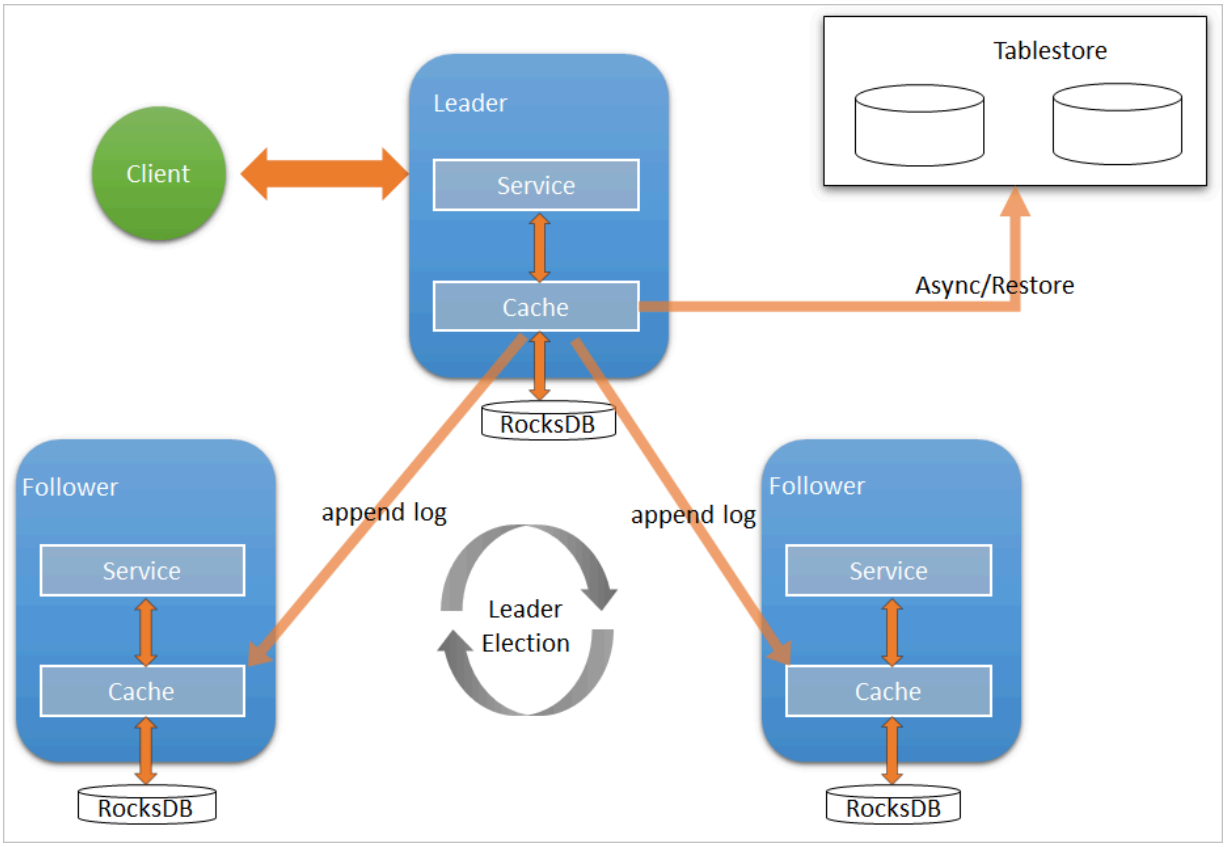
- 创建3 Master的EMR集群，详情请参见#unique_6。

  说明:
  若无部署方式，请提交工单联系E-MapReduce产品团队授予设置部署方式的权限。

背景信息

RocksDB通过Raft协议实现3个节点之间的复制。集群可以绑定1个Tablestore（OTS）实例，作为Jindo的元数据服务的额外存储介质，本地的元数据信息会实时异步地同步到用户的Tablestore实例上。

元数据服务-多机Raft-RocksDB-Tablestore+HA如下图所示。



配置本地raft后端

1. 新建EMR集群后，暂停SmartData所有服务。

   a) 登录阿里云E-MapReduce控制台。

   b) 在顶部菜单栏处，根据实际情况选择地域（Region）和资源组。

   c) 单击上方的**集群管理**页签。

   d) 在**集群管理**页面，单击相应集群所在行的**详情**。

   e) 单击右上角的**操作** > **停止 All Components**。

2. 根据使用需求，添加需要的namespace。

   配置详情可参见JindoFS块存储模式使用说明。

3. 进入SmartData服务的**bigboot**页签。

   a) 在左侧导航栏单击**集群服务** > **SmartData**。

   b) 单击**配置**页签。

   c) 在**服务配置**区域，单击**bigboot**页签。

4. 在SmartData服务的**bigboot**页签，设置以下参数。

参数	描述	示例
namespace.backend.type	设置namespace后端存储类型，支持： • rocksdb • ots • raft 默认为rocksdb。	raft
namespace.backend.raft.initconf	部署raft实例的3个Master地址（固定值）。	emr-header-1:8103:0,emr-header-2:8103:0,emr-header-3:8103:0
jfs.namespace.server.rpc-address	Client端访问raft实例的3个Master地址（固定值）	emr-header-1:8101,emr-header-2:8101,emr-header-3:8101

说明：

如果不需要使用OTS远端存储，直接执行步骤6和步骤7；如果需要使用OTS远端存储，请执行步骤5~步骤7。

5. （可选）配置远端OTS异步存储。

在SmartData服务的**bigboot**页签，设置以下参数。

参数	参数说明	示例
namespace.ots.instance	Tablestore实例名称。	emr-jfs
namespace.ots.accessKey	Tablestore实例的AccessKey ID。	kkkkkk
namespace.ots.accessSecret	Tablestore实例的AccessKey Secret。	XXXXXX
namespace.ots.endpoint	Tablestore实例的endpoint地址，通常EMR集群，推荐使用VPC地址。	http://emr-jfs.cn-hangzhou.vpc.tablestore.aliyuncs.com

参数	参数说明	示例
namespace.backend.raft.asyncOts.enable	是否开启OTS异步上传，包括： • true • false 当设置为true时，需要在SmartData服务完成初始化前，开启OTS异步上传功能。 **说明：** 如果SmartData服务已完成初始化，则不能再开启该功能。因为OTS的数据已经落后于本地RocksDB的数据。	true

6. 保存配置。

   a) 单击右上角的**保存**。

   b) 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。

   c) 单击**确定**。

7. 单击右上角的**操作** > **启动 All Components**。

从Tablestore恢复元数据信息

如果您在原始集群开启了远端Tablestore异步存储，则Tablestore上会有1份完整的JindoFS元数据的副本。您可以在停止或释放原始集群后，在新创建的集群上恢复原先的元数据，从而继续访问之前保存的文件。

1. （可选）准备工作。

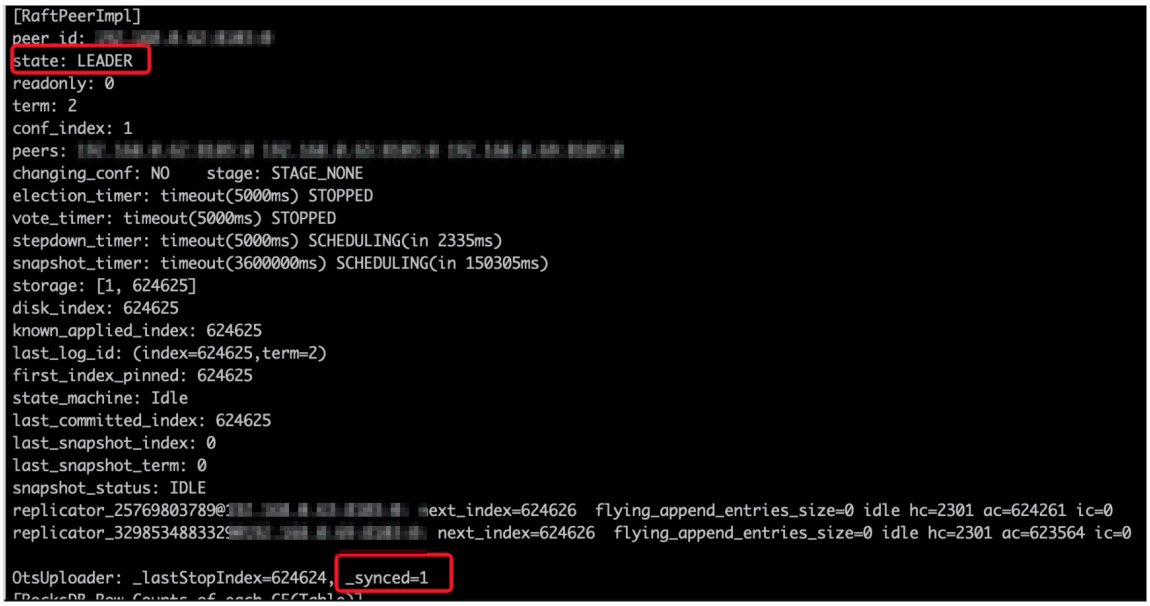
   a) （可选）统计原始集群的元数据信息（文件和文件夹数量）。

```
[hadoop@emr-header-1 ~]$ hadoop fs -count jfs://test/
       1596      1482809                 25 jfs://test/
```

（文件夹个数）（文件个数）

b) 停止原始集群的作业，等待30~120秒左右，等待原始集群的数据已经完全同步到Tablestore。执行以下命令查看状态。如果LEADER节点显示_synced=1，则表示Tablestore为最新数据，同步完成。

```
jindo jfs -metaStatus -detail
```



c) 停止或释放原始集群，确保没有其它集群正在访问当前的Tablestore实例。

2. 创建新集群。

   新建与Tablestore实例相同Region的EMR集群，暂停SmartData所有服务。详情请参见配置本地raft后端中的 步骤1。

3. 初始化配置。

   在SmartData服务的**bigboot**页签，设置以下参数。

参数	描述	示例
namespace.backend.raft.asyncOtsEnabled	是否开启OTS同步上传，包括： • true • false	false
namespace.backend.raft.recoverymode	是否开启从OTS恢复元数据，包括： • true • false	true

4. 保存配置。

   a) 单击右上角的**保存**。

   b) 在**确认修改**对话框中，输入执行原因，开启 **自动更新配置**。

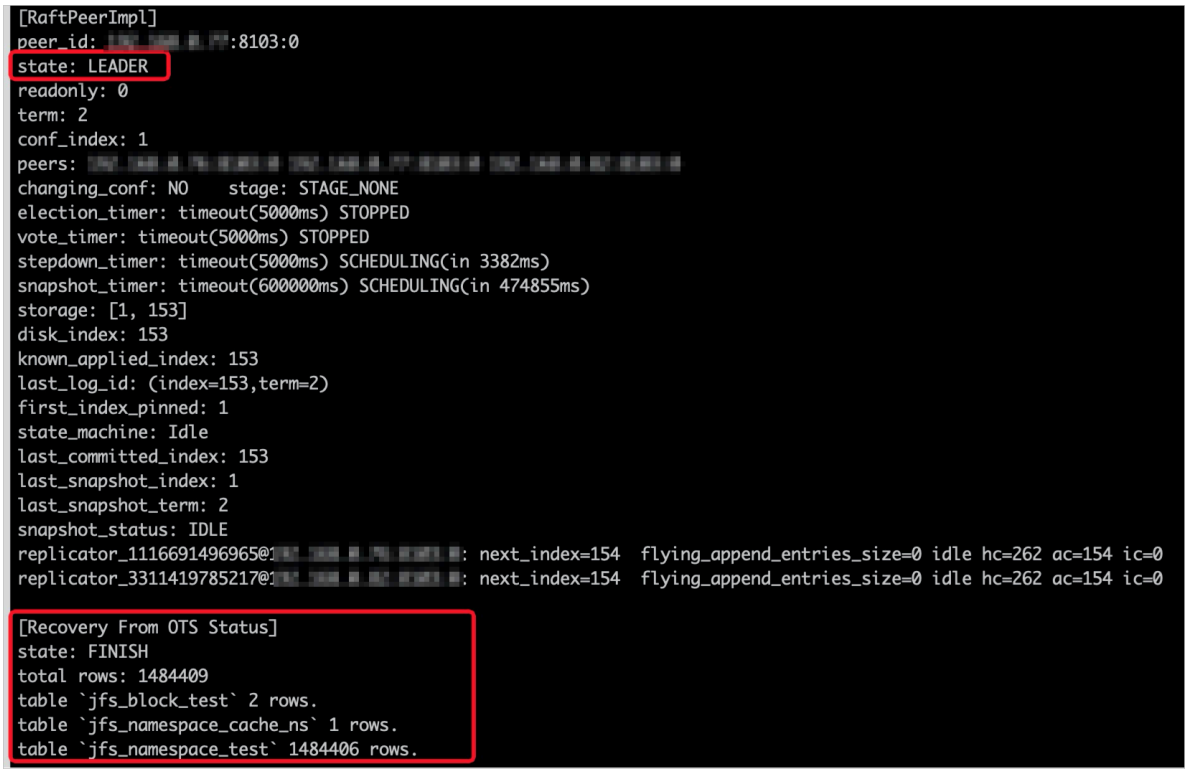
   c) 单击**确定**。

5. 单击右上角的**操作** > **启动 All Components**。

6. 新集群的SmartData服务启动后，自动从OTS恢复元数据到本地Raft-RocksDB上，可以通过以下命令查看恢复进度。

```
jindo jfs -metaStatus -detail
```

   如图所示，LEADER节点的state为FINISH表示恢复完成。



7. （可选） 执行以下操作，可以比较一下文件数量与原始集群是否一致。

   此时的集群为恢复模式，也是只读模式。

```
# 对比文件数量一致
[hadoop@emr-header-1 ~]$ hadoop fs -count jfs://test/
      1596     1482809              25 jfs://test/

# 文件可正常读取(cat、get命令)
[hadoop@emr-header-1 ~]$ hadoop fs -cat jfs://test/testfile
this is a test file
# 查看目录
[hadoop@emr-header-1 ~]$ hadoop fs -ls jfs://test/
Found 3 items
drwxrwxr-x   - root   root          0 2020-03-25 14:54 jfs://test/emr-header-1.cluster-
50087
```

```
-rw-r-----   1 hadoop hadoop          5 2020-03-25 14:50 jfs://test/haha-12096RANDOM
.txt
-rw-r-----   1 hadoop hadoop         20 2020-03-25 15:07 jfs://test/testfile

# 只读状态，不可修改文件
[hadoop@emr-header-1 ~]$ hadoop fs -rm jfs://test/testfile
java.io.IOException: ErrorCode : 25021 , ErrorMsg: Namespace is under recovery mode
, and is read-only.
```

8. 修改配置，将集群设置为正常模式，开启OTS异步上传功能。

   在SmartData服务的**bigboot**页签，设置以下参数。

参数	描述	示例
namespace.backend.raft.asyncOts.enable	是否开启OTS异步上传，包括： • true • false	true
namespace.backend.raft.recovery.mode	是否开启从OTS恢复元数据，包括： • true • false	false

9. 重启集群。

   a) 单击上方的**集群管理**页签。

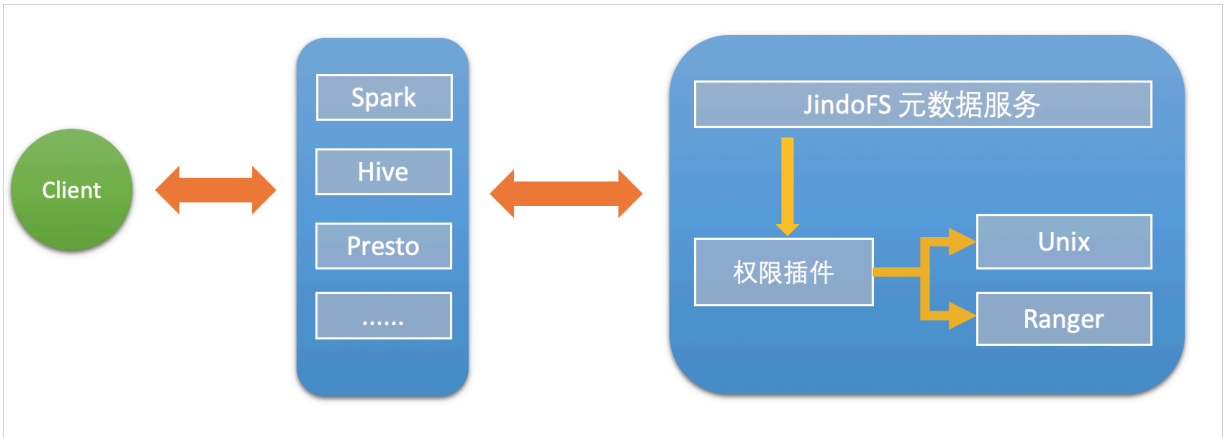
   b) 在**集群管理**页面，单击相应集群所在行的**更多** > **重启**。

2.5 JindoFS权限功能

JindoFS的Block模式支持文件系统权限功能。权限管理目前支持两种方式，unix权限和Ranger权限。

背景信息

使用unix权限，用户可以设置JindoFS文件系统上的文件的777权限，owner、group，根据这些信息对当前用户的权限进行校验。使用Ranger权限，用户可以配置更复杂的权限，比如使用路径通配符。用户可以在Apache Ranger组件上配置用户权限，然后在JindoFS上开启Ranger插件，就可以在Ranger上对JindoFS权限（和其它组件权限）进行一站式管理。



启用JindoFS Unix权限

1. 进入SmartData服务。
   a)
   b)
   c)
   d)
   e) 在左侧导航栏单击**集群服务** > **SmartData**。
2. 进入bigboot服务配置。
   a) 单击**配置**页签。
   b) 单击**bigboot**。

3. 单击**自定义配置**，在**新增配置项**对话框中，设置Key为**jfs.namespaces.<namespace>.permission.method**，Value为**unix**，单击**确定**。
4. 保存配置。
   a) 单击右上角的**保存**。
   b) 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。
   c) 单击**确定**。
5. 单击右上角的**操作** > **重启 Jindo Namespace Service**。

   开启文件系统权限后，使用方式跟HDFS一样。支持以下命令：

```
hadoop fs -chmod 777 jfs://{namespace_name}/dir1/file1
```

```
hadoop fs -chown john:staff jfs://{namespace_name}/dir1/file1
```

如果用户对某一个文件没有权限，将返回如下错误信息。

```
[root@emr-header-1 ~]# hadoop fs -ls jfs://test/user/
ls: java.security.AccessControlException: Permission denied. Server Exception
```

启用JindoFS Ranger权限

1. 添加ranger。
   a) 在**bigboot**页签，单击**自定义配置**。
   b) 在**新增配置项**对话框中，设置Key为**jfs.namespaces.<namespace>.permission.method**，Value为**ranger**。
   c) 保存配置。
      A. 单击右上角的**保存**。
      B. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。
      C. 单击**确定**。
   d) 单击右上角的**操作** > **重启 Jindo Namespace Service**。
2. 配置ranger。
   a) 进入Ranger UI页面。
      进入详情请参见#unique_22
   b) Ranger UI添加HDFS service。

   c) 配置相关参数。

参数	说明
Service Name	jfs-{namespace_name}。
Username	自定义。
Password	自定义。
Namenode URL	输入jfs://{namespace_name}。
Authorization Enabled	使用默认值No。
Authentication Type	使用默认值Simple。
dfs.datanode.kerberos.principal	不填写。

参数	说明
dfs.namenode.kerberos.principal	
dfs.secondary.namenode.kerberos.principal	
Add New Configurations	不填写。

d) 保存配置。

A. 单击右上角的**保存**。

B. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。

C. 单击**确定**。

启用JindoFS Ranger权限+LDAP用户组

如果您在Ranger UserSync上开启了从LDAP同步用户组信息的功能，那么JindoFS也需要修改相应的配置，才能获取LDAP的用户组信息，从而对当前用户组进行Ranger的权限校验。

1. 在**bigboot**页签，单击**自定义配置**。
2. 在**新增配置项**对话框中，设置以下参数配置LDAP，单击**确定**。

参数	示例
hadoop.security.group.mapping	org.apache.hadoop.security.CompositeGroupsMapping
hadoop.security.group.mapping.providers	shell4services,ad4users
hadoop.security.group.mapping.providers.combined	true
hadoop.security.group.mapping.provider.shell4services	org.apache.hadoop.security.ShellBasedUnixGroupsMapping
hadoop.security.group.mapping.provider.ad4users	org.apache.hadoop.security.LdapGroupsMapping
hadoop.security.group.mapping.ldap.url	ldap://emr-header-1:10389
hadoop.security.group.mapping.ldap.search.filter.user	(&(objectClass=person)(uid={0}))
hadoop.security.group.mapping.ldap.search.filter.group	(objectClass=groupOfNames)
hadoop.security.group.mapping.ldap.base	o=emr

说明：

配置项请遵循开源HDFS相关内容。

3. 保存配置。

   a) 单击右上角的**保存**。

   b) 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。

   c) 单击**确定**。

4. 单击右上角的**操作** > **重启 All Components**。

5. 通过ssh登录emr-header-1节点，配置Ranger UserSync，启用LDAP选项。

   详情请参见#unique_23。

2.6 Jindo Job Committer使用说明

本文主要介绍JindoOssCommitter的使用说明。

背景信息

Job Committer是MapReduce或Spark等分布式计算框架的一个基础组件，用来处理分布式任务写数据的一致性问题。MapReduce和Spark默认使用FileOutputCommitter的基本实现方式是：Task将数据写到task临时目录，在task完成时，将文件移动到job临时目录，最后在所有task都完成后，在MapReduce Application Master或者Spark Driver中，将文件从job临时目录移动到最终目录。文件的移动是通过rename实现的，对于HDFS来说，rename的代价很小，这种commit方式安全且高效。但是对于OSS对象存储系统来说，rename相当于一次拷贝+删除的操作，job commit的时间随着分布式任务写出数据的增加线性增长，对于大数据量任务，这种job commit方式基本不太可用。

Jindo Job Committer是阿里云EMR针对OSS场景开发的一个高效的Job Committer实现，基于OSS的multipart upload接口，结合OSS Filesystem层的定制化支持，使用Jindo Job Committer时，task数据直接写到最终目录中，且在完成job commit前，中间数据对外不可见，彻底避免了rename操作，同时保证数据的一致性。

注意：

- OSS拷贝数据的性能，针对不同的用户或Bucket可能有差异，和OSS带宽、是否开启某些高级特性等诸多因素有关，具体问题可咨询OSS的技术支持。
- 在所有任务都完成后，MapReduce Application Master或Spark Driver执行最终的job commit操作时，会有一个短暂的时间窗口，在这个时间窗口因为节点crash等其他因素导致任务失败时，无法保证写出数据的一致性，可能会出现部分结果文件在最终目录中的情况。时间窗口的大小和文件数量线性相关，通过增大fs.oss.committer.threads可以提高并发处理的速度。
- Hive，Presto等没有使用Hadoop的Job Committer，而是使用了一套自己写出数据一致性保证的机制，所以无法使用Jindo Oss Committer优化。

- Jindo Oss Committer相关参数在EMR集群中默认打开。

在MapReduce中使用Jindo Job Committer

1. 进入YARN服务的**mapred-site**页签。
   a) 登录阿里云E-MapReduce控制台。
   b) 在顶部菜单栏处，根据实际情况选择地域（Region）和资源组。
   c) 单击上方的**集群管理**页签。
   d) 在**集群管理**页面，单击相应集群所在行的**详情**。
   e) 在左侧导航栏单击**集群服务** > **YARN**。
   f) 单击**配置**页签。
   g) 在**服务配置**区域，单击**mapred-site**页签。
2. 针对Hadoop不同版本，在YARN服务中配置以下参数。
   - Hadoop 2.x版本

     在YARN服务的**mapred-site**页签，设置**mapreduce.outputcommitter.class**为**com.aliyun.emr.fs.oss.commit.JindoOssCommitter**。
   - Hadoop 3.x版本

     在YARN服务的**mapred-site**页签，设置**mapreduce.outputcommitter.factory.scheme.oss**为**com.aliyun.emr.fs.oss.commit.JindoOssCor**
3. 保存配置。
   a) 单击右上角的**保存**。
   b) 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。
   c) 单击**确定**。
4. 进入SmartData服务的**smartdata-site**页签。
   a) 在左侧导航栏单击**集群服务** > **SmartData**。
   b) 单击**配置**页签。
   c) 在**服务配置**区域，单击**smartdata-site**页签。
5. 在SmartData服务的**smartdata-site**页签，设置**fs.oss.committer.magic.enabled**为**true**。
6. 保存配置。
   a) 单击右上角的**保存**。
   b) 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。
   c) 单击**确定**。

说明：

在设置**mapreduce.outputcommitter.class**为**com.aliyun.emr.fs.oss.commit.JindoOssCommitter**后，可以通过开关**fs.oss.committer.magic.enabled**便捷地控制所使用的job committer。当打开时，MapReduce任务会使用无需Rename操作的Jindo Oss “Magic” Committer，当关闭时，JindoOssCommitter行为会和FileOutputCommitter一样。

在Spark中使用Jindo Job Committer

1. 进入Spark服务的**spark-defaults**页签。
   a) 在左侧导航栏单击**集群服务** > **Spark**。
   b) 单击**配置**页签。
   c) 在**服务配置**区域，单击**spark-defaults**页签。
2. 在Spark服务的**spark-defaults**页签，设置以下参数。

参数	参数值
spark.sql.sources.outputCommitterClass	com.aliyun.emr.fs.oss.commit.JindoOssCommitter
spark.sql.parquet.output.committer.class	com.aliyun.emr.fs.oss.commit.JindoOssCommitter
spark.sql.hive.outputCommitterClass	com.aliyun.emr.fs.oss.commit.JindoOssCommitter

   这三个参数分别用来设置写入数据到Spark DataSource表，Spark Parquet格式的DataSource表和Hive表时使用的job committer，可根据需要具体设置。
3. 保存配置。
   a) 单击右上角的**保存**。
   b) 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。
   c) 单击**确定**。
4. 进入SmartData服务的**smartdata-site**页签。
   a) 在左侧导航栏单击**集群服务** > **SmartData**。
   b) 单击**配置**页签。
   c) 在**服务配置**区域，单击**smartdata-site**页签。

5. 在SmartData服务的**smartdata-site**页签，设置**fs.oss.committer.magic.enabled**为**true**。

   说明：
   可以通过开关fs.oss.committer.magic.enabled便捷地控制所使用的job committer。当打开时，Spark任务会使用无需Rename操作的Jindo Oss “Magic” Committer，当关闭时，JindoOssCommitter行为会和FileOutputCommitter一样。

6. 保存配置。

   a) 单击右上角的**保存**。

   b) 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。

   c) 单击**确定**。

优化Jindo Job Committer性能

当MapReduce或Spark任务写大量文件的时候，可以调整MapReduce Application Master或Spark Driver中并发执行commit相关任务的线程数量，提升job commit性能。

1. 进入SmartData服务的**smartdata-site**页签。

   a) 在左侧导航栏单击**集群服务** > **SmartData**。

   b) 单击**配置**页签。

   c) 在**服务配置**区域，单击**smartdata-site**页签。

2. 在SmartData服务的**smartdata-site**页签，设置**fs.oss.committer.threads**为**8**。

   默认值为8。

3. 保存配置。

   a) 单击右上角的**保存**。

   b) 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。

   c) 单击**确定**。

3 JindoFS基础使用（EMR-3.28.0以上版本）

3.1 Jindo DistCp使用说明

本文介绍JindoFS的数据迁移工具Jindo DistCp的使用方法。

前提条件

已创建EMR-3.28.0版本的集群，详情请参见#unique_6。

使用Jindo Distcp

执行以下命令，获取帮助信息。

```
[root@emr-header-1 opt]# jindo distcp --help
```

返回信息如下。

```
    --help   -   Print help text
    --src=VALUE   -   Directory to copy files from
    --dest=VALUE   -   Directory to copy files to
    --parallelism=VALUE   -   Copy task parallelism
    --outputManifest=VALUE   -   The name of the manifest file
    --previousManifest=VALUE   -   The path to an existing manifest file
    --requirePreviousManifest=VALUE   -   Require that a previous manifest is present if
specified
    --copyFromManifest   -   Copy from a manifest instead of listing a directory
    --srcPrefixesFile=VALUE   -   File containing a list of source URI prefixes
    --srcPattern=VALUE   -   Include only source files matching this pattern
    --deleteOnSuccess   -   Delete input files after a successful copy
    --outputCodec=VALUE   -   Compression codec for output files
    --groupBy=VALUE   -   Pattern to group input files by
    --targetSize=VALUE   -   Target size for output files
    --enableBalancePlan   -   Enable plan copy task to make balance
    --enableDynamicPlan   -   Enable plan copy task dynamically
    --enableTransaction   -   Enable transation on Job explicitly
    --diff   -   show the difference between src and dest filelist
```

Jindo DistCp参数详细信息如下：

- --src和--dest
- --parallelism
- --srcPattern
- --deleteOnSuccess
- --outputCodec
- --outputManifest和--requirePreviousManifest
- --outputManifest和--previousManifest
- --copyFromManifest

- --srcPrefixesFile
- --groupBy和-targetSize
- --enableBalancePlan
- --enableDynamicPlan
- --enableTransaction
- --diff

--src和--dest

--src表示指定源文件的路径，--dest表示目标文件的路径。

Jindo DistCp默认将--src目录下的所有文件拷贝到指定的--dest路径下。您可以通过指定--dest路径来确定拷贝后的文件目录，如果不指定根目录，Jindo DistCp会自动创建根目录。

例如，如果您需要将/opt/tmp下的文件拷贝到OSS bucket，可以执行以下命令。

```
jindo distcp --src /opt/tmp --dest oss://yang-hhht/tmp
```

--parallelism

--parallelism用于指定MapReduce作业里的mapreduce.job.reduces参数。该参数默认为7，您可以根据集群的资源情况，通过自定义--parallelism大小来控制DistCp任务的并发度。

例如，将HDFS上/opt/tmp目录拷贝到OSS bucket，可以执行以下命令。

```
jindo distcp --src /opt/tmp --dest oss://yang-hhht/tmp --parallelism 20
```

--srcPattern

--srcPattern使用正则表达式，用于选择或者过滤需要复制的文件。您可以编写自定义的正则表达式来完成选择或者过滤操作，正则表达式必须为全路径正则匹配。

例如，如果您需要复制/data/incoming/hourly_table/2017-02-01/03下所有log文件，您可以通过指定--srcPattern的正则表达式来过滤需要复制的文件。

执行以下命令，查看/data/incoming/hourly_table/2017-02-01/03下的文件。

```
[root@emr-header-1 opt]# hdfs dfs -ls /data/incoming/hourly_table/2017-02-01/03
```

返回信息如下。

```
Found 6 items
-rw-r-----   2 root hadoop       2252 2020-04-17 20:42 /data/incoming/hourly_table/2017
-02-01/03/000151.sst
-rw-r-----   2 root hadoop       4891 2020-04-17 20:47 /data/incoming/hourly_table/2017
-02-01/03/1.log
-rw-r-----   2 root hadoop       4891 2020-04-17 20:47 /data/incoming/hourly_table/2017
-02-01/03/2.log
```

```
-rw-r-----   2 root hadoop      4891 2020-04-17 20:42 /data/incoming/hourly_table/2017
-02-01/03/OPTIONS-000109
-rw-r-----   2 root hadoop      1016 2020-04-17 20:47 /data/incoming/hourly_table/2017
-02-01/03/emp01.txt
-rw-r-----   2 root hadoop      1016 2020-04-17 20:47 /data/incoming/hourly_table/2017
-02-01/03/emp06.txt
```

执行以下命令，复制以log结尾的文件。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --
srcPattern .*\.log --parallelism 20
```

执行以下命令，查看目标bucket的内容。

```
[root@emr-header-1 opt]# hdfs dfs -ls oss://yang-hhht/hourly_table/2017-02-01/03
```

返回信息如下，显示只复制了以log结尾的文件。

```
Found 2 items
-rw-rw-rw-   1      4891 2020-04-17 20:52 oss://yang-hhht/hourly_table/2017-02-01/03/
1.log
-rw-rw-rw-   1      4891 2020-04-17 20:52 oss://yang-hhht/hourly_table/2017-02-01/03/
2.log
```

--deleteOnSuccess

--deleteOnSuccess可以移动数据并从源位置删除文件。

例如，执行以下命令，您可以将/data/incoming/下的hourly_table文件移动到OSS bucket中，并删除源位置文件。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --
deleteOnSuccess --parallelism 20
```

--outputCodec

--outputCodec可以在线高效地存储数据和压缩文件。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --
outputCodec=gz --parallelism 20
```

目标文件夹中的文件已经使用gz编解码器压缩了。

```
[root@emr-header-1 opt]# hdfs dfs -ls oss://yang-hhht/hourly_table/2017-02-01/03
```

返回信息如下：

```
Found 6 items
-rw-rw-rw-   1       938 2020-04-17 20:58 oss://yang-hhht/hourly_table/2017-02-01/03/
000151.sst.gz
-rw-rw-rw-   1      1956 2020-04-17 20:58 oss://yang-hhht/hourly_table/2017-02-01/03/
1.log.gz
-rw-rw-rw-   1      1956 2020-04-17 20:58 oss://yang-hhht/hourly_table/2017-02-01/03/
2.log.gz
```

```
-rw-rw-rw-  1      1956 2020-04-17 20:58 oss://yang-hhht/hourly_table/2017-02-01/03/
OPTIONS-000109.gz
-rw-rw-rw-  1       506 2020-04-17 20:58 oss://yang-hhht/hourly_table/2017-02-01/03/
emp01.txt.gz
-rw-rw-rw-  1       506 2020-04-17 20:58 oss://yang-hhht/hourly_table/2017-02-01/03/
emp06.txt.gz
```

Jindo DistCp当前版本支持编解码器gzip、gz、lzo、lzop、snappy以及关键字none和keep（默认值）。关键字含义如下：

- none表示保存为未压缩的文件。如果文件已压缩，则Jindo DistCp会将其解压缩。
- keep表示不更改文件压缩形态，按原样复制。

> **说明：**
> 如果您想在开源Hadoop集群环境中使用编解码器lzo，则需要安装gplcompression的native库和hadoop-lzo包。

--outputManifest和--requirePreviousManifest

--outputManifest可以指定生成DistCp的清单文件，用来记录copy过程中的目标文件、源文件和数据量大小等信息。

如果您需要生成清单文件，则指定--requirePreviousManifest为false。当前outputManifest文件默认且必须为gz类型压缩文件。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --
outputManifest=manifest-2020-04-17.gz --requirePreviousManifest=false --parallelism
20
```

查看outputManifest文件内容。

```
[root@emr-header-1 opt]# hadoop fs -text oss://yang-hhht/hourly_table/manifest-2020
-04-17.gz > before.lst
[root@emr-header-1 opt]# cat before.lst
```

返回信息如下。

```
{"path":"oss://yang-hhht/hourly_table/2017-02-01/03/000151.sst","baseName":"2017-
02-01/03/000151.sst","srcDir":"oss://yang-hhht/hourly_table","size":2252}
{"path":"oss://yang-hhht/hourly_table/2017-02-01/03/1.log","baseName":"2017-02-01/
03/1.log","srcDir":"oss://yang-hhht/hourly_table","size":4891}
{"path":"oss://yang-hhht/hourly_table/2017-02-01/03/2.log","baseName":"2017-02-01/
03/2.log","srcDir":"oss://yang-hhht/hourly_table","size":4891}
{"path":"oss://yang-hhht/hourly_table/2017-02-01/03/OPTIONS-000109","baseName":"
2017-02-01/03/OPTIONS-000109","srcDir":"oss://yang-hhht/hourly_table","size":4891}
{"path":"oss://yang-hhht/hourly_table/2017-02-01/03/emp01.txt","baseName":"2017-
02-01/03/emp01.txt","srcDir":"oss://yang-hhht/hourly_table","size":1016}
```

```
{"path":"oss://yang-hhht/hourly_table/2017-02-01/03/emp06.txt","baseName":"2017-
02-01/03/emp06.txt","srcDir":"oss://yang-hhht/hourly_table","size":1016}
```

--outputManifest和--previousManifest

--outputManifest表示包含所有已复制文件（旧文件和新文件）的列表，--previousManifest表示只包含之前复制文件的列表。您可以使用--outputManifest和--previousManifest重新创建完整的操作历史记录，查看运行期间复制的文件。

例如，在源文件夹中新增加了两个文件，命令如下所示。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_tab
le --outputManifest=manifest-2020-04-18.gz --previousManifest=oss://yang-hhht/
hourly_table/manifest-2020-04-17.gz --parallelism 20
```

执行以下命令，查看运行期间复制的文件。

```
[root@emr-header-1 opt]# hadoop fs -text oss://yang-hhht/hourly_table/manifest-2020
-04-18.gz > current.lst
[root@emr-header-1 opt]# diff before.lst current.lst
```

返回信息如下。

```
3a4,5
> {"path":"oss://yang-hhht/hourly_table/2017-02-01/03/5.log","baseName":"2017-02-
01/03/5.log","srcDir":"oss://yang-hhht/hourly_table","size":4891}
> {"path":"oss://yang-hhht/hourly_table/2017-02-01/03/6.log","baseName":"2017-02-
01/03/6.log","srcDir":"oss://yang-hhht/hourly_table","size":4891}
```

--copyFromManifest

使用--outputManifest生成清单文件后，您可以使用--copyFromManifest指定--outputManifest生成的清单文件，并将dest目录生成的清单文件中包含的文件信息拷贝到新的目录下。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --
previousManifest=oss://yang-hhht/hourly_table/manifest-2020-04-17.gz --copyFromMa
nifest --parallelism 20
```

--srcPrefixesFile

--srcPrefixesFile可以一次性完成多个文件夹的复制。

示例如下，查看hourly_table下文件。

```
[root@emr-header-1 opt]# hdfs dfs -ls oss://yang-hhht/hourly_table
```

返回信息如下。

```
Found 4 items
drwxrwxrwx   -          0 1970-01-01 08:00 oss://yang-hhht/hourly_table/2017-02-01
```

```
drwxrwxrwx   -          0 1970-01-01 08:00 oss://yang-hhht/hourly_table/2017-02-02
```

执行以下命令，复制hourly_table下文件到folders.txt。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --
srcPrefixesFile file:///opt/folders.txt --parallelism 20
```

查看folders.txt文件的内容。

```
[root@emr-header-1 opt]# cat folders.txt
```

返回信息如下。

```
hdfs://emr-header-1.cluster-50466:9000/data/incoming/hourly_table/2017-02-01
hdfs://emr-header-1.cluster-50466:9000/data/incoming/hourly_table/2017-02-02
```

--groupBy和-targetSize

因为Hadoop可以从HDFS中读取少量的大文件，而不再读取大量的小文件，所以在大量小文件的场景下，您可以使用Jindo DistCp将小文件聚合为指定大小的大文件，以便于优化分析性能和降低成本。

例如，执行以下命令，查看如下文件夹中的数据。

```
[root@emr-header-1 opt]# hdfs dfs -ls /data/incoming/hourly_table/2017-02-01/03
```

返回信息如下。

```
Found 8 items
-rw-r-----   2 root hadoop      2252 2020-04-17 20:42 /data/incoming/hourly_table/2017
-02-01/03/000151.sst
-rw-r-----   2 root hadoop      4891 2020-04-17 20:47 /data/incoming/hourly_table/2017
-02-01/03/1.log
-rw-r-----   2 root hadoop      4891 2020-04-17 20:47 /data/incoming/hourly_table/2017
-02-01/03/2.log
-rw-r-----   2 root hadoop      4891 2020-04-17 21:08 /data/incoming/hourly_table/2017
-02-01/03/5.log
-rw-r-----   2 root hadoop      4891 2020-04-17 21:08 /data/incoming/hourly_table/2017
-02-01/03/6.log
-rw-r-----   2 root hadoop      4891 2020-04-17 20:42 /data/incoming/hourly_table/2017
-02-01/03/OPTIONS-000109
-rw-r-----   2 root hadoop      1016 2020-04-17 20:47 /data/incoming/hourly_table/2017
-02-01/03/emp01.txt
-rw-r-----   2 root hadoop      1016 2020-04-17 20:47 /data/incoming/hourly_table/2017
-02-01/03/emp06.txt
```

执行以下命令，将如下文件夹中的TXT文件合并为不超过10M的文件。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --
targetSize=10 --groupBy='.*/([a-z]+).*.txt' --parallelism 20
```

经过合并后，可以看到两个TXT文件被合并成了一个文件。

```
[root@emr-header-1 opt]# hdfs dfs -ls oss://yang-hhht/hourly_table/2017-02-01/03/
```

```
Found 1 items
-rw-rw-rw-   1       2032 2020-04-17 21:18 oss://yang-hhht/hourly_table/2017-02-01/03/
emp2
```

--enableBalancePlan

在您要拷贝的数据大小均衡、小文件和大文件混合的场景下，因为DistCp默认的执行计划是随机进行文件分配的，所以您可以指定--enableBalancePlan来更改Jindo DistCp的作业分配计划，以达到更好的DistCp性能。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --
enableBalancePlan --parallelism 20
```

说明：

该参数不支持和--groupby或--targetSize同时使用。

--enableDynamicPlan

当您要拷贝的数据大小分化严重、小文件数据较多的场景下，您可以指定--enableDynamicPlan来更改Jindo DistCp的作业分配计划，以达到更好的DistCp性能。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --
enableDynamicPlan --parallelism 20
```

说明：

该参数不支持和--groupby或--targetSize参数一起使用。

--enableTransaction

--enableTransaction可以保证Job级别的完整性以及保证Job之间的事务支持。示例如下。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --
enableTransaction --parallelism 20
```

--diff

DistCp任务完成后，您可以使用--diff查看当前DistCp的文件差异。

例如，执行以下命令，查看/data/incoming/。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --diff
```

如果全部任务完成则会提示如下信息。

```
INFO distcp.JindoDistCp: distcp has been done completely
```

如果src的文件未能同步到dest上，则会在当前目录下生成manifest文件，您可以使用--copyFromManifest和--previousManifest拷贝剩余文件，从而完成数据大小和文件个数的校验。如果您的DistCp任务包含压缩或者解压缩，则--diff不能显示正确的文件差异，因为压缩或者解压缩会改变文件的大小。

```
jindo distcp --src /data/incoming/hourly_table --dest oss://yang-hhht/hourly_table --dest oss://yang-hhht/hourly_table --previousManifest=file:///opt/manifest-2020-04-17.gz --copyFromManifest --parallelism 20
```

说明：

如果您的--dest为HDFS路径，目前仅支持/path、hdfs://hostname:ip/path和hdfs://headerIp:ip/path的写法，暂不支持hdfs:///path、hdfs:/path和其他自定义写法。

查看Distcp Counters

执行以下命令，在MapReduce的Counter信息中查找Distcp Counters的信息。

```
Distcp Counters
        Bytes Destination Copied=11010048000
        Bytes Source Read=11010048000
        Files Copied=1001

Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
```

说明：

如果您的DistCp操作中包含压缩或者解压缩文件，则Bytes Destination Copied和Bytes Source Read的大小可能不相等。

4 JindoFS 生态

4.1 迁移Hadoop文件系统数据至JindoFS

本文以OSS为例，介绍如何将Hadoop文件系统上的数据迁移至JindoFS。

迁移数据

- Hadoop FsShell

  对于文件较少或者数据量较小的场景，可以直接使用Hadoop的FsShell进行同步：

  - `hadoop dfs -cp hdfs://emr-cluster/README.md jfs://emr-jfs/`
  - `hadoop dfs -cp oss://oss_bucket/README.md jfs://emr-jfs/`

- DistCp

  对于文件较多或者数据量较大的场景，推荐使用Hadoop内置的DistCp进行同步：

  - `hadoop distcp hdfs://emr-cluster/files jfs://emr-jfs/output/`
  - `hadoop distcp oss://oss_bucket/files jfs://emr-jfs/output/`

  说明：
  更多DistCp参数可参见DistCp Version2 Guide。

利用JindoFS缓存模式

缓存模式是兼容现有OSS的存储方式：文件会以原生对象的形式存储在OSS上，同时OSS文件通过JindoFS缓存模式访问时，也有机会在本地进行数据和元数据的缓存、加速访问，具体可参见JindoFS缓存模式。

4.2 使用MapReduce处理JindoFS上的数据

本文介绍如何使用MapReduce读写JindoFS上的数据。

JindoFS配置

已创建名为emr-jfs的命名空间，示例如下：

- **jfs.namespaces=emr-jfs**
- **jfs.namespaces.emr-jfs.uri=oss://oss-bucket/oss-dir**
- **jfs.namespaces.emr-jfs.mode=block**

MapReduce简介

Hadoop MapReduce作业一般通过HDFS进行读写，JindoFS目前已兼容大部分HDFS接口，通常只需要将MapReduce作业的输入、输出目录配置到JindoFS，即可实现读写JindoFS上的文件 。

Hadoop Map/Reduce是一个使用简易的软件框架，基于它写出来的应用程序能够运行在由上千个商用机器组成的大型集群上，并以一种可靠容错的方式并行处理上T级别的数据集。一个Map/Reduce作业（job） 通常会把输入的数据集切分为若干独立的数据块，由map任务（task）以完全并行的方式处理它们。框架会对map的输出先进行排序， 然后把结果输入给reduce任务。通常作业的输入和输出都会被存储在文件系统中。 整个框架负责任务的调度和监控，以及重新执行已经失败的任务。

作业的输入和输出

MapReduce作业一般会指明输入/输出的位置（路径），并通过实现合适的接口或抽象类提供map和reduce函数。Hadoop的 job client 再加上其他作业的参数提交给ResourceManager，进行调度执行。这种情况下，我们直接修改作业的输入和输出目录即可实现JindoFS的读写。

MapReduce on JindoFS样例

以下是MapReduce作业通过修改输入输出实现JindoFS的读写的例子。

- Teragen数据生成样例

  Teragen是Example中生成随机数据演示程序，在指定目录上生成指定行数的数据，具体命令如下：

  ```
  hadoop jar /usr/lib/hadoop-current/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar teragen <num rows> <output dir>
  ```

  替换输出路径，可以把数据输出到JindoFS 上：

  ```
  hadoop jar /usr/lib/hadoop-current/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar teragen 100000 jfs://emr-jfs/teragen_data_0
  ```

- Terasort数据生成样例

  Terasort是Example中数据排序演示样例，有输入和输出目录，具体命令如下：

  ```
  hadoop jar /usr/lib/hadoop-current/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar terasort <in> <out>
  ```

  替换输入和输出路径，即可处理JindoFS上的数据：

  ```
  hadoop jar /usr/lib/hadoop-current/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar terasort jfs://emr-jfs/teragen_data_0/ jfs://emr-jfs/terasort_data_0
  ```

4.3 使用Hive查询JindoFS上的数据

Apache Hive是Hadoop生态中广泛使用的SQL引擎之一，让用户可以使用SQL实现分布式的查询，Hive中数据主要以undefinedDatabase、Table和Partition的形式进行管理，通过指定位置（Location）对应到后端的数据。

JindoFS配置

已创建名为emr-jfs的命名空间，示例如下：

- **jfs.namespaces=emr-jfs**
- **jfs.namespaces.emr-jfs.uri=oss://oss-bucket/oss-dir**
- **jfs.namespaces.emr-jfs.mode=block**

Warehouse/Database/Table/Partition的Location

- Warehouse的Location

  Hive的hive-site中有hive.metastore.warehouse.dir，表示Hive数仓存放数据的默认路径，例如配置成：`jfs://emr-jfs/user/hive/warehouse`。

- Database的Location

  Hive的Database会有一个Location属性，database的Location作为下属Table的默认路径。默认情况下，创建Database不是必须指定Location，默认会使用hive-site中hive.metastore.warehouse.dir的值加上database的名字作为路径。通过下面的命令可以指定Database的Location到JindoFS：

  - 创建Database时指定Location到JindoFS。

    ```
    CREATE DATABASE database_name
    LOCATION
    'jfs://namespace/database_dir';
    ```

    例如，创建名为database_on_jindofs，location为jfs://emr-jfs/warehouse/database_on_jindofs的Hive数据库。

    ```
    CREATE DATABASE database_on_jindofs
    LOCATION
    'jfs://emr-jfs/hive/warehouse/database_on_jindofs';
    ```

  - 修改Database的Location到JindoFS。

    1. 通过SHOW CREATE语句查看Database的Location。

       ```
       SHOW CREATE DATABASE database_name;
       ```

    2. 一般情况下，默认为warehouse目录，查询结果如下。

       ```
       CREATE DATABASE `database_name`
       LOCATION
       'hdfs://emr-jfs/user/hive/warehouse/database_name.db'
       ```

    3. 通过修改Location，可以把默认路径指定到JindoFS上。此操作不会影响存量表，当新建表没有指定默认Location时，才会使用此目录。

       例如，查看表 jfs_table_name下的某个Partition。

       ```
       ALTER DATABASE database_name SET LOCATION jfs_path;
       ```

- Table/Partition的Location

  Table/Partition的Location与Database类似，对于非Partition表，数据直接存放在Table Location下，Partition表的数据存放在Partition目录下，相关操作如下：

  - 创建Table时指定Location到JindoFS。

    ```
    CREATE [EXTERNAL] TABLE table_name
      [(col_name data_type,...)]
    ```

```
LOCATION 'jfs://emr-jfs/database_dir/table_dir';
```

- 修改Table/Partition指定Location到 JindoFS。

  1. 通过DESCRIBE语句查看Table/Partition的 location。

     ```
     DESCRIBE FORMATTED [PARTITION partition_spec] table_name;
     ```

  2. 通过修改Location，可以把默认路径指定到JindoFS上。

     ```
     ALTER TABLE table_name [PARTITION partition_spec] SET LOCATION "jfs_path";
     ```

     例如，查看表 jfs_table_name下的某个Partition。

     ```
     DESCRIBE FORMATTED jfs_table_name PARTITION (partition_key1=123,partition_
     key2='xxxx');
     ```

Hive scratch目录

Hive会把一些临时输出文件和作业计划存储在scratch目录，可以通过设置hive-site的hive.exec.scratchdir把地址指向到JindoFS，也可以通过命令行传参。

```
bin/hive --hiveconf hive.exec.scratchdir=jfs://emr-jfs/scratch_dir
```

或者

```
set hive.exec.scratchdir=jfs://emr-jfs/scratch_dir;
```

4.4 使用Spark处理JindoFS上的数据

Spark处理JindoFS上的数据，主要有两种方式，一种是直接调用文件系统接口使用；一种是通过SparkSQL读取存在JindoFS的数据表。

JindoFS配置

已创建名为emr-jfs的命名空间，示例如下：

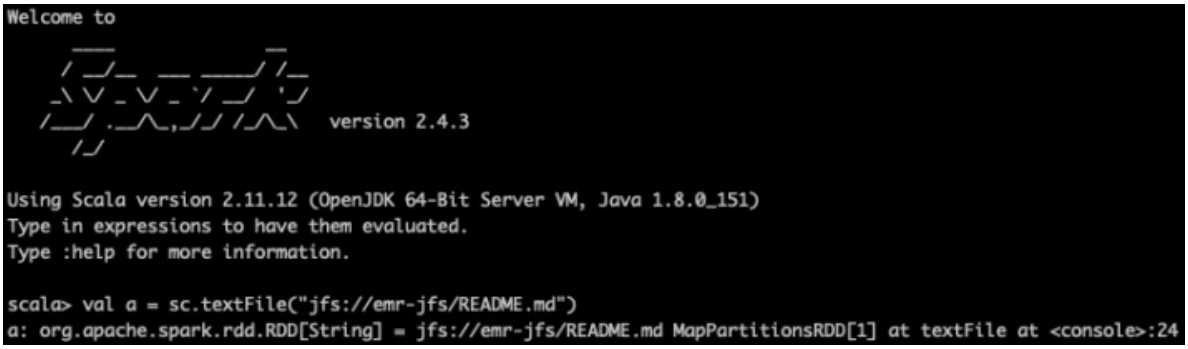
- **jfs.namespaces=emr-jfs**
- **jfs.namespaces.emr-jfs.uri=oss://oss-bucket/oss-dir**
- **jfs.namespaces.emr-jfs.mode=block**

处理JindoFS上的数据

- 调用文件系统

  Spark中读写JindoFS上的数据，与处理其他文件系统的数据类似，以RDD操作为例，直接使用jfs的路径即可：

  ```
  val a = sc.textFile("jfs://emr-jfs/README.md")
  ```

  写入数据：

  ```
  scala> a.collect().saveAsTextFile("jfs://emr-jfs/output")
  ```

- SparkSQL

  创建数据库、数据表以及分区时指定Location到JindoFS即可，详情请参见使用Hive查询JindoFS上的数据。对于已经创建好的存储在JindoFS上的数据表，直接查询即可。

4.5 使用Flink处理JindoFS上的数据

本文介绍如何使用Flink处理JindoFS上的数据。

JindoFS配置

已创建名为emr-jfs的命名空间，示例如下：

- **jfs.namespaces=emr-jfs**
- **jfs.namespaces.emr-jfs.uri=oss://oss-bucket/oss-dir**
- **jfs.namespaces.emr-jfs.mode=block**

使用JindoFS

Flink作业同样可以将作业的输入输出指定为JindoFS相应Namespace下的路径，即可实现Flink作业对JindoFS数据的交互。

例如，HDFS 上的作业命令如下：

```
flink run -m yarn-cluster -yD taskmanager.network.memory.fraction=0.4 -yD akka.ask.timeout=60s -yjm 2048 -ytm 2048 -ys 4 -yn 14 -c xxx.xxx.FlinkWordCount -p 56 XXX.jar --input hdfs:///test//large-input-flink --output hdfs:///runjob/test/large-output-flink"
```

相应的改成如下命令即可：

```
flink run -m yarn-cluster -yD taskmanager.network.memory.fraction=0.4 -yD akka.ask.timeout=60s -yjm 2048 -ytm 2048 -ys 4 -yn 14 -c xxx.xxx.FlinkWordCount -p 56 XXX.jar --input jfs://emr-jfs/test/large-input-flink --output jfs://emr-jfs/test/large-output-flink"
```

4.6 使用Impala/Presto查询JindoFS上的数据

本文介绍如何使用Impala/Presto查询JindoFS上的数据。

JindoFS配置

已创建名为emr-jfs的命名空间，示例如下：

- **jfs.namespaces=emr-jfs**
- **jfs.namespaces.emr-jfs.uri=oss://oss-bucket/oss-dir**
- **jfs.namespaces.emr-jfs.mode=block**

使用JindoFS

目前， 在E-MapReduce 3.22.0及以上版本，Impala/Presto支持Hive元数据的读取，对于存储在JindoFS的Hive数据表，E-MapReduce Impala/Presto可以直接读取。

同样，也可以将建表语句的Location设置为JindoFS路径，即可实现表数据落在JindoFS上。

例如，原有HDFS上的建表语句如下：

```
Create external table lineitem (L_ORDERKEY INT, L_PARTKEY INT, L_SUPPKEY INT, L_LINENUMBER INT, L_QUANTITY DOUBLE, L_EXTENDEDPRICE DOUBLE, L_DISCOUNT DOUBLE, L_TAX DOUBLE, L_RETURNFLAG STRING, L_LINESTATUS STRING, L_SHIPDATE STRING, L_COMMITDATE STRING, L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING, L_SHIPMODE STRING, L_COMMENT STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION 'hdfs:///tpch_impala/lineitem';
```

相应的改成如下命令即可：

```
Create external table lineitem (L_ORDERKEY INT, L_PARTKEY INT, L_SUPPKEY INT, L_LINENUMBER INT, L_QUANTITY DOUBLE, L_EXTENDEDPRICE DOUBLE, L_DISCOUNT DOUBLE, L_TAX DOUBLE, L_RETURNFLAG STRING, L_LINESTATUS STRING, L_SHIPDATE STRING, L_COMMITDATE STRING, L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING,
```

```
L_SHIPMODE STRING, L_COMMENT STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION 'jfs://emr-jfs/tpch_impala/lineitem';
```

4.7 使用JindoFS作为HBase的底层存储

本文介绍如何使用JindoFS作为HBase的底层存储。

背景信息

HBase是Hadoop生态中的实时数据库，有很高的写入性能，E-MapReduce HBase（E-MapReduce 3.22.0及以上版本） 支持使用 JindoFS/OSS作为底层存储，相对于HDFS存储来说，使用更加灵活。

JindoFS配置

已创建名为emr-jfs的命名空间，示例如下：

- **jfs.namespaces=emr-jfs**
- **jfs.namespaces.emr-jfs.uri=oss://oss-bucket/oss-dir**
- **jfs.namespaces.emr-jfs.mode=block**

指定HBase的存储路径

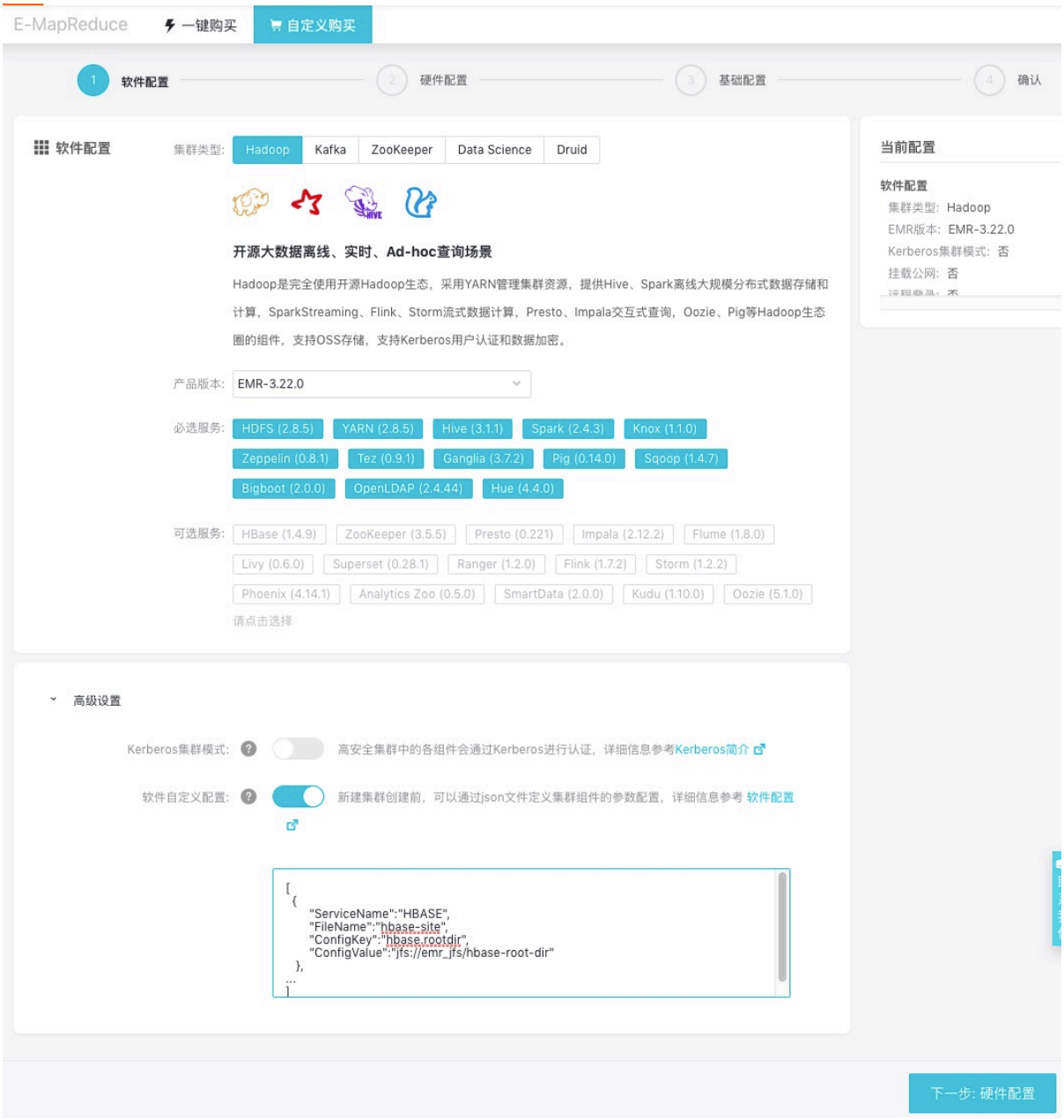
由于JindoFS和OSS在E-MapReduce 3.22.0版本暂不支持Sync操作，需要把hbase-site的hbase.rootdir指向JindoFS/OSS地址，hbase.wal.dir指向本地的undefinedHDFS地址，通过本地HDFS集群存储WAL文件。如果要释放集群，需要先Disable table，确保WAL文件已经完全更新到HFile。

配置文件	参数	参数说明	示例
hbase-site	**hbase.rootdir**	指定HBase的ROOT存储目录到JindoFS	jfs://emr-jfs/hbase-root-dir
	hbase.wal.dir	指定HBase的WAL存储目录到本地HDFS集群	hdfs://emr-cluster/hbase

创建集群

创建集群详情请参见#unique_6。

添加软件自定义配置，如下图所示。



使用JindoFS

以JindoFS作为HBase后端为例，替换oss_bucket及对应路径，自定义配置如下：

```
[
 {
    "ServiceName":"BIGBOOT",
    "FileName":"bigboot",
    "ConfigKey":"jfs.namespaces",
    "ConfigValue":"emr-jfs"
 },
 {
    "ServiceName":"BIGBOOT",
    "FileName":"bigboot",
    "ConfigKey":"jfs.namespaces.emr-jfs.uri",
    "ConfigValue":"oss://oss-bucket/jindoFS"
 },
 {
```

```
    "ServiceName":"BIGBOOT",
    "FileName":"bigboot",
    "ConfigKey":"jfs.namespaces.emr-jfs.mode",
    "ConfigValue":"block"
  },
  {
    "ServiceName":"HBASE",
    "FileName":"hbase-site",
    "ConfigKey":"hbase.rootdir",
    "ConfigValue":"jfs://emr-jfs/hbase-root-dir"
  },
  {
    "ServiceName":"HBASE",
    "FileName":"hbase-site",
    "ConfigKey":"hbase.wal.dir",
    "ConfigValue":"hdfs://emr-cluster/hbase"
  }
]
```

4.8 基于JindoFS存储YARN MR/SPARK作业日志

本文介绍如何将MapReduce、Spark作业日志配置到JindoFS/OSS上。

概述

E-MapReduce集群支持按量计费以及包年包月的付费方式，满足不同用户的使用需求。对于按量计费的集群随时会被释放，而Hadoop默认会把日志存储在HDFS上，当集群释放以后，按量计费的用户就无法查询作业的日志了，这也给按量计费用户排查作业问题带来了困难。本文会介绍如何将MapReduce、Spark作业日志配置到JindoFS/OSS上，集群重新创建以后，也可以继续查询之前作业相关的日志。

JindoFS、YARN Container日志和Spark HistoryServer配置

- JindoFS配置

配置文件	参数	参数说明	示例
bigboot	**jfs.namespaces**	表示当前JindoFS支持的命名空间，多个命名空间时以逗号隔开。	emr-jfs
	jfs.namespaces.emr-jfs.uri	表示emr-jfs命名空间的后端存储。	oss://oss-bucket/oss-dir
	jfs.namespaces.test.mode	表示emr-jfs命名空间为块存储模式。	block **说明：** JindoFS支持block和cache两种存储模式。

- YARN Container日志配置

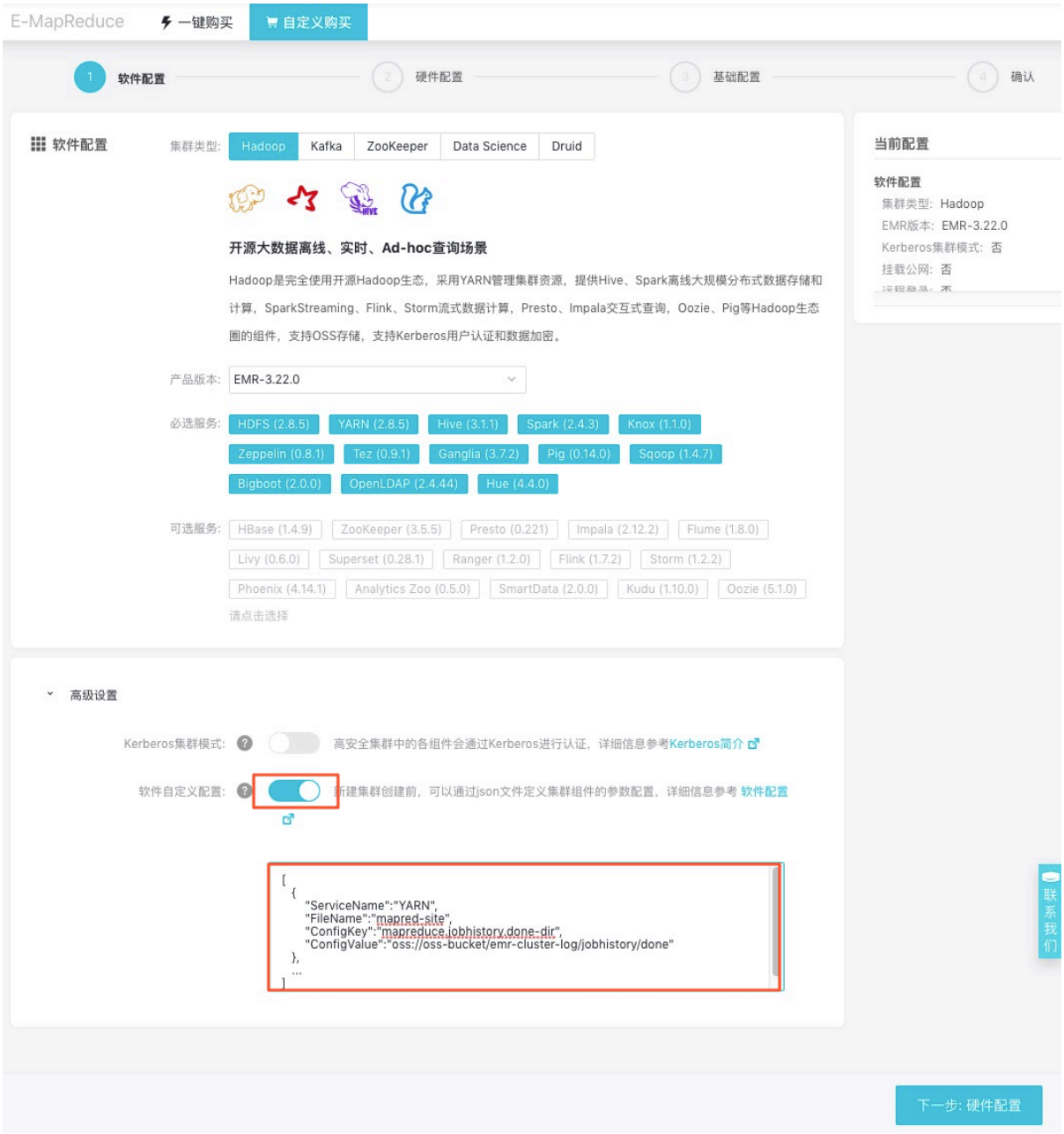
配置文件	参数	参数说明	示例
yarn-site	**yarn.nodemanager.remote-app-log-dir**	当应用程序运行结束后，日志聚合的存储位置，YARN日志聚合功能默认已打开。	jfs://emr-jfs/emr-cluster-log/yarn-apps-logs或者oss://${oss-bucket}/emr-cluster-log/yarn-apps-logs
mapred-site	**mapreduce.jobhistory.done-dir**	JobHistory存放已经运行完的Hadoop作业记录的目录。	jfs://emr-jfs/emr-cluster-log/jobhistory/done或者oss://${oss-bucket}/emr-cluster-log/jobhistory/done
	mapreduce.jobhistory.intermediate-done-dir	JobHistory存放未归档的 Hadoop作业记录的目录。	jfs://emr-jfs/emr-cluster-log/jobhistory/done_intermediate或者oss://${oss-bucket}/emr-cluster-log/jobhistory/done_intermediate

- Spark HistoryServer配置

配置文件	参数	参数说明	示例
spark-defaults	**spark_eventlog_dir**	存放Spark作业历史的目录。	jfs://emr-jfs/emr-cluster-log/spark-history或者oss://${oss-bucket}/emr-cluster-log/spark-history

创建集群

添加软件自定义配置，如下图所示。



JindoFS样例

以JindoFS存储日志为例，替换oss_bucket及对应路径，自定义配置如下：

```
[
 {
    "ServiceName":"BIGBOOT",
    "FileName":"bigboot",
    "ConfigKey":"jfs.namespaces",
    "ConfigValue":"emr-jfs"
 },
 {
    "ServiceName":"BIGBOOT",
    "FileName":"bigboot",
    "ConfigKey":"jfs.namespaces.emr-jfs.uri",
    "ConfigValue":"oss://oss-bucket/jindoFS"
 },
 {
```

```
    "ServiceName":"BIGBOOT",
    "FileName":"bigboot",
    "ConfigKey":"jfs.namespaces.emr-jfs.mode",
    "ConfigValue":"block"
  },
  {
    "ServiceName":"YARN",
    "FileName":"mapred-site",
    "ConfigKey":"mapreduce.jobhistory.done-dir",
    "ConfigValue":"jfs://emr-jfs/emr-cluster-log/jobhistory/done"
  },
  {
    "ServiceName":"YARN",
    "FileName":"mapred-site",
    "ConfigKey":"mapreduce.jobhistory.intermediate-done-dir",
    "ConfigValue":"jfs://emr-jfs/emr-cluster-log/jobhistory/done_intermediate"
  },
  {
    "ServiceName":"YARN",
    "FileName":"yarn-site",
    "ConfigKey":"yarn.nodemanager.remote-app-log-dir",
    "ConfigValue":"jfs://emr-jfs/emr-cluster-log/yarn-apps-logs"
  },
  {
    "ServiceName":"SPARK",
    "FileName":"spark-defaults",
    "ConfigKey":"spark_eventlog_dir",
    "ConfigValue":"jfs://emr-jfs/emr-cluster-log/spark-history"
  }
]
```

以OSS存储日志为例，替换oss_bucket及对应路径，自定义配置如下：

```
[
  {
    "ServiceName":"YARN",
    "FileName":"mapred-site",
    "ConfigKey":"mapreduce.jobhistory.done-dir",
    "ConfigValue":"oss://oss_bucket/emr-cluster-log/jobhistory/done"
  },
  {
    "ServiceName":"YARN",
    "FileName":"mapred-site",
    "ConfigKey":"mapreduce.jobhistory.intermediate-done-dir",
    "ConfigValue":"oss://oss_bucket/emr-cluster-log/jobhistory/done_intermediate"
  },
  {
    "ServiceName":"YARN",
    "FileName":"yarn-site",
    "ConfigKey":"yarn.nodemanager.remote-app-log-dir",
    "ConfigValue":"oss://oss_bucket/emr-cluster-log/yarn-apps-logs"
  },
  {
    "ServiceName":"SPARK",
    "FileName":"spark-defaults",
    "ConfigKey":"spark_eventlog_dir",
    "ConfigValue":"oss://oss_bucket/emr-cluster-log/spark-history"
   }
```

```
]
```

4.9 将Kafka数据导入JindoFS

Kafka广泛用于日志收集、监控数据聚合等场景，支持离线或流式数据处理、实时数据分析等。本文主要介绍Kafka数据导入到JindoFS的几种方式。

常见Kafka数据导入方式

- 通过Flume导入

  推荐使用Flume方式导入到JindoFS，利用Flume对HDFS的支持，替换路径到JindoFS即可完成。

  ```
  a1.sinks = emr-jfs

  ...

  a1.sinks.emr-jfs.type = hdfs
  a1.sinks.emr-jfs.hdfs.path = jfs://emr-jfs/kafka/%{topic}/%y-%m-%d
  a1.sinks.emr-jfs.hdfs.rollInterval = 10
  a1.sinks.emr-jfs.hdfs.rollSize = 0
  a1.sinks.emr-jfs.hdfs.rollCount = 0
  a1.sinks.emr-jfs.hdfs.fileType = DataStream
  ```

- 通过调用Kafka API导入

  对于MapReduce、Spark以及其他调用Kafka API导入数据的方式，只需引用Hadoop FileSystem，然后使用JindoFS的路径写入即可。

- 通过Kafka Connector导入

  使用Kafka HDFS Connector也可以把Kafka数据导入到Hadoop生态，将sink的输出路径替换成JindoFS的路径即可。

5 JindoCube

5.1 E-MapReduce JindoCube使用说明

JindoCube在E-MapReduce 3.24.0及之后版本中可用。本文主要介绍E-MapReduce JindoCube的安装、部署和使用等。

前提条件

已创建表或者视图。

概述

JindoCube是E-MapReduce Spark支持的高级特性，通过预计算加速数据处理，实现十倍甚至百倍的性能提升。您可以将任意View表示的数据进行持久化，持久化的数据可以保存在HDFS或OSS等任意Spark支持的DataSource中。EMR Spark自动发现可用的已持久化数据，并优化执行计划，对用户完全透明。JindoCube主要用于查询模式相对比较固定的业务场景，通过提前设计JindoCube，对数据进行预计算和预组织，从而加速业务查询的速度，常见的使用场景包括MOLAP多维分析、报表生成、数据Dashboard和跨集群数据同步等。

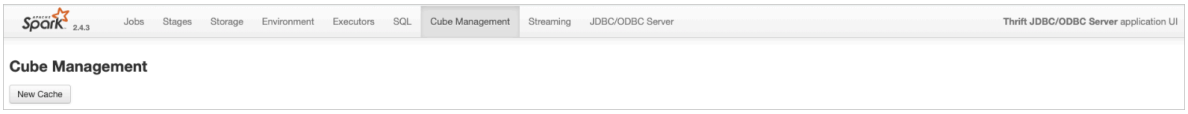
JindoCube的安装与部署

JindoCube作为EMR Spark组件的高级特性，所有使用EMR Spark提交的Dataset、DataFrame API、SQL任务，均可以基于JindoCube进行加速，无须额外的组件部署与维护。

1. UI页面展示。

   JindoCube主要通过Spark的UI页面进行管理，包括JindoCube的创建、删除和更新等。
   通过UI创建JindoCube完成后，即可自动用于该集群所有Spark任务的查询加速。通过**spark.sql.cache.tab.display**参数可以控制是否在Spark UI页面展示JindoCube的Tab，可以

通过EMR控制台在Spark服务中配置相关参数，或者在Spark提交命令中指定参数值，该参数默认值为**false**。



JindoCube还提供了**spark.sql.cache.useDatabase**参数，可以针对业务方向，按不同的业务建立database，把需要建cache的view放在这个database中。对于分区表JindoCube还提供了**spark.sql.cache.cacheByPartition**参数，可指定cache使用分区字段进行存储。

参数	说明	示例值
spark.sql.cache.tab.display	显示Cube Mangament页面。	true
spark.sql.cache.useDatabase	cube存储数据库。	db1,db2,dbn
spark.sql.cache.cacheByPartition	按照分区字段存储cube。	true

2. 优化查询。

spark.sql.cache.queryRewrite用于控制是否允许使用JindoCube中的Cache数据加速Spark查询任务，用户可以在集群、session、SQL等层面使用该配置，默认值为**true**。

JindoCube的使用

1. 创建JindoCube。
   a. 通过主账号登录阿里云 E-MapReduce 控制台。
   b. 单击**集群管理**页签。
   c. 单击待操作集群所在行的集群ID。
   d. 单击左侧导航栏的**访问链接与端口**。
   e. 在**公网访问链接**页面，单击YARN UI所在行的链接，进入Knox代理的YARN UI页面。

      Knox相关使用说明请参见#unique_39。
   f. 单击**User**为**spark**，**Name**为**Thrift JDBC/ODBC Server**所在行的**ApplicationMaster**。
   g. 单击最上面的**Cube Management**页签。
   h. 单击**New Cache**。

      用户可以选择某一个表或视图，单击action中的链接继续创建Cache。可以选择的Cache类型分为两类：

      - Raw Cache：某一个表或者视图的raw cache，表示将对应表或视图代表的表数据按照指定的方式持久化。

        在创建Raw Cache时，用户需要指定如下信息：

参数	描述	是否必选
Cache Name	指定Cache的名字，支持字母、数字、连接号（-）和下划线（_）的组合。	必选
Column Selector	选择需要Cache哪些列的数据。	必选
Rewrite	是否允许该Cache被用作后续查询的执行计划优化。	必选
Provider	Cache数据的存储格式，支持JSON、PARQUET、ORC等所有Spark支持的数据格式。	必选
Partition Columns	Cache数据的分区字段。	可选

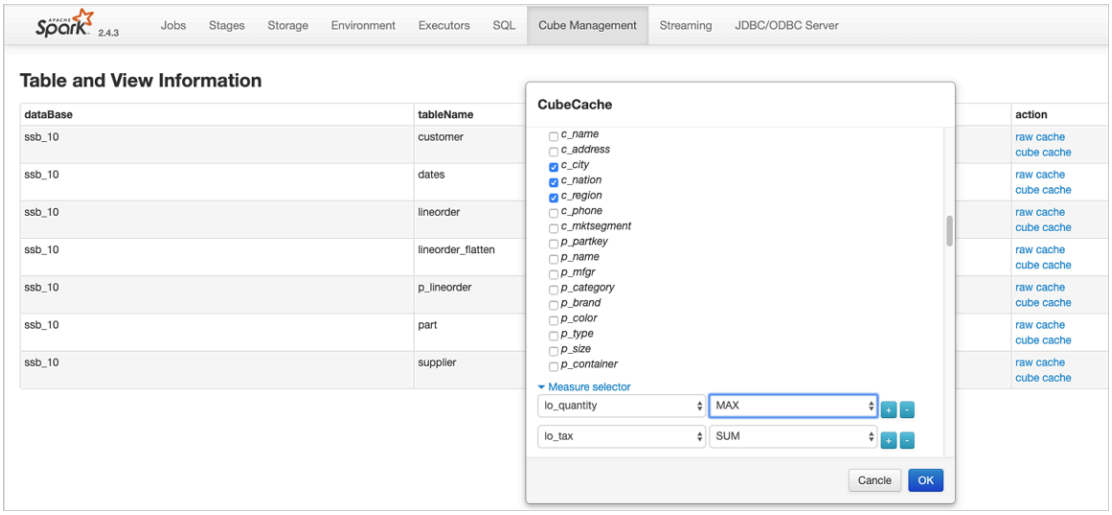
参数	描述	是否必选
ZOrder Columns	ZOrder是一种支持多列排序的方法，Cache数据按照ZOrder字段排序后，对于基于ZOrder字段过滤的查询会有更好的加速效果。	可选

- Cube Cache：基于某一个表或者视图的原始数据，按照用户指定的方式构建cube，并将cube数据持久化。

  在创建Cube Cache时，用户需要指定如下信息：

参数	描述	是否必选
Cache Name	指定Cache的名字，支持字母、数字、连接号（-）和下划线（_）的组合。	必选
Dimension Selector	选择构建Cube时的维度字段。	必选
Measure Selector	选择构建Cube时的measure字段和measure预计算函数。	必选
Rewrite	是否允许该Cache被用作后续查询的执行计划优化。	必选
Provider	Cache数据的存储格式，支持JSON、PARQUET、ORC等所有Spark支持的数据格式。	必选
Partition Columns	Cache数据的分区字段。	可选

参数	描述	是否必选
ZOrder Columns	ZOrder是一种支持多列排序的方法，Cache数据按照ZOrder字段排序后，对于基于ZOrder字段过滤的查询会有更好的加速效果。	可选



JindoCube通过用户指定的Dimension和Measure信息来构建Cube，对于上图的示例，创建的Cube Cache可以用SQL表示为：

```
SELECT c_city, c_nation, c_region, MAX(lo_quantity), SUM(lo_tax)
FROM lineorder_flatten
GROUP BY c_city, c_nation, c_region;
```

JindoCube计算Cube的最细粒度维度组合，在优化使用更粗粒度的维度组合的查询时，基于Spark强大的现场计算能力，通过重聚合实现。在定义Cube Cache时，必须使用JindoCube支持的预计算函数。JindoCube支持的预计算函数和其对应的聚合函数类型如下：

聚合函数类型	预计算函数
COUNT	COUNT
SUM	SUM
MAX	MAX
MIN	MIN
AVG	COUNT，SUM
COUNT（DISTINCT ）	PRE_COUNT_DISTINCT

聚合函数类型	预计算函数
APPROX_COUNT_DISTINCT	PRE_APPROX_COUNT_DISTINCT

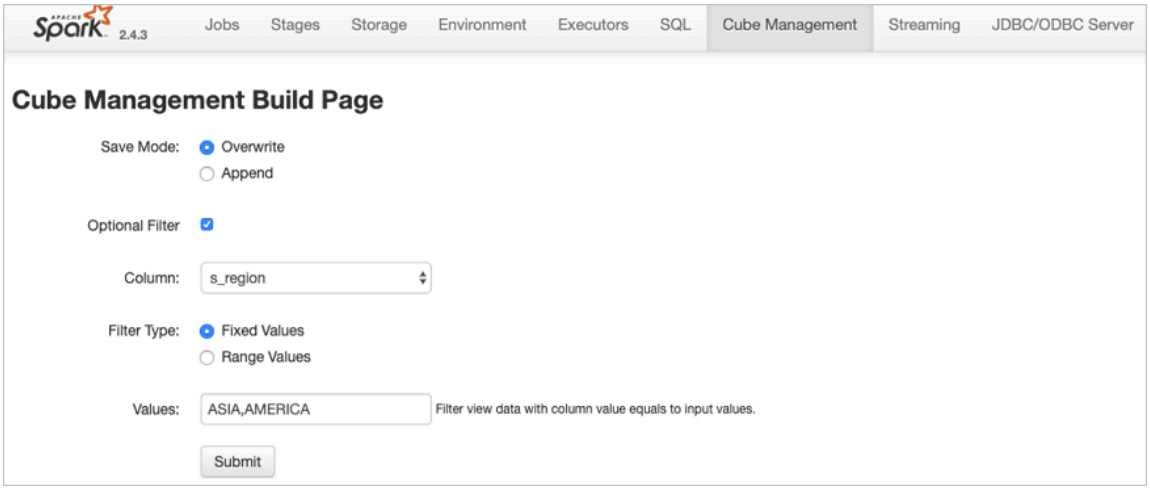
在**Cube Management**页面，展示所有的Cache列表。单击**Detail**进入Cache的详细页面，在Cache详细页会展示Cache的详细信息、包括基本信息、Cache数据分区信息、构建Cache信息以及构建历史信息等。

2. 构建JindoCube。

创建JindoCube Cache只是进行元数据操作，Cache表示的数据并未持久化，需要继续构建Cache，从而持久化Cache数据到HDFS或OSS等存储中。此外Cache对应的源表数据可能会新增或者更新，需要更新Cache中的数据从而保持一致。JindoCube支持两类构建操作：

- Build Cache。

  通过Build Cache链接，用户可以主动触发一次构建操作，构建页面相关信息如下：



在构建JindoCube的Cache时，相关用户选项如下：

参数	描述
Save Mode	支持Overwrite和Append两种模式。 - Overwrite：会覆盖之前曾经构建的Cache数据。 - Append：会新增数据到Cache中。

参数	描述
Optional Filter	用户可以选择额外的过滤条件，在构建时，将该Cache表示的数据过滤后再持久化。 - Column：过滤字段。 - Filter Type：过滤类型，支持固定值和范围值两种。 ■ Fixed Values：指定过滤值，可以多个，以“,”分隔。 ■ Range Values：指定范围值的最小和最大值，最大值可以为空，过滤条件包含最小值，不包含最大值。

上图中构建任务想要构建lineorder_flatten视图的Raw Cache数据，要写入Cache中的数据可以使用如下SQL表示：

```
SELECT * FROM lineorder_flatten
WHERE s_region == 'ASIA' OR s_region == 'AMERICA';
```

单击**Submit**，提交构建任务，返回到Cache详细页面，对应的构建任务会提交到Spark集群中执行，在Build Information中可以看到当前是否正在构建Cache的信息。在Cache构建完成后，可以在Build History中看到相关的信息。

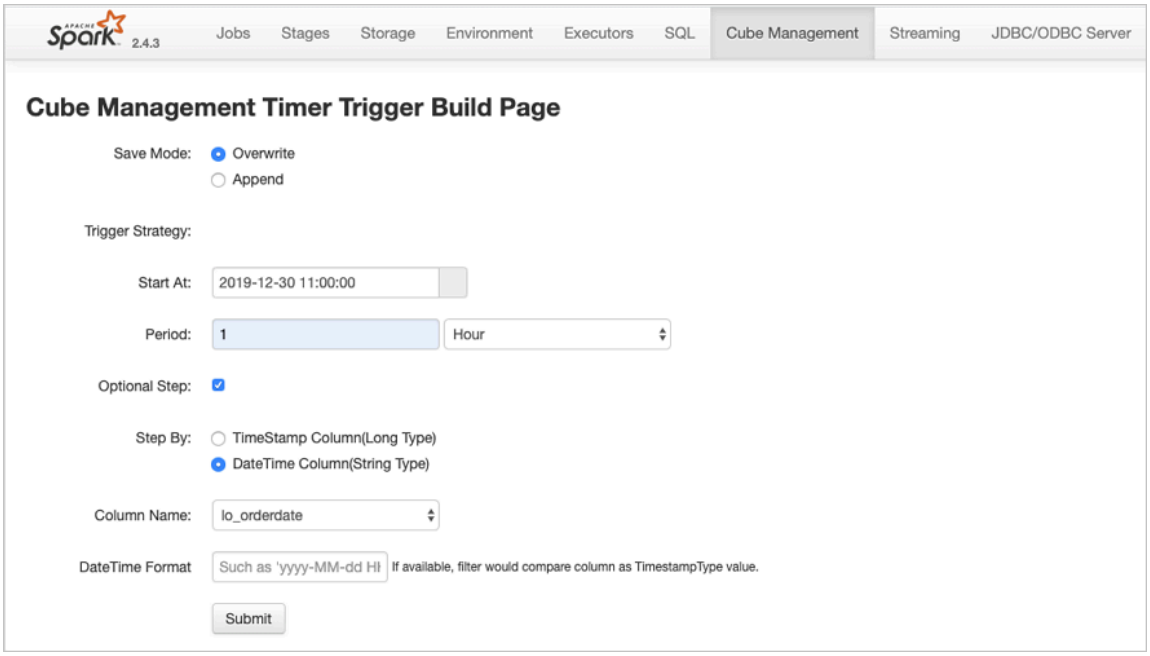
说明：

Cache数据由Spark任务写到一个指定目录中，和普通的Spark写表或者写目录一样，对于Parquet、Json、ORC等数据格式，并发构建同一个Cache可能导致Cache数据不准确，不

可用，应避免这种情况。如果无法避免并发构建、更新Cache，可以考虑使用delta等支持并发写的数据格式。

- Trigger Period Build。

定期更新功能可以方便用户设置自动更新Cache的策略，保持Cache数据和源表数据的一致。相关页面如下：



定期更新的相关用户选项如下：

参数	描述
Save Mode	支持Overwrite和Append两种模式。 - Overwrite：会覆盖之前曾经构建的Cache数据。 - Append：会新增数据到Cache中。
Trigger Strategy	触发策略，设置触发构建任务的开始时间和间隔时间。 - Start At：通过时间控件选择或者手工输入第一次触发构建任务的时间点，日期格式为yyyy-MM-dd hh:mm:ss。 - Period：设置触发构建任务的间隔时间。

参数	描述
Optional Step	设置每次触发构建任务的数据筛选条件，通过指定时间类型的字段，配合触发策略中的间隔时间，可以实现按照时间间隔增量的更新Cache。如果不选择，每次全量更新Cache。 - Step By：选择增量更新字段类型，只支持时间类型字段，包括Long类型的timestamp字段，以及指定dateformat信息的String类型字段。 - Column Name：增量更新字段名称。

在Cache详细页面中，可以看到当前设置的定期更新策略，用户可以随时通过Cancel Period Build取消定期更新。所有触发的构建任务信息在完成后也可以在Build History列表中看到。

说明：

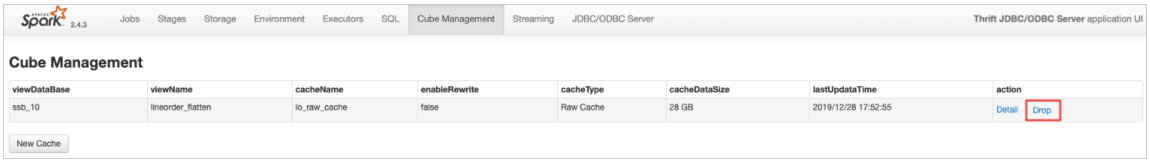
- 定期更新任务是Spark集群级别的，相关设置保存在SparkContext中，并由Spark Driver定期触发，当Spark集群关闭后，定期更新任务也随之关闭。
- 当前Spark集群所有的构建任务完成后，都会展示在Build History列表中，包含开始/结束时间、SaveMode、构建条件，任务最终状态等。Build History也是Spark集群级别的信息，当Spark集群关闭后，相关信息也随之释放。

3. 管理JindoCube。

   创建和构建JindoCube的Cache数据后，通过Cube Management的UI页面，可以对JindoCube的Cache数据进行进一步的管理。

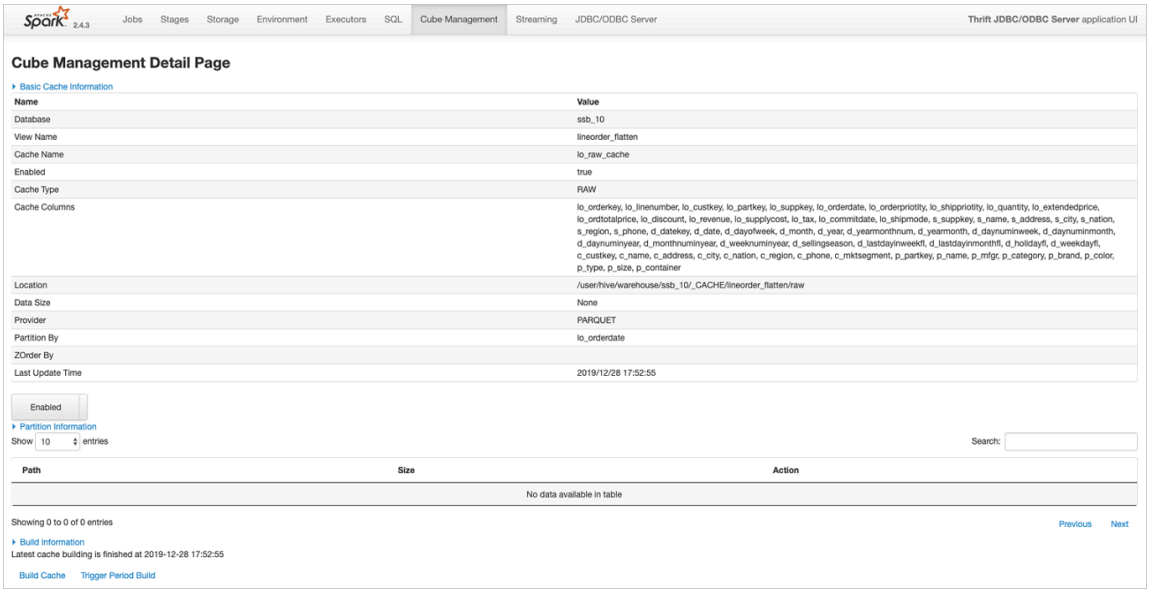
   - 删除cache。

     在JindoCube Cache列表页面，可以通过action列的**Drop**删除对应Cache，删除成功后，Cache的相关元数据和存储数据都会被清理。



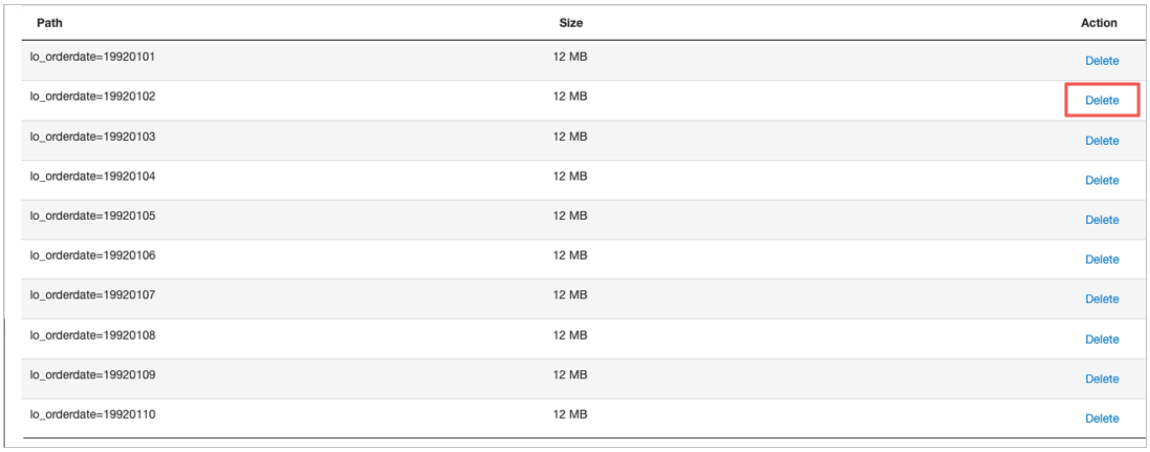
   - 开启或关闭Cache优化。

     JindoCube支持在Cache级别，设置是否允许用于Spark查询的优化，在Cache的详细页面，您可以通过基本信息中的**Enabled**或**Disabled**，启用或者停用该Cache，控制是否允许该Cache用于查询加速。



   - 删除分区数据。

     如果Cache的数据是按照分区存储的，当确认某些分区数据不再需要时，删除这些分区数据可以节省大量存储空间。在Cache的详细页面，分区Cache的相关分区会通过列表展示，用户可以通过**Delete**删除特定分区的数据。



Path	Size	Action
lo_orderdate=19920101	12 MB	Delete
lo_orderdate=19920102	12 MB	Delete
lo_orderdate=19920103	12 MB	Delete
lo_orderdate=19920104	12 MB	Delete
lo_orderdate=19920105	12 MB	Delete
lo_orderdate=19920106	12 MB	Delete
lo_orderdate=19920107	12 MB	Delete
lo_orderdate=19920108	12 MB	Delete
lo_orderdate=19920109	12 MB	Delete
lo_orderdate=19920110	12 MB	Delete

说明：

在删除Cache分区数据之前，请谨慎确认，确保该分区数据不会被使用。如果用户的查询经过优化需要用到该Cache被删除的分区数据，会导致错误的查询结果。

4. 查询优化。

目前JindoCube支持基于View的查询优化，当用户使用某个视图创建了Raw Cache或者Cube Cache后，后续的查询使用到了该视图，EMR Spark会在满足逻辑语义的前提下，尝试使用Cache重写查询的执行计划，新的执行计划直接访问Cache数据，从而加速查询速度。以如下场

景为例，lineorder_flatten视图是将lineorder和其他维度表关联之后的大宽表视图，其相关定义如下：

```
0: jdbc:hive2://localhost:10001/ssb_10> desc extended lineorder_flatten;
19/12/30 14:08:16 WARN [main] HiveConf: HiveConf of name hive.enforce.bucketing does not exist
19/12/30 14:08:16 WARN [main] HiveConf: HiveConf of name hive.support.sql11.reserved.keywords does not exist
+-------------------------------+----------------------------------------------------+----------+
|           col_name            |                     data_type                      | comment  |
+-------------------------------+----------------------------------------------------+----------+
| lo_orderkey                   | bigint                                             | NULL     |
| lo_linenumber                 | bigint                                             | NULL     |
| lo_custkey                    | int                                                | NULL     |
| lo_partkey                    | int                                                | NULL     |
| lo_suppkey                    | int                                                | NULL     |
| lo_orderdate                  | int                                                | NULL     |
| lo_orderpriotity              | string                                             | NULL     |
| lo_shippriotity               | int                                                | NULL     |
| lo_quantity                   | bigint                                             | NULL     |
| lo_extendedprice              | bigint                                             | NULL     |
| lo_ordtotalprice              | bigint                                             | NULL     |
| lo_discount                   | bigint                                             | NULL     |
| lo_revenue                    | bigint                                             | NULL     |
| lo_supplycost                 | bigint                                             | NULL     |
| lo_tax                        | bigint                                             | NULL     |
| lo_commitdate                 | int                                                | NULL     |
| lo_shipmode                   | string                                             | NULL     |
| s_suppkey                     | int                                                | NULL     |
| s_name                        | string                                             | NULL     |
| s_address                     | string                                             | NULL     |
| s_city                        | string                                             | NULL     |
| s_nation                      | string                                             | NULL     |
| s_region                      | string                                             | NULL     |
| s_phone                       | string                                             | NULL     |
| d_datekey                     | int                                                | NULL     |
| d_date                        | string                                             | NULL     |
| d_dayofweek                   | string                                             | NULL     |
| d_month                       | string                                             | NULL     |
| d_year                        | int                                                | NULL     |
| d_yearmonthnum                | int                                                | NULL     |
| d_yearmonth                   | string                                             | NULL     |
| d_daynuminweek                | int                                                | NULL     |
| d_daynuminmonth               | int                                                | NULL     |
| d_daynuminyear                | int                                                | NULL     |
| d_monthnuminyear              | int                                                | NULL     |
| d_weeknuminyear               | int                                                | NULL     |
| d_sellingseason               | string                                             | NULL     |
| d_lastdayinweekfl             | int                                                | NULL     |
| d_lastdayinmonthfl            | int                                                | NULL     |
| d_holidayfl                   | int                                                | NULL     |
| d_weekdayfl                   | int                                                | NULL     |
| c_custkey                     | int                                                | NULL     |
| c_name                        | string                                             | NULL     |
| c_address                     | string                                             | NULL     |
| c_city                        | string                                             | NULL     |
| c_nation                      | string                                             | NULL     |
| c_region                      | string                                             | NULL     |
| c_phone                       | string                                             | NULL     |
| c_mktsegment                  | string                                             | NULL     |
| p_partkey                     | int                                                | NULL     |
| p_name                        | string                                             | NULL     |
| p_mfgr                        | string                                             | NULL     |
| p_category                    | string                                             | NULL     |
| p_brand                       | string                                             | NULL     |
| p_color                       | string                                             | NULL     |
| p_type                        | string                                             | NULL     |
| p_size                        | int                                                | NULL     |
| p_container                   | string                                             | NULL     |
|                               |                                                    |          |
| # Detailed Table Information  |                                                    |          |
| Database                      | ssb_10                                             |          |
| Table                         | lineorder_flatten                                  |          |
| Owner                         | hadoop                                             |          |
| Created Time                  | Sat Dec 28 17:30:02 CST 2019                       |          |
| Last Access                   | Thu Jan 01 08:00:00 CST 1970                       |          |
| Created By                    | Spark 2.2 or prior                                 |          |
| Type                          | VIEW                                               |          |
| View Text                     | SELECT `lineorder`.`lo_orderkey`, `lineorder`.`lo_linenumber`, `lineorder`.`lo_custkey`, `lineorder`.`lo_partkey`, `
lineorder`.`lo_suppkey`, `lineorder`.`lo_orderdate`, `lineorder`.`lo_orderpriotity`, `lineorder`.`lo_shippriotity`, `lineorder`.`lo_quantity`, `lineor
der`.`lo_extendedprice`, `lineorder`.`lo_ordtotalprice`, `lineorder`.`lo_discount`, `lineorder`.`lo_revenue`, `lineorder`.`lo_supplycost`, `lineorder`
.`lo_tax`, `lineorder`.`lo_commitdate`, `lineorder`.`lo_shipmode`, `supplier`.`s_suppkey`, `supplier`.`s_name`, `supplier`.`s_address`, `supplier`.`s_
city`, `supplier`.`s_nation`, `supplier`.`s_region`, `supplier`.`s_phone`, `dates`.`d_datekey`, `dates`.`d_date`, `dates`.`d_dayofweek`, `dates`.`d_mo
nth`, `dates`.`d_year`, `dates`.`d_yearmonthnum`, `dates`.`d_yearmonth`, `dates`.`d_daynuminweek`, `dates`.`d_daynuminmonth`, `dates`.`d_daynuminyear`
, `dates`.`d_monthnuminyear`, `dates`.`d_weeknuminyear`, `dates`.`d_sellingseason`, `dates`.`d_lastdayinweekfl`, `dates`.`d_lastdayinmonthfl`, `dates`
.`d_holidayfl`, `dates`.`d_weekdayfl`, `customer`.`c_custkey`, `customer`.`c_name`, `customer`.`c_address`, `customer`.`c_city`, `customer`.`c_nation`
, `customer`.`c_region`, `customer`.`c_phone`, `customer`.`c_mktsegment`, `part`.`p_partkey`, `part`.`p_name`, `part`.`p_mfgr`, `part`.`p_category`, `
part`.`p_brand`, `part`.`p_color`, `part`.`p_type`, `part`.`p_size`, `part`.`p_container` FROM `ssb_10`.`lineorder`, `ssb_10`.`supplier`, `ssb_10`.`da
tes`, `ssb_10`.`customer`, `ssb_10`.`part`
WHERE `lineorder`.`lo_orderdate` = `dates`.`d_datekey` AND `lineorder`.`lo_custkey` = `customer`.`c_custkey` AND `lineorder`.`lo_suppkey` = `supplier`
.`s_suppkey` AND `lineorder`.`lo_partkey` = `part`.`p_partkey` |          |
| Table Properties              | [transient_lastDdlTime=1577677599]                 |          |
| Serde Library                 | org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe |          |
| InputFormat                   | org.apache.hadoop.mapred.TextInputFormat           |          |
| OutputFormat                  | org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat |          |
| Storage Properties            | [serialization.format=1]                           |          |
+-------------------------------+----------------------------------------------------+----------+
73 rows selected (0.057 seconds)
0: jdbc:hive2://localhost:10001/ssb_10>
```

基于lineorder_flatten视图简单查询的执行计划如下：

```
0: jdbc:hive2://localhost:10001/ssb_10> explain select * from lineorder_flatten where c_region = 'ASIA' limit 10;
19/12/30 14:19:32 WARN [main] HiveConf: HiveConf of name hive.enforce.bucketing does not exist
19/12/30 14:19:32 WARN [main] HiveConf: HiveConf of name hive.support.sql11.reserved.keywords does not exist
+----------------------------------------------------+
|                        plan                        |
+----------------------------------------------------+
| == Physical Plan ==
CollectLimit 10
+- *(11) SortMergeJoin [lo_partkey#313], [p_partkey#359], Inner
   :- *(8) Sort [lo_partkey#313 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(lo_partkey#313, 200)
   :     +- *(7) SortMergeJoin [lo_custkey#312], [c_custkey#351], Inner
   :        :- *(4) Sort [lo_custkey#312 ASC NULLS FIRST], false, 0
   :        :  +- Exchange hashpartitioning(lo_custkey#312, 200)
   :        :     +- *(3) BroadcastHashJoin [lo_orderdate#315], [d_datekey#334], Inner, BuildRight
   :        :        :- *(3) BroadcastHashJoin [lo_suppkey#314], [s_suppkey#327], Inner, BuildRight
   :        :        :  :- *(3) Filter (((isnotnull(lo_suppkey#314) && isnotnull(lo_orderdate#315)) && isnotnull(lo_custkey#312)) && isnotnull(lo_part
key#313))
   :        :        :  :  +- Scan hive ssb_10.lineorder [lo_orderkey#310L, lo_linenumber#311L, lo_custkey#312, lo_partkey#313, lo_suppkey#314, lo_ord
erdate#315, lo_orderpriotity#316, lo_shippriotity#317, lo_quantity#318L, lo_extendedprice#319L, lo_ordtotalprice#320L, lo_discount#321L, lo_revenue#32
2L, lo_supplycost#323L, lo_tax#324L, lo_commitdate#325, lo_shipmode#326], HiveTableRelation `ssb_10`.`lineorder`, org.apache.hadoop.hive.serde2.lazy.L
azySimpleSerDe, [lo_orderkey#310L, lo_linenumber#311L, lo_custkey#312, lo_partkey#313, lo_suppkey#314, lo_orderdate#315, lo_orderpriotity#316, lo_ship
priotity#317, lo_quantity#318L, lo_extendedprice#319L, lo_ordtotalprice#320L, lo_discount#321L, lo_revenue#322L, lo_supplycost#323L, lo_tax#324L, lo_c
ommitdate#325, lo_shipmode#326]
   :        :        :  +- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)))
   :        :        :     +- *(1) Filter isnotnull(s_suppkey#327)
   :        :        :        +- Scan hive ssb_10.supplier [s_suppkey#327, s_name#328, s_address#329, s_city#330, s_nation#331, s_region#332, s_phone#
333], HiveTableRelation `ssb_10`.`supplier`, org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, [s_suppkey#327, s_name#328, s_address#329, s_city#330
, s_nation#331, s_region#332, s_phone#333]
   :        :        +- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint)))
   :        :           +- *(2) Filter isnotnull(d_datekey#334)
   :        :              +- Scan hive ssb_10.dates [d_datekey#334, d_date#335, d_dayofweek#336, d_month#337, d_year#338, d_yearmonthnum#339, d_yearm
onth#340, d_daynuminweek#341, d_daynuminmonth#342, d_daynuminyear#343, d_monthnuminyear#344, d_weeknuminyear#345, d_sellingseason#346, d_lastdayinweek
fl#347, d_lastdayinmonthfl#348, d_holidayfl#349, d_weekdayfl#350], HiveTableRelation `ssb_10`.`dates`, org.apache.hadoop.hive.serde2.lazy.LazySimpleSe
rDe, [d_datekey#334, d_date#335, d_dayofweek#336, d_month#337, d_year#338, d_yearmonthnum#339, d_yearmonth#340, d_daynuminweek#341, d_daynuminmonth#34
2, d_daynuminyear#343, d_monthnuminyear#344, d_weeknuminyear#345, d_sellingseason#346, d_lastdayinweekfl#347, d_lastdayinmonthfl#348, d_holidayfl#349,
 d_weekdayfl#350]
   :        +- *(6) Sort [c_custkey#351 ASC NULLS FIRST], false, 0
   :           +- Exchange hashpartitioning(c_custkey#351, 200)
   :              +- *(5) Filter ((isnotnull(c_region#356) && (c_region#356 = ASIA)) && isnotnull(c_custkey#351))
   :                 +- Scan hive ssb_10.customer [c_custkey#351, c_name#352, c_address#353, c_city#354, c_nation#355, c_region#356, c_phone#357, c_mk
tsegment#358], HiveTableRelation `ssb_10`.`customer`, org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, [c_custkey#351, c_name#352, c_address#353, c
_city#354, c_nation#355, c_region#356, c_phone#357, c_mktsegment#358]
   +- *(10) Sort [p_partkey#359 ASC NULLS FIRST], false, 0
      +- Exchange hashpartitioning(p_partkey#359, 200)
         +- *(9) Filter isnotnull(p_partkey#359)
            +- Scan hive ssb_10.part [p_partkey#359, p_name#360, p_mfgr#361, p_category#362, p_brand#363, p_color#364, p_type#365, p_size#366, p_conta
iner#367], HiveTableRelation `ssb_10`.`part`, org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, [p_partkey#359, p_name#360, p_mfgr#361, p_category#3
62, p_brand#363, p_color#364, p_type#365, p_size#366, p_container#367] |
+----------------------------------------------------+
1 row selected (0.435 seconds)
0: jdbc:hive2://localhost:10001/ssb_10>
```

在为line order_flatten视图创建Raw Cache并构建完成后，执行相同查询，EMR Spark会自动使用Cache数据优化执行计划，优化后的执行计划如下：

```
0: jdbc:hive2://localhost:10001/ssb_10> explain select * from lineorder_flatten where c_region = 'ASIA' limit 10;
19/12/30 14:17:47 INFO [main] HiveConf: Found configuration file file:/etc/ecm/spark-conf-2.4.3-hadoop2.8-1.4.2/hive-site.xml
19/12/30 14:17:47 WARN [main] HiveConf: HiveConf of name hive.enforce.bucketing does not exist
19/12/30 14:17:47 WARN [main] HiveConf: HiveConf of name hive.support.sql11.reserved.keywords does not exist
+----------------------------------------------------+
|                        plan                        |
+----------------------------------------------------+
| == Physical Plan ==
CollectLimit 10
+- *(1) Project [lo_orderkey#128L, lo_linenumber#129L, lo_custkey#130, lo_partkey#131, lo_suppkey#132, lo_orderdate#133, lo_orderpriotity#134, lo_ship
priotity#135, lo_quantity#136L, lo_extendedprice#137L, lo_ordtotalprice#138L, lo_discount#139L, lo_revenue#140L, lo_supplycost#141L, lo_tax#142L, lo_c
ommitdate#143, lo_shipmode#144, s_suppkey#145, s_name#146, s_address#147, s_city#148, s_nation#149, s_region#150, s_phone#151, ... 34 more fields]
   +- *(1) Filter (isnotnull(c_region#174) && (c_region#174 = ASIA))
      +- *(1) FileScan parquet [lo_orderkey#128L,lo_linenumber#129L,lo_custkey#130,lo_partkey#131,lo_suppkey#132,lo_orderpriotity#134,lo_shippriotity#
135,lo_quantity#136L,lo_extendedprice#137L,lo_ordtotalprice#138L,lo_discount#139L,lo_revenue#140L,lo_supplycost#141L,lo_tax#142L,lo_commitdate#143,lo_
shipmode#144,s_suppkey#145,s_name#146,s_address#147,s_city#148,s_nation#149,s_region#150,s_phone#151,d_datekey#152,... 34 more fields] Batched: true,
Format: Parquet, Location: InMemoryFileIndex[/user/hive/warehouse/ssb_10/_CACHE/lineorder_flatten/raw], PartitionCount: 2406, PartitionFilters: [], Pu
shedFilters: [IsNotNull(c_region), EqualTo(c_region,ASIA)], ReadSchema: struct<lo_orderkey:bigint,lo_linenumber:bigint,lo_custkey:int,lo_partkey:int,lo
_suppkey:int,lo_or... |
+----------------------------------------------------+
```

可以看到，优化后的执行计划省去了lineorder_flatten视图的所有计算逻辑，直接访问HDFS中Cache的数据。

注意事项

1. JindoCube并不保证Cache数据和源表数据的一致性，而是需要用户通过手工触发或者设置定期策略触发更新任务同步Cache中的数据，用户需要根据查询对于数据一致性的需求，触发Cache的更新任务。
2. 在对查询的执行计划进行优化的时候，JindoCube根据视图的元数据判断是否可以使用Cache优化查询的执行计划。优化后，如果Cache的数据不完整，可能会影响查询结果的完整性或正确性。可能导致Cache数据不完整的情况包括：用户在Cache详情页主动删除查询需要的Cache

Partition数据，构建、更新Cache时指定的过滤条件过滤掉了查询需要的数据，查询需要的数据还未及时更新到Cache等。