ALIBABA CLOUD

Alibaba Cloud

E-MapReduce

最佳实践

文档版本：20201029

阿里云

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. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，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
危险	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。	危险 重置操作将丢失用户配置数据。
警告	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。
注意	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。
说明	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	命令或代码。	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.集群管理

1.1. 使用YARN CGroups功能对CPU进行控制测试

CGroups（Control Groups）是Linux内核提供的一种功能，用来控制、限制与隔离一个进程组群的资源，包括CPU、内存和磁盘输入输出等资源。

概述

YARN中集成了CGroups的功能，使得NodeManger可以对Container的CPU的资源使用进行控制，例如可以对单个Container的CPU使用进行控制，也可以对NodeManger管理的总CPU进行控制。

开启YARN CGroups功能

> **说明** E-MapReduce集群中的YARN默认没有开启CGroups的功能，需要用户根据需求自行开启。

1. 登录阿里云 E-MapReduce 控制台。
2. 进入YARN的**集群服务**页面。
   i. 单击上方的**集群管理**页签。
   ii. 在**集群管理**页面，单击相应集群所在行的**集群ID**。
   iii. 在左侧导航栏，单击**集群服务 > YARN**。
3. 启用CGroups。
   i. 单击右上角的**操作 > 启用 CGroups**。
   ii. 在**执行集群操作**页面，根据需求设置相应参数。
   iii. 单击**确定**。
   iv. 在**确认**页面，单击**确定**。
4. 重启NodeManger。
   i. 单击右上角的**操作 > 重启NodeManger**。
   ii. 在**执行集群操作**页面，根据需求设置相应参数。
   iii. 单击**确定**。
   iv. 在**确认**页面，单击**确定**。

控制参数

在开启了CGroups功能的前提下，可以通过调节YARN中的参数来控制CPU的资源使用行为：

1. 登录阿里云 E-MapReduce 控制台。
2. 进入YARN的**集群服务**页面。
   i. 单击上方的**集群管理**页签。
   ii. 在**集群管理**页面，单击相应集群所在行的**集群ID**。
   iii. 在左侧导航栏，单击**集群服务 > YARN**。
3. 调节参数。

i. 单击**配置**页签。

ii. 调节如下参数。

参数	描述
yarn.nodemanager.resource.percentage-physical-cpu-limit	NodeManager管理的所有Container使用CPU的阈值，默认100%。 **说明** 任何场景下，NodeManger管理的Container的CPU都不能超过yarn.nodemanager.resource.percentage-physical-cpu-limit设置的比例。
yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage	对Container的CPU使用资源是否严格按照被分配的比例来进行控制。默认是false，即Container可以使用空闲的CPU。

总Container的CPU控制测试

通过调节**yarn.nodemanager.resource.percentage-physical-cpu-limit**参数，以控制NodeManager管理的所有Container的CPU使用率。

下面集群配置为3台4核16GB，其中2台NodeManger，1台ResourceManager，分别设置该值以10、30和50为例，在YARN中运行一个hadoop pi作业，观察NodeManger所在机器的CPU的使用率。

说明 下图中的 `%CPU` 表示进程占用单个核的比例；`%CPU(s)` 表示所有用户进程占总CPU的比例。

- 设置参数为10。

  如上图所示，`%CPU(s)` 接近10%；`%CPU` 表示所有的test用户的Container进程加起来（7%+5.3%+5%+4.7%+4.7%+4.3%+4.3%+4%+2%=41.3%=0.413个核，约等于10%*4core=0.4核，即4个核的10%比例）。

- 设置参数为30。

  如上图所示，`%CPU(s)` 接近30%；`%CPU` 表示所有的test用户的Container进程加起来（19%+18.3%+18.3%+17%+16.7%+16.3%+14.7%+12%=132.3%=1.323个核，约等于30%*4core=1.2核，即4个核的30%比例）。

- 设置参数为50。

  如上图所示，`%CPU(s)` 接近50%；`%CPU` 表示所有的test用户的Container进程加起来（65.1%+60.1%+43.5%+20.3%+3.7%+2%=194.7%=1.947个核，约等于50%*4core=2核，即4个核的50%比例）。

Container间的CPU控制测试

NodeManger上面启动多个Container，所有这些Container对CPU资源的占用不超过总Container的CPU控制测试中设置**yarn.nodemanager.resource.percentage-physical-cpu-limit**的阈值，NodeManger有共享模式（share）和严格模式（strict）两种方式，来管理和控制多个Container之间的CPU使用率，这两种方式通过参数**yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage**控制。

- 共享模式（share）

  当**yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage**设置为*false*时，即为共享模式（默认为false）。在这种模式下，Container除了实际被需要分配的CPU资源外，还可以利用空闲的CPU资源。

  例如，设置**yarn.nodemanager.resource.percentage-physical-cpu-limit**为*50*，设置**yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage**为*false*。NodeManger所在节点是4核，那么该Container申请按比例被分配的cpu资源为（1vcore÷8vcore）*（4core*50%）=0.25核，但是如果CPU有空闲，理论上该Container可以占满NodeManger管理的上限4core*50%=2核。

  

  上图可以看出，test用户的多个Container进程占用CPU核数的比例相差很大（65%=0.65核；60.1%=0.61核； 3.7%=0.37核等），即单个Container的CPU使用没有被严格限制在（1vcore÷8vcore）*（4core*50%）=0.25核。

- 严格模式（strict）

  当**yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage**设置为*true*时，即为严格模式（strict）。在这种模式下，Container只能使用被需要分配的CPU资源，即使CPU有空闲也不能使用。

  例如，设置**yarn.nodemanager.resource.percentage-physical-cpu-limit**为*50*，设置**yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage**为*true*。

  

  上图可以看出，test用户的多个Container进程占用CPU核数均约在0.25核（26.6%=0.266核；24.9%=0.249核），即该Container实际应该被分配的CPU使用率（1vcore÷8vcore）*（4core*50%）=0.25核。

1.2. OSS数据权限隔离

本文介绍如何使用访问控制RAM（Resource Access Management），对不同子账号的OSS数据进行隔离。

前提条件

已获取云账号。

背景信息

E-MapReduce支持使用RAM来隔离不同子账号的数据。

操作步骤

1.
2. 创建RAM用户。
   i. 在左侧导航栏中，单击**人员管理 > 用户**。

ii. 单击**创建用户**。

> **说明** 可一次性创建多个RAM用户。

iii. 输入**登录名称**和**显示名称**。

iv. 在**访问方式**区域，选中**控制台密码登录**或**编程访问**。

- **控制台密码登录**：完成对登录安全的基本设置，包括自动生成或自定义登录密码、是否要求下次登录时重置密码以及是否要求开启多因素认证。
- **编程访问**：自动为RAM用户生成访问密钥（AccessKey），支持通过API或其他开发工具访问阿里云。

> **说明** 为了保障账号安全，建议仅为RAM用户选择一种登录方式，避免RAM用户离开组织后仍可以通过访问密钥访问阿里云资源。

v. 单击**确定**。

3. 新建权限策略。

RAM除了提供常用的默认权限策略外，还支持让您自定义权限策略，使得授权更加灵活。根据不同需要，您可创建多个权限策略。

i. 在左侧导航栏中，**权限管理 > 权限策略管理**。

ii. 单击**创建权限策略**。

iii. 填写**策略名称**。

iv. 选中**脚本配置**。**脚本配置**方法请参见语法结构编辑策略内容。 本例分别按照以下两个脚本示例来创建两个权限策略：

测试环境（test-bucket）	生产环境（prod-bucket）

<pre>{
"Version": "1",
"Statement": [
{
"Effect": "Allow",
"Action": [
"oss:ListBuckets"
],
"Resource": [
"acs:oss:*:*:*"
]
},
{
"Effect": "Allow",
"Action": [
"oss:Listobjects",
"oss:GetObject",
"oss:PutObject",
"oss:DeleteObject"
],
"Resource": [
"acs:oss:*:*:test-bucket",
"acs:oss:*:*:test-bucket/*"
]
}
]
}</pre>
	<pre>{
	"Version": "1",
	"Statement": [
	{
	"Effect": "Allow",
	"Action": [
	"oss:ListBuckets"
	],
	"Resource": [
	"acs:oss:*:*:*"
	]
	},
	{
	"Effect": "Allow",
	"Action": [
	"oss:Listobjects",
	"oss:GetObject",
	"oss:PutObject"
	],
	"Resource": [
	"acs:oss:*:*:prod-bucket",
	"acs:oss:*:*:prod-bucket/*"
	]
	}
	]
	}</pre>

按上述脚本示例进行权限隔离后，RAM用户在E-MapReduce控制台的限制如下：

- 在创建集群、创建作业和创建工作流的OSS文件页面，可以看到所有的bucket，但是只能进入被授权的bucket。
- 只能看到被授权的bucket下的内容，无法看到其他bucket内的内容。

- 作业中只能读写被授权的bucket，读写未被授权的bucket会报错。

v. 单击**确定**。

4. （可选）为RAM用户授权。 如果创建RAM用户时未开启控制台登录权限，则您可按以下方法进行开启。
   i. 单击左侧导航栏的**人员管理 > 用户**。
   ii. 单击待授权RAM用户所在行的**添加权限**。
   iii. 单击需要授予RAM用户的权限策略，单击**确定**。
   iv. 单击**完成**。
5. （可选）开启RAM用户的控制台登录权限。 如果创建RAM用户时未开启控制台登录权限，则您可按以下方法进行开启。
   i. 单击左侧导航栏的**人员管理 > 用户**。
   ii. 单击目标RAM用户的用户登录名称。
   iii. 在**控制台登录管理**区域，单击**修改登录设置**。
   iv. **控制台密码登录**选中**开启**。
   v. 单击**确定**。
6. 通过RAM用户登录E-MapReduce控制台。
   i. RAM用户登录控制台。
   ii. 登录控制台后，选择**E-MapReduce**产品即可。

2.数据开发

2.1. 使用E-MapReduce采集Kafka客户端Metrics数据

本文介绍通过使用E-MapReduce产品从Kafka客户端采集Metrics数据从而有效地进行性能监控。

背景信息

Kafka提供了一套非常完善的Metrics数据，覆盖Broker、Consumer、Producer、Stream以及Connect 。E-MapReduce通过Ganglia收集了Kafka Broker Metrics信息，可以很好地监控Broker运行状态。Kafka应用包括Kafka Broker和Kafka客户端这两个角色，当发生读写性能问题时，仅从 Broker角度难以发现问题，可以结合客户端的运行状况联合分析。

实现原理

- 如何采集Metrics

  Metrics Kafka默认提供了包含JmxReporter的Metrics Reporter插件扩展功能，即我们已经可以通过 JMX 工具来查看 Kafka 的 Metrics 。所以，我们可以自己实现一套 Metrics Reporter（实现org.apache.kafka.common.metrics.MetricsReporter），来自定义获取这些 Metrics。

- 如何存放Metrics

  以上我们实现了自定义采集 Kafka Metrics，还需要选择一个存储系统将这些 Metrics存储起来，以便后续的使用和分析。考虑到 Kafka 自身就是一种存储系统，我们可以将 Metrics 存储到 Kafka 中，这样做有以下几种优势：

  - 无需第三方存储系统支持。
  - 数据很方便地接入到其他系统中。

  所以，完整的客户端 Metrics 采集方案如下图所示。

环境准备

- 限制条件
  - 只支持 Java 类客户端程序。
  - 只支持 0.10 之后版本 Kafka 客户端。
- 无需自己编译，EMR 已经向 Maven 发布了 jar 包，直接下载即可。
- 本文使用阿里云 EMR 服务自动化搭建 Kafka 集群，详细过程请参见创建集群。

  本文使用的 EMR Kafka 版本信息如下：

  - EMR 版本：EMR-3.12.1。
  - 集群类型：Kafka。
  - 软件信息：Kafka-Manager (1.3.3.16)/Kafka (2.11-1.0.1)/ZooKeeper (3.4.12)/ Ganglia (3.7.2)。
  - Kafka 集群使用专有网络，区域为华东 1（杭州），主实例组 ECS 计算资源配置公网及内网 IP，具体配置如下所示。

- 参数配置。

配置项	说明
metric.reporters	使用 EMR 的实现：*org.apache.kafka.clients.reporter.EMRClientMetricsReporter*。
emr.metrics.reporter.bootstrap.servers	Metrics 存储 Kafka 集群的 bootstrap.servers。
emr.metrics.reporter.zookeeper.connect	Metrics 存储 Kafka 集群的 Zookeeper 地址。

- 如何加载 Metrics。
  - 将 emr-kafka-client-metrics 的 jar 包放在客户端程序的 Classpath 可以加载到的机器上即可。
  - 直接将 emr-kafka-client-metrics 依赖打进客户端程序 jar 包中。

实施步骤

1. 下载最新版本 emr-kafka-client-metrics 包。

```
wget http://central.maven.org/maven2/com/aliyun/emr/emr-kafka-client-metrics/1.4.3/emr-kafka-client-metrics-1.4.3.jar
```

2. 将 emr-kafka-client-metrics 包放到 Kafka 的 lib 中。

```
cp emr-kafka-client-metrics-1.4.3.jar /usr/lib/kafka-current/libs/
```

3. 创建一个测试 Topic。

```
kafka-topics.sh --zookeeper emr-header-1:2181/kafka-1.0.1 --partitions 10 --replication-factor 2 --topic test-metrics  --create
```

4. 向测试 Topic 写入数据，这里我们将 producer 的配置项配置到本地文件 *client.conf* 中。

```
## client.conf:
metric.reporters=org.apache.kafka.clients.reporter.EMRClientMetricsReporter
emr.metrics.reporter.bootstrap.servers=emr-worker-1:9092
emr.metrics.reporter.zookeeper.connect=emr-header-1:2181/kafka-1.0.1
bootstrap.servers=emr-worker-1:9092
## 命令:
kafka-producer-perf-test.sh --topic test-metrics --throughput 1000 --num-records 100000
--record-size 1024 --producer.config client.conf
```

5. 查看这次客户端的 Metrics，注意默认的 Metrics Topic 是 *_emr-client-metrics*。

```
kafka-console-consumer.sh --topic _emr-client-metrics --bootstrap-server emr-worker-1:9092
--from-beginning
```

返回的消息如下所示。

```
{prefix=kafka.producer, client.ip=192.168.xxx.xxx, client.process=25536@emr-header-1.cluster-xxxx,
attribute=request-rate, value=894.4685104965012, timestamp=1533805225045, group=producer-metric
s,
tag.client-id=producer-1}
```

参数说明：

字段名	说明
client.ip	客户端所在机器 IP。
client.process	客户端程序进程 ID。
attribute	Metric 属性名。
value	Metric 值。
timestamp	Metric 采集的时间戳。
tag.xxx	Metric 其他 tag 信息。

2.2. 使用E-MapReduce处理离线作业

本文介绍使用E-MapReduce（以下简称EMR） 产品从OSS服务上读取数据 ，并进行数据采集、分析等一系列离线数据处理的使用场景。

背景信息

EMR集群适用于多种场景。EMR本质就是Hadoop和Spark的集群服务，所以Hadoop ecosystem以及Spark能够支持的场景，EMR都可以支持。您完全可以将EMR集群使用的阿里云ECS主机视为自己专属的物理主机。

大数据处理目前比较常见的有两种方法：

- 离线处理：只是希望得到数据的分析结果，对处理的时间要求不严格，例如批量数据处理，用户将数据传输到OSS服务，OSS服务作为EMR产品的输入输出，利用MapReduce、Hive、Pig、Spark处理离线数据。
- 在线处理：对于数据的分析结果在时间上有比较严格的要求，例如实时流式数据处理，使用Spark Streaming对消息数据进行处理，与Spark mllib、GrapX、SQL深度整合。

下面将使用EMR产品运行一个word count离线作业。

基本架构

OSS -> EMR -> Hadoop MapReduce

上述链路主要包含两个过程：

1. 把数据存储到OSS服务中。
2. 通过EMR服务将OSS中的数据读取出来，进行分析。

环境准备

- 本文以Windows环境为例，请确保Git、Maven、 Java已经安装并配置成功。
- 本文使用阿里云EMR服务自动化搭建Hadoop集群，详细步骤请参见创建集群。
  - EMR版本：EMR-3.12.1

- EMR版本：EMR-3.12.1
- 集群类型：Hadoop
- 软件信息：HDFS 2.7.2、YARN 2.7.2、Hive 2.3.3、Ganglia 3.7.2、Spark 2.3.1、HUE 4.1.0、Zeppelin 0.8.0、Tez 0.9.1、Sqoop 1.4.7、Pig 0.14.0、ApacheDS 2.0.0、Knox 0.13.0
- Hadoop集群使用专有网络，区域为华东 1（杭州），主实例组ECS计算资源配置公网及内网IP，**高可用**选择为**否**（非HA模式）。

操作步骤

1. 下载示例代码到本地 。

   在本地打开git bash运行clone 命令：

   ```
   git clone https://github.com/aliyun/aliyun-emapreduce-demo.git
   ```

   执行 `mvn install` 进行编译。

2. 创建OSS存储空间，详细步骤请参见创建存储空间。

   > **说明** Bucket应与E-MapReduce集群在同一个区域。

3. 上传JAR包和资源文件。

   i. 登录 OSS管理控制台 ，单击**文件管理**。

   ii. 单击**上传文件**，上传*aliyun-emapreduce-demo/resources*目录下的资源文件以及*aliyun-emapreduce-demo/target*目录下的JAR文件。

4. 创建工作流项目。

   详细步骤请参见工作流项目管理。

5. 创建作业。

   详细步骤请参见作业编辑，这里我们以MapReduce为例，分别填写所属项目、作业名称、作业描述、并选择作业类型MR。

   

6. 配置作业内容，单击**运行**。

   

   - 关于OSS使用，请参见OSS 参考使用说明。
   - 关于各类作业的具体开发，请参见作业部分。

   > **说明**
   > - OSS输出路径如果已经存在，在执行作业时会报错。
   > - 插入OSS路径时，如果选择OSSREF文件前缀，系统会把OSS文件下载到集群本地，并添加到classpath中。
   > - 当前所有操作都只支持标准存储类型的OSS。

查看日志

作业运行成功后，您可以在页面下方的运行记录页签中查看作业的运行日志。单击**详情**跳转到运行记录中该作业的详细日志页面，可以查看作业的提交日志、YARN Container日志。您也可以通过SSH方式登录到集群查看相关日志，详细步骤请参见使用SSH连接主节点。

总结

至此，我们成功在E-MapReduce上部署一套Hadoop集群，将OSS服务作为数据源输入，并运行MapReduce作业消费OSS上数据。E-MapReduce支持多种类型作业的开发，例如，Storm作业消费Kafka数据， Spark Streaming和Flink组件，同样可以方便地在Hadoop集群上运行，处理Kafka数据。

2.3. 使用E-MapReduce提交Storm作业处理Kafka数据

本文介绍如何使用阿里云E-MapReduce（以下简称EMR） 部署Storm集群和Kafka集群，并运行Storm作业消费Kafka数据。

环境准备

本文选择在杭州Region进行测试，版本选择EMR-3.8.0，本次测试需要的组件版本有：

- Kafka：2.11_1.0.0
- Storm: 1.0.1

本文使用阿里云EMR服务自动化搭建Kafka集群，详细过程请参见创建集群。

- 创建Hadoop集群

  

- 创建Kafka集群

  

  > **说明**
  > - 如果使用经典网络，请注意将Hadoop集群和Kafka集群放置在同一个安全组下面，这样可以省去配置安全组，避免网络不通的问题。
  > - 如果使用VPC网络，请注意将Hadoop集群和Kafka集群放置在同一个VPC或VSwitch以及安全组下面，这样同样省去配置网路和安全组，避免网络不通。
  > - 如果您熟悉ECS的网络和安全组，可以按需配置。

- 配置Storm环境

  如果我们想在Storm上运行作业消费Kafka的话，集群初始环境下是会失败的，因为Storm运行环境缺少了必须的依赖包，包括curator-client、curator-framework、curator-recipes、json-simple、metrics-core、scala-library、zookeeper、commons-cli、commons-collections、commons-configuration、htrace-core、jcl-over-slf4j、protobuf-java、guava、hadoop-common、kafka-clients、kafka、storm-hdfs、storm-kafka。

  如果您再测试过程中引入了其他依赖，也一同添加在Storm lib中，具体操作如下：

  

  上述操作需要在Hadoop集群的每台机器执行一遍。执行完在E-MapReduce控制台重启Storm服务。

  

查看操作历史，待Storm重启完毕。

开发Storm和Kafka作业

- EMR已经提供了现成的示例代码，直接使用即可，地址如下：
  - e-mapreduce-demo
  - e-mapreduce-sdk
- Topic数据准备
  i. 登录到Kafka集群。
  ii. 创建一个test topic，分区数10，副本数2。

```
/usr/lib/kafka-current/bin/kafka-topics.sh --partitions 10 --replication-factor 2 --zookeeper emr-header-1:/kafka-1.0.0 --topic test --create
```

  iii. 向test topic写入100条数据。

```
/usr/lib/kafka-current/bin/kafka-producer-perf-test.sh --num-records 100 --throughput 10000 --record-size 1024 --producer-props bootstrap.servers=emr-worker-1:9092 --topic test
```

> **说明** 以上命令在kafka集群的emr-header-1节点执行，当然也可以客户端机器上执行。

- 运行Storm作业

  登录到Hadoop集群，将编译得到的*/target/shaded*目录下的*examples-1.1-shaded.jar*拷贝到集群emr-header-1上，这里以放在root根目录下面为例提交作业。

```
/usr/lib/storm-current/bin/storm jar examples-1.1-shaded.jar com.aliyun.emr.example.storm.StormKafkaSample test aaa.bbb.ccc.ddd hdfs://emr-header-1:9000 sample
```

- 查看作业运行
  - 查看Storm运行状态。

    查看集群上服务的WebUI有2种方式：
    - 通过Knox方式，请参见 Knox 使用说明。
    - SSH隧道，请参见通过SSH隧道方式访问开源组件Web UI。

    本文选择使用SSH隧道方式，访问地址：*http://localhost:9999/index.html*。可以看到我们刚刚提交的Topology。点进去可以看到执行详情：

- 查看HDFS输出
  - 查看HDFS文件输出。

```
[root@emr-header-1 ~]# hadoop fs -ls /foo/
-rw-r--r--   3 root hadoop     615000 2018-02-11 13:37 /foo/bolt-2-0-1518327393692.txt
-rw-r--r--   3 root hadoop     205000 2018-02-11 13:37 /foo/bolt-2-0-1518327441777.txt
[root@emr-header-1 ~]# hadoop fs -cat /foo/bolt-2-0-1518327441777.txt | wc -l
200
```

  - 向Kafka写120条数据。

```
[root@emr-header-1 ~]# /usr/lib/kafka-current/bin/kafka-producer-perf-test.sh --num-records 120 --throughput 10000 --record-size 1024 --producer-props bootstrap.servers=emr-worker-1:9092 --topic test
120 records sent, 816.326531 records/sec (0.80 MB/sec), 35.37 ms avg latency, 134.00 ms max latency, 35 ms 50th, 39 ms 95th, 41 ms 99th, 134 ms 99.9th.
```

  - 查看HDFS文件输出。

```
[root@emr-header-1 ~]# hadoop fs -cat /foo/bolt-2-0-1518327441777.txt | wc -l
320
```

总结

至此，我们成功实现了在E-MapReduce上部署一套Storm集群和一套Kafka集群，并运行Storm作业消费Kafka数据。当然，E-MapReduce也支持Spark Streaming和Flink组件，同样可以方便在Hadoop集群上运行，处理Kafka数据。

> **说明**
>
> 由于E-MapReduce没有单独的Storm集群类别，所以我们是创建的Hadoop集群，并安装了Storm组件。如果您在使用过程中用不到其他组件，可以很方便地在E-MapReduce管理控制台将那些组件停掉。这样，可以将Hadoop集群作为一个纯粹的Storm集群使用。

2.4. 在E-MapReduce上使用Intel Analytics Zoo进行深度学习

Analytics Zoo是由Intel开源，基于Apache Spark和Inte BigDL的大数据分析和AI平台，方便您开发基于大数据、端到端的深度学习应用。本文介绍了如何在阿里云E-MapReduce使用Analytics Zoo来进行深度学习。

系统要求

- JDK 8
- Spark 集群（推荐使用EMR支持的 Spark 2.x）
- Python-2.7（python 3.5，3.6 也支持）、pip

安装Analytics Zoo

1. Scala安装
   i. 下载pre-build版本

      可以从GitHub，analytics主页下载pre-build 版本。

   ii. 通过script build

      安装Apache Maven，设置Maven环境。

      ```
      export MAVEN_OPTS="-Xmx2g -XX:ReservedCodeCacheSize=512m"
      ```

      如果使用ECS机器进行编译，推荐修改Maven仓库mirror:

      ```
      <mirror>
        <id>nexus-aliyun</id>
        <mirrorOf>central</mirrorOf>
        <name>Nexus aliyun</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public</url>
      </mirror>
      ```

   iii. 下载Analytics Zoo release 版本，解压后在目录下运行。

      ```
      bash make-dist.sh
      ```

   iv. build结束后，在dist目录中包含了所有的运行环境。将dist目录放到EMR软件栈运行时统一目录。

      ```
      cp -r dist/ /usr/lib/analytics_zoo
      ```

2. Python安装

   Analytics Zoo支持pip安装和非pip安装，pip安装会安装pyspark、bigdl等，由于EMR集群已经安装了pyspark，通过pip安装有可能引起冲突，所以采用非pip安装。

   首先要运行:

   ```
   bash make-dist.sh
   ```

   进入pyzoo目录，安装analytcis zoo。

   ```
   python setup.py install
   ```

3. 设置环境变量

   在scala安装结束后将dist目录放到了EMR软件栈统一目录，然后设置环境变量。编辑*/etc/profile.d/analytics_zoo.sh*，增加以下命令。

   ```
   export ANALYTICS_ZOO_HOME=/usr/lib/analytics_zoo
   export PATH=$ANALYTICS_ZOO_HOME/bin:$PATH
   ```

   EMR已经设置了SPARK_HOME，所以无需再次设置。

使用Analytics Zoo

- 使用Spark来训练和测试深度学习模型。

- 使用Analytics Zoo来做文本分类，代码和说明在GitHub。根据说明下载必须的数据，提交以下命令。

```
spark-submit --master yarn \
--deploy-mode cluster --driver-memory 8g \
--executor-memory 20g --class com.intel.analytics.zoo.examples.textclassification.TextClassification \
/usr/lib/analytics_zoo/lib/analytics-zoo-bigdl_0.6.0-spark_2.1.0-0.2.0-jar-with-dependencies.jar --baseDir /news
```

- 查看Spark运行详情页面，具体步骤可参见通过SSH隧道方式访问开源组件Web UI或访问链接与端口。

同时查看日志，能够看到每个epoch的accuracy信息等。

```
INFO optim.DistriOptimizer$: [Epoch 2 9600/15107][Iteration 194][Wall Clock 193.266637037s] Trained 128 records in 0.958591653 seconds. Throughput is 133.52922 records/second. Loss is 0.74216986.
INFO optim.DistriOptimizer$: [Epoch 2 9728/15107][Iteration 195][Wall Clock 194.224064816s] Trained 128 records in 0.957427779 seconds. Throughput is 133.69154 records/second. Loss is 0.51025534.
INFO optim.DistriOptimizer$: [Epoch 2 9856/15107][Iteration 196][Wall Clock 195.189488678s] Trained 128 records in 0.965423862 seconds. Throughput is 132.58424 records/second. Loss is 0.553785.
INFO optim.DistriOptimizer$: [Epoch 2 9984/15107][Iteration 197][Wall Clock 196.164318688s] Trained 128 records in 0.97483001 seconds. Throughput is 131.30495 records/second. Loss is 0.5517549.
```

- 在Analytics Zoo中使用pyspark和Jupyter来进行深度学习训练。
  i. 安装Jupyter。

```
pip install jupyter
```

  ii. 使用以下命令启动。

```
jupyter-with-zoo.sh
```

  iii. 使用Analytics Zoo，推荐采用内置的Wide And Deep模型来进行，相关内容可参见 GitHub。
    a. 导入数据。

    b. 定义模型和优化器。

    c. 进行训练。

    d. 查看训练结果。

2.5. SparkSQL自适应执行

阿里云E-MapReduce-3.13.0版本的SparkSQL支持自适应执行功能，用来解决Reduce个数的动态调整/数据倾斜/执行计划的动态优化问题。

解决问题

SparkSQL自适应执行解决以下问题：

- Shuffle partition个数

  目前SparkSQL中reduce阶段的task个数取决于固定参数 spark.sql.shuffle.partition（默认值 200），一个作业一旦设置了该参数，它运行过程中的所有阶段的reduce个数都是同一个值。

  而对于不同的作业，以及同一个作业内的不同reduce阶段，实际的数据量大小可能相差很大，例如reduce阶段要处理的数据可能是10MB，也有可能是100GB, 如果使用同一个值对实际运行效率会产生很大影响，例如10MB的数据一个task就可以解决，如果 spark.sql.shuffle.partition使用默认值200的话，那么10MB的数据就要被分成200个task处理，增加了调度开销，影响运行效率。

  SparkSQL自适应框架可以通过设置shuffle partition的上下限区间，在这个区间内对不同作业不同阶段的reduce个数进行动态调整。

  通过区间的设置，一方面可以大大减少调优的成本（不需要找到一个固定值），另一方面同一个作业内部不同reduce阶段的reduce个数也能动态调整。

  参数：

属性名称	默认值	备注
spark.sql.adaptive.enabled	false	自适应执行框架的开关。
spark.sql.adaptive.minNumPostShufflePartitions	1	reduce个数区间最小值。
spark.sql.adaptive.maxNumPostShufflePartitions	500	reduce个数区间最大值。
spark.sql.adaptive.shuffle.targetPostShuffleInputSize	67108864	动态调整reduce个数的 partition 大小依据，如设置64MB，则 reduce 阶段每个task最少处理64MB的数据。
spark.sql.adaptive.shuffle.targetPostShuffleRowCount	20000000	动态调整reduce个数的partition条数依据，如设置20000000则reduce阶段每个task最少处理20000000条的数据。

- 数据倾斜

  Join中会经常碰到数据倾斜的场景，导致某些task处理的数据过多，出现很严重的长尾。目前SparkSQL没有对倾斜的数据进行相关的优化处理。

  SparkSQL自适应框架可以根据预先的配置在作业运行过程中自动检测是否出现倾斜，并对检测到的倾斜进行优化处理。

  优化的主要逻辑是对倾斜的partition进行拆分由多个task来进行处理，最后通过union进行结果合并。

  支持的 Join 类型：

join类型	备注
Inner	左/右表均可处理倾斜。
Cross	左/右表均可处理倾斜。
LeftSemi	只对左表处理倾斜。
LeftAnti	只对左表处理倾斜。
LeftOuter	只对左表处理倾斜。
RightOuter	只对右表处理倾斜。

参数：

属性名称	默认值	备注
spark.sql.adaptive.enabled	false	自适应执行框架的开关。
spark.sql.adaptive.skewedJoin.enabled	false	倾斜处理开关。
spark.sql.adaptive.skewedPartitionFactor	10	当一个 partition的size大小或大于该值（所有parititon大小的中位数） 且大于spark.sql.adaptive.skewedPartitionSizeThreshold，或者parition的条数大于该值（所有parititon条数的中位数）且大于spark.sql.adaptive.skewedPartitionRowCountThreshold， 才会被当做倾斜的partition进行相应的处理。
spark.sql.adaptive.skewedPartitionSizeThreshold	67108864	倾斜的partition大小不能小于该值。
spark.sql.adaptive.skewedPartitionRowCountThreshold	10000000	倾斜的partition条数不能小于该值。
spark.shuffle.statistics.verbose	false	打开后MapStatus会采集每个partition条数的信息，用于倾斜处理。

- Runtime执行计划优化

  SparkSQL的Catalyst优化器会将sql语句转换成物理执行计划，然后真正运行物理执行计划。但是Catalyst转换物理执行计划的过程中，由于缺少Statistics统计信息，或者Statistics统计信息不准等原因，实际转换的物理执行计划可能并不是最优的，例如转换为SortMergeJoinExec，但实际BroadcastJoin更合适。

  SparkSQL自适应执行框架会在物理执行计划真正运行的过程中，动态的根据shuffle阶段shuffle write的实际数据大小，来调整是否可以用 BroadcastJoin来代替SortMergeJoin，提高运行效率。

  参数：

属性名称	默认值	备注
spark.sql.adaptive.enabled	false	自适应执行框架的开关。
spark.sql.adaptive.join.enabled	true	开关。
spark.sql.adaptiveBroadcastJoinThreshold	等于 spark.sql.autoBroadcastJoinThreshold	运行过程中用于判断是否满足 BroadcastJoin 条件。

测试

以TPC-DS中某些query为例

- shuffle partition个数
  - Query 30

    原生Spark。

    

  - 自适应调整reduce个数。

    

- Runtime 执行计划优化（SortMergeJoin转BroadcastJoin）

  

  自适应转换为 BroadcastJoin

  

2.6. E-MapReduce数据迁移方案

在开发过程中我们通常会碰到需要迁移数据的场景，本文介绍如何将自建集群数据迁移到E-MapReduce集群中。

背景信息

适用范围：

- 线下Hadoop到E-MapReduce迁移。
- 线上ECS自建Hadoop到E-MapReduce迁移。

迁移场景：HDFS增量上游数据源包括RDS增量数据、flume。

新旧集群网络打通

- 线下IDC自建Hadoop

  现在自建Hadoop迁移到E-MapReduce可以通过OSS进行过度，或者使用阿里云高速通道产品建立线下IDC和线上E-MapReduce所在VPC网络的连通。

- 利用ECS自建Hadoop

  由于VPC实现用户专有网络之间的逻辑隔离，E-MapReduce建议使用VPC网络。

- 经典网络与VPC网络打通

  如果ECS自建Hadoop，需要通过ECS的classiclink的方式将经典网络和VPC网络打通，详情请参见建立ClassicLink连接。

- VPC网络之间连通

  数据迁移一般需要较高的网络带宽连通，建议新旧集群尽量处在同一个区域的同一个可用区内。

HDFS数据迁移

- Distcp工具同步数据

  HDFS数据迁移可以通过Hadoop社区标准的DistCp工具迁移，可以实现全量和增量的数据迁移。为减轻现有集群资源压力，建议在新旧集群网络连通后在新集群执行**distcp**命令。

  - 全量数据同步

    ```
    hadoop distcp -pbugpcax -m 1000 -bandwidth 30 hdfs://oldclusterip:8020/user/hive/warehouse /user/hive/warehouse
    ```

  - 增量数据同步

    ```
    hadoop distcp -pbugpcax -m 1000 -bandwidth 30  -update -delete hdfs://oldclusterip:8020/user/hive/warehouse /user/hive/warehouse
    ```

  参数说明：

  - `oldclusterip`：填写旧集群namenode ip，多个namenode情况填写当前状态为active的。
  - `-p`：默认副本数为3，如想保留原有副本数，`-p`后加r如`-prbugpcax`。如果不同步权限和ACL，`-p`后去掉p和a。
  - `-m`：指定map数，和集群规模、数据量有关。例如集群有2000核CPU，就可以指定2000个map。
  - `-bandwidth`：指定单个map的同步速度，是靠控制副本复制速度实现的，是大概值。
  - `-update`：校验源文件和目标文件的checksum和文件大小，如果不一致源文件会更新掉目标集群数据，新旧集群同步期间还有数据写入，可以通过`-update`做增量数据同步。
  - `-delete`：如果源集群数据不存在，新集群的数据也会被删掉。

  > **说明**
  > - 迁移整体速度受集群间带宽、集群规模影响。同时文件越多，checksum需要的时间越长。如果迁移数据量大，可以先试着同步几个目录评估一下整体时间。如果只能在指定时间段内同步，可以将目录切为几个小目录，依次同步。
  > - 一般全量数据同步，需要有个短暂的业务停写，以启用双写双算或直接将业务切换到新集群上。

- HDFS权限配置

  HDFS有权限设置，确定旧集群是否有ACL规则，是否要同步，检查新旧集群dfs.permissions.enabled和dfs.namenode.acls.enabled的配置是否一致，按照实际需要修改。

如果有ACL规则要同步，distcp参数后要加 `-p` 同步权限参数。如果distcp操作提示xx集群不支持 ACL，说明对应集群没配置ACL规则。新集群没配置ACL规则可以修改配置并重启namenode。旧集群不支持，说明旧集群根本就没有ACL方面的设置，也不需要同步。

Hive元数据同步

- 概述

  Hive元数据，一般存在MySQL里，与一般MySQL同步数据相比，要注意两点：

  - Location变化
  - Hive版本对齐

  E-MapReduce支持Hive Meta DB：

  - 统一元数据库，E-MapReduce管控RDS，每个用户一个Schema
  - 用户自建RDS
  - 用户ECS自建MySQL

  为了保证迁移之后新旧数据完全一致，最好是在迁移的时候将老的metastore服务停掉，等迁移过去之后，再把旧集群上的metastore服务打开，然后新集群开始提交业务作业。

- 操作步骤：

  i. 将新集群的元数据库删除，直接输出命令 `drop database xxx` 。

  ii. 将旧集群的元数据库的表结构和数据通过 `mysqldump` 命令全部导出。

  iii. 替换location、Hive元数据中分区等信息均带有location信息的，带dfs nameservices前缀的表，如*hdfs://mycluster:8020/*，而E-MapReduce集群的nameservices前缀是统一的E-MapReduce-cluster，所以需要订正。

  订正的最佳方式是先导出数据 。

  ```
  mysqldump --databases hivemeta --single-transaction -u root -p > hive_databases.sql
  ```

  用sed替换*hdfs://oldcluster:8020/*为*hdfs://E-MapReduce-cluster/*，再导入新db中。

  ```
  mysql hivemeta -p < hive_databases.sql
  ```

  iv. 在新集群的界面上，停止掉hivemetastore服务。

  v. 登录新的元数据库， `create database` 创建数据库。

  vi. 在新的元数据库中，导入替换location字段之后的老元数据库导出来的所有数据。

  vii. 版本对齐，E-MapReduce的Hive版本一般是当前社区最新的稳定版，自建集群Hive版本可能会更老，所以导入的旧版本数据可能不能直接使用。需要执行 Hive的升级脚本（期间会有表、字段已存在的问题可以忽略），请参见Hive升级脚本。例如Hive从1.2 升级到2.3.0，需要依次执行**upgrade-1.2.0-to-2.0.0.mysql.sql**、**upgrade-2.0.0-to-2.1.0.mysql.sql**、**upgrade-2.1.0-to-2.2.0.mysql.sql**、**upgrade-2.2.0-to-2.3.0.mysql.sql**。脚本主要是建表，加字段，改内容，如有表已存在，字段已存在的异常可以忽略。

  viii. Meta数据全部订正后，就可以重启metaserver了。命令行输入 `hive` ，查询库和表、查询数据、验证数据的正确性。

Flume数据迁移

- Flume双写配置

  在新集群上也开启flume服务，并且将数据按照和老集群完全一致的规则写入到新集群中。

- Flume分区表写入

  Flume数据双写，双写时需控制开始的时机，要保证flume在开始一个新的时间分区的时候来进行新集群的同步。 如flume每小时整点会同步所有的表，那就要整点之前，开启flume同步服务，这样flume在一个新的小时内写入的数据，在旧集群和新集群上是完全一致的。而不完整的旧数据在distcp的时候，全量的同步会覆盖它。而开启双写时间点后的新数据，在数据同步的时候不进行同步。 这个新的写入的数据，我们在划分数据阶段，记得不要放到数据同步的目录里。

有关Flume的使用和配置，请参见使用说明和配置说明。

作业同步

Hadoop、Hive、Spark、MR等如果有较大的版本升级，可能涉及作业改造，要视具体情况而定。

常见问题：

- Gateway OOM

  修改*/etc/ecm/hive-conf/hive-env.sh*文件中的 `export HADOOP_HEAPSIZE` 为1024。

- 作业执行内存不足

  `set mapreduce.map.java.opts=-Xmx3072m`

  `mapreduce.map.java.opts` 调整的是启动JVM虚拟机时，传递给虚拟机的启动参数，而默认值 `-Xmx3072m` 表示这个Java程序可以使用的最大堆内存数，一旦超过这个大小，JVM就会抛出Out of Memory异常，并终止进程。

  `set mapreduce.map.memory.mb=3840`

  `mapreduce.map.memory.mb` 设置的是Container的内存上限，这个参数由NodeManager读取并进行控制，当Container的内存大小超过了这个参数值，NodeManager会负责终止Container。

数据校验

由客户自行抽检报表完成。

Presto集群迁移

如果有单独的Presto集群仅仅用来做数据查询，需要修改Hive中配置文件，详情请参见Presto文档。

需要修改*hive.properties*中如下参数：

- `connector.name=hive-hadoop2`
- `hive.metastore.uri=thrift://E-MapReduce-header-1.cluster-500148414:9083`
- `hive.config.resources=/etc/ecm/hadoop-conf/core-site.xml, /etc/ecm/hadoop-conf/hdfs-site.xml`
- `hive.allow-drop-table=true`
- `hive.allow-rename-table=true`
- `hive.recursive-directories=true`

附录

Hive 1.2升级到2.3的版本对齐示例。

```
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-1.2.0-to-2.0.0.mysql.sql
CREATE TABLE COMPACTION_QUEUE (
  CQ_ID bigint PRIMARY KEY,
  CQ_DATABASE varchar(128) NOT NULL,
  CQ_TABLE varchar(128) NOT NULL,
  CQ_PARTITION varchar(767),
  CQ_STATE char(1) NOT NULL,
  CQ_TYPE char(1) NOT NULL,
  CQ_WORKER_ID varchar(128),
  CQ_START bigint,
  CQ_RUN_AS varchar(128),
  CQ_HIGHEST_TXN_ID bigint,
  CQ_META_INFO varbinary(2048),
  CQ_HADOOP_JOB_ID varchar(32)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE TXNS (
  TXN_ID bigint PRIMARY KEY,
  TXN_STATE char(1) NOT NULL,
  TXN_STARTED bigint NOT NULL,
  TXN_LAST_HEARTBEAT bigint NOT NULL,
  TXN_USER varchar(128) NOT NULL,
  TXN_HOST varchar(128) NOT NULL,
  TXN_AGENT_INFO varchar(128),
  TXN_META_INFO varchar(128),
  TXN_HEARTBEAT_COUNT int
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE HIVE_LOCKS (
  HL_LOCK_EXT_ID bigint NOT NULL,
  HL_LOCK_INT_ID bigint NOT NULL,
  HL_TXNID bigint,
  HL_DB varchar(128) NOT NULL,
  HL_TABLE varchar(128),
  HL_PARTITION varchar(767),
  HL_LOCK_STATE char(1) not null,
  HL_LOCK_TYPE char(1) not null,
  HL_LAST_HEARTBEAT bigint NOT NULL,
  HL_ACQUIRED_AT bigint,
  HL_USER varchar(128) NOT NULL,
  HL_HOST varchar(128) NOT NULL,
```

```
  HL_HEARTBEAT_COUNT int,
  HL_AGENT_INFO varchar(128),
  HL_BLOCKEDBY_EXT_ID bigint,
  HL_BLOCKEDBY_INT_ID bigint,
  PRIMARY KEY(HL_LOCK_EXT_ID, HL_LOCK_INT_ID),
  KEY HIVE_LOCK_TXNID_INDEX (HL_TXNID)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE INDEX HL_TXNID_IDX ON HIVE_LOCKS (HL_TXNID);
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-1.2.0-to-2.0.0.mysql.sql
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.0.0-to-2.1.0.mysql.sql

CREATE TABLE TXN_COMPONENTS (
  TC_TXNID bigint,
  TC_DATABASE varchar(128) NOT NULL,
  TC_TABLE varchar(128),
  TC_PARTITION varchar(767),
  FOREIGN KEY (TC_TXNID) REFERENCES TXNS (TXN_ID)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.0.0-to-2.1.0.mysql.sql
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.1.0-to-2.2.0.mysql.sql
CREATE TABLE IF NOT EXISTS `NOTIFICATION_LOG`
(
    `NL_ID` BIGINT(20) NOT NULL,
    `EVENT_ID` BIGINT(20) NOT NULL,
    `EVENT_TIME` INT(11) NOT NULL,
    `EVENT_TYPE` varchar(32) NOT NULL,
    `DB_NAME` varchar(128),
    `TBL_NAME` varchar(128),
    `MESSAGE` mediumtext,
    PRIMARY KEY (`NL_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE IF NOT EXISTS `PARTITION_EVENTS` (
  `PART_NAME_ID` bigint(20) NOT NULL,
  `DB_NAME` varchar(128) CHARACTER SET latin1 COLLATE latin1_bin DEFAULT NULL,
  `EVENT_TIME` bigint(20) NOT NULL,
  `EVENT_TYPE` int(11) NOT NULL,
  `PARTITION_NAME` varchar(767) CHARACTER SET latin1 COLLATE latin1_bin DEFAULT NULL,
  `TBL_NAME` varchar(128) CHARACTER SET latin1 COLLATE latin1_bin DEFAULT NULL,
  PRIMARY KEY (`PART_NAME_ID`),
  KEY `PARTITIONEVENTINDEX` (`PARTITION_NAME`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
CREATE TABLE COMPLETED_TXN_COMPONENTS (
  CTC_TXNID bigint NOT NULL,
  CTC_DATABASE varchar(128) NOT NULL,
  CTC_TABLE varchar(128),
  CTC_PARTITION varchar(767)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
 source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.1.0-to-2.2.0.mysql.sql
 source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.2.0-to-2.3.0.mysql.sql
  CREATE TABLE NEXT_TXN_ID (
  NTXN_NEXT bigint NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
INSERT INTO NEXT_TXN_ID VALUES(1);
CREATE TABLE NEXT_LOCK_ID (
  NL_NEXT bigint NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
INSERT INTO NEXT_LOCK_ID VALUES(1);
```

2.7. 通过Flink作业处理OSS数据

本文介绍如何在Hadoop集群中运行Flink作业来消费OSS数据。

前提条件

- 已注册阿里云账号，详情请参见阿里云账号注册流程。
- 已开通E-MapReduce服务和OSS服务。
- 已完成云账号的授权，详情请参见角色授权。

步骤一：准备环境

在创建Flink作业前，您需要在本地安装Maven和Java环境，以及在E-MapReduce上创建Hadoop集群。如果Maven是3.0以上版本，则建议Java选择2.0及以下版本，否则会造成不兼容情况。

1. 在本地安装Maven和Java环境。
2. 已创建E-MapReduce的Hadoop集群，并且选择了Flink服务，详情请参见创建集群。

步骤二：准备测试数据

在创建Flink作业前，您需要在OSS上传测试数据。本示例上传一个*test.txt*文件，文件内容为 `Nothing is impossible for a willing heart. While there is a life, there is a hope~` 。

1. 登录 OSS管理控制台。
2. 创建存储空间并上传测试数据文件，详情请参见创建存储空间和上传文件。测试数据的上传路径在后续步骤的代码中会使用，本例的上传路径为*oss://emr-logs2/test/test.txt*。

> **说明** 上传文件后，请保留OSS的登录窗口，后续仍会使用。

步骤三：制作JAR包并上传到OSS或Hadoop集群

本示例JAR包来源：下载E-MapReduce示例代码aliyun-emapreduce-demo，编译生成JAR包。JAR包可以上传至Hadoop集群的header主机中，也可以上传至OSS中。本示例上传到OSS。

1. 下载并解压缩aliyun-emapreduce-demo示例到本地。
2. 在Intellij IDE中，单击**file > open**，打开解压缩后的*aliyun-emapreduce-demo-master-2*。
3. 在下载文件中的*pom.xml*所在目录，执行如下命令制作JAR包。

   ```
   mvn clean package -DskipTests
   ```

4. 返回 OSS管理控制台。
5. 上传JAR包至OSS任一路径下。JAR包的上传路径在后续步骤的代码中会使用，本示例的上传路径为*oss://emr-logs2/test/examples-1.2.0.jar*。

步骤四：创建并运行Flink作业

1. 登录阿里云E-MapReduce控制台。
2. 在顶部菜单栏处，根据实际情况选择地域（Region）和资源组。
3. 单击上方的**数据开发**页签。
4. 在**数据开发**页面，创建项目，详情请参见项目管理。
5. 进入新建的项目，新建**Flink**类型的作业，详情请参见Flink（VVR）作业配置。
6. 新建Flink作业后，配置**作业内容**。**作业内容**示例如下。

   ```
   run -m yarn-cluster  -yjm 1024 -ytm 1024 -yn 4 -ys 4 -ynm flink-oss-sample -c com.aliyun.emr.example.flink.FlinkOSSSample  ossref://emr-logs2/test/examples-1.2.0.jar --input oss://emr-logs2/test/test.txt
   ```

   示例代码中的关键参数说明如下：

   - *ossref://emr-logs2/test/examples-1.2.0.jar*：上传至OSS的JAR包。
   - *oss://emr-logs2/test/test.txt*：上传到OSS的测试数据。

7. 作业配置完成后，单击右上方的**运行**。在**运行作业**对话框中，选择**执行集群**为新建的Hadoop集群。
8. 单击**确定**。作业成功运行后，即成功实现了在E-MapReduce集群上运行Flink作业处理OSS数据。

步骤五：查看作业提交日志和作业信息（可选）

如果需要定位作业失败的原因或了解作业的详细信息，则您可以查看作业的日志和作业信息。

1. 查看作业提交日志。当前提交日志支持在E-MapReduce控制台查看，也支持在SSH客户端查看。
   - 提交作业后，您可以在E-MapReduce控制台的**运行记录**页签，单击待查看作业所在行的**详情**。
   - 通过SSH客户端登录到Hadoop集群的header节点，查看提交的日志信息。

     默认情况下，根据Flink的**log4j**配置（详情请参见*/etc/ecm/flink-conf/log4j-yarn-session.properties*），提交日志会保存在*/mnt/disk1/log/flink/flink-{user}-client-{hostname}.log*。

其中，user为提交Flink作业的用户，hostname为提交作业所在的节点。以root用户在**emr-header-1**节点提交Flink作业为例，日志的路径为*/mnt/disk1/log/flink/flink-flink-historyserver-0-emr-header-1.cluster-126601.log*。

2. 查看作业信息。通过**Yarn UI**可以查看Flink作业的信息。访问**Yarn UI**有SSH隧道和Knox两种方式，SSH隧道方式请参见通过SSH隧道方式访问开源组件Web UI，Knox方式请参见Knox和访问链接与端口。下面以Knox方式为例进行介绍。
   i. 在Hadoop集群的**访问链接与端口**页面中，单击**Yarn UI**后的链接，
   ii. 在Hadoop控制台，单击作业的**ID**。查看作业运行详情。

   详细信息如下。
   iii. 如果您需要查看运行中的Flink作业，则可以在作业详情页面，单击**Tracking URL**后面的链接，进入**Flink Dashboard**查看。
   iv. 作业运行结束后，通过访问*http://emr-header-1:8082*，您可以查看所有已经完成的作业列表。

2.8. 使用E-MapReduce Hive关联云HBase

本文介绍如何使用E-MapReduce Hive关联云HBase的表。云HBase需要借助外部Hive对多表进行关联分析。

> **说明** 后续云HBase将集成Spark，建议使用Spark分析HBase数据。

准备工作

- 购买按量计费的EMR集群，配置依据实际场景确定，注意云HBase要和EMR处在同一 VPC下，建议不开启高可用。
- 将EMR所有节点的IP加入到云HBase白名单。

  获取云HBase的Zookeeper访问地址，可在云HBase控制台查看。
- 由于云HBase的HDFS端口默认是不开的，需要提交工单开通HDFS端口。

实施步骤

1. 使用SSH方式登录到集群，具体步骤请参见使用SSH连接主节点。
2. 修改Hive配置。
   - 进入Hive配置目录*/etc/ecm/hive-conf/*。
   - 修改*hbase-site.xml*，将hbase.zookeeper.quorum的值修改为云HBase的Zookeeper访问连接：

```
<property>
        <name>hbase.zookeeper.quorum</name>
        <value>hb-bp183x4tu8x7q****-001.hbase.rds.aliyuncs.com,hb-bp1mhyea7754b****-002.hbase.rds.aliyuncs.com,hb-bp1mhyea7754b****-003.hbase.rds.aliyuncs.com</value>
    </property>
```

3. Hive中创建云HBase表。
   - 如果HBase表不存在，可在Hive中直接创建云HBase关联表。
     a. 输入**hive**命令进入 `Hive cli` 命令行。
     b. 创建HBase表。

```
CREATE TABLE hive_hbase_table(key int, value string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")
TBLPROPERTIES ("hbase.table.name" = "hive_hbase_table", "hbase.mapred.output.outputtable" =
"hive_hbase_table");
```

     c. Hive中向HBase插入数据。

```
insert into hive_hbase_table values(212,'bab');
```

     d. 查看云HBase表，HBase表已创建，数据也已经写入。

     e. 在HBase中写入数据。

     在Hive中查看。

     f. Hive删除表，HBase表也一并删除。

     查看HBase表，报错不存在表。

   - 如果HBase表已存在，可在Hive中HBase外表进行关联，外部表在删除时不影响HBase已创建表。
     a. 云HBase中创建HBase表，并put测试数据。

     b. Hive中创建HBase外部关联表，并查看数据。

     c. 删除Hive表不影响HBase已存在的表。

总结

Hive更多操作HBase步骤，请参见HBaseIntegration。如果使用ECS自建MR集群的Hive时，操作步骤跟EMR操作类似，需要注意的是自建Hive的*hbase-site.xml*部分配置项可能与云HBase不一致，简单来说网络和端口开放后，只保留hbase.zookeeper.quorum即可与云HBase进行关联。

2.9. 使用 E-MapReduce 进行 MySQL Binlog 日志准实时传输

本文介绍如何利用阿里云的 SLS 插件功能和 E-MapReduce 集群进行 MySQL binlog 的准实时传输。

基本架构

RDS -> SLS -> Spark Streaming -> Spark HDFS

上述链路主要包含3个过程：

1. 如何把 RDS 的 binlog 收集到 SLS。
2. 如何通过 Spark Streaming 将 SLS 中的日志读取出来，进行分析。
3. 如何把链路 2 中读取和处理过的日志，保存到 Spark HDFS中。

环境准备

1. 安装一个 MySQL 类型的数据库（使用 MySQL 协议，例如 RDS、DRDS 等），开启 log-bin 功能，且配置 binlog 类型为 ROW 模式（RDS默认开启）。
2. 开通 SLS 服务。

操作步骤

1. 检查 MySQL 数据库环境。
   i. 查看是否开启 log-bin 功能。

```
mysql> show variables like "log_bin";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |
+---------------+-------+
1 row in set (0.02 sec)
```

   ii. 查看 binlog 类型。

```
mysql> show variables like "binlog_format";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
1 row in set (0.03 sec)
```

2. 添加用户权限。（也可以直接通过RDS控制台添加）

```
CREATE USER canal IDENTIFIED BY 'canal';
GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'canal'@'%';
FLUSH PRIVILEGES;
```

3. 为 SLS 服务添加对应的配置文件，并检查数据是否正常采集。
   i. 在 SLS 控制台添加对应的 project 和 logstore，例如：创建一个名称为 canaltest 的 project，然后创建一个名称为 canal 的 logstore。
   ii. 对 SLS 进行配置：在 */etc/ilogtail* 目录下创建文件user_local_config.json，具体配置如下。

```
{
"metrics": {
  "##1.0##canaltest$plugin-local": {
    "aliuid": "****",
    "enable": true,
    "category": "canal",
    "defaultEndpoint": "*******",
    "project_name": "canaltest",
    "region": "cn-hangzhou",
    "version": 2
    "log_type": "plugin",
    "plugin": {
      "inputs": [
        {
          "type": "service_canal",
          "detail": {
            "Host": "*****",
            "Password": "****",
            "ServerID": ****,
            "User" : "***",
            "DataBases": [
              "yourdb"
            ],
            "IgnoreTables": [
              "\\S+_inner"
            ],
             "TextToString" : true
          }
        }
      ],
      "flushers": [
        {
          "type": "flusher_sls",
          "detail": {}
        }
      ]
    }
  }
}
}
```

其中 detail 中的 Host 和 Password 等信息为 MySQL 数据库信息，User 信息为之前授权过的用户名。aliUid、defaultEndpoint、project_name、category 请根据自己的实际情况填写对应的用户和 SLS 信息。

iii. 等待约 2 分钟，通过 SLS 控制台查看日志数据是否上传成功，具体如图所示。

如果日志数据没有采集成功，请根据SLS的提示，查看SLS的采集日志进行排查。

4. 准备代码，将代码编译成 jar 包，然后上传到 OSS。

i. 将 EMR 的示例代码通过 git 复制下来，然后进行修改，具体命令为：`git clone https://github.com/aliyun/aliyun-emapreduce-demo.git` 。示例代码中已经有 LoghubSample 类，该类主要用于从 SLS 采集数据并打印。以下是修改后的代码，仅供参考。

```
package com.aliyun.emr.example
import org.apache.spark.SparkConf
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.aliyun.logservice.LoghubUtils
import org.apache.spark.streaming.{Milliseconds, StreamingContext}
object LoghubSample {
def main(args: Array[String]): Unit = {
if (args.length < 7) {
 System.err.println(
  """Usage: bin/spark-submit --class LoghubSample examples-1.0-SNAPSHOT-shaded.jar
    |
    |
  """.stripMargin)
 System.exit(1)
}
val loghubProject = args(0)
val logStore = args(1)
val loghubGroupName = args(2)
val endpoint = args(3)
val accessKeyId = args(4)
val accessKeySecret = args(5)
val batchInterval = Milliseconds(args(6).toInt * 1000)
val conf = new SparkConf().setAppName("Mysql Sync")
//    conf.setMaster("local[4]");
val ssc = new StreamingContext(conf, batchInterval)
val loghubStream = LoghubUtils.createStream(
 ssc,
 loghubProject,
 logStore,
 loghubGroupName,
```

```
 endpoint,
 1,
 accessKeyId,
 accessKeySecret,
 StorageLevel.MEMORY_AND_DISK)
loghubStream.foreachRDD(rdd =>
  rdd.saveAsTextFile("/mysqlbinlog")
)
ssc.start()
ssc.awaitTermination()
}
}
```

其中的主要改动是：`loghubStream.foreachRDD(rdd => rdd.saveAsObjectFile("/mysqlbinlog") )`。这样在 EMR 集群中运行时，就会把Spark Streaming 中流出来的数据，保存到 EMR 的 HDFS 中。

> **说明**
>
> - 由于如果要在本地运行，请在本地环境提前搭建 Hadoop 集群。
> - 由于 EMR 的 Spark SDK 做了升级，其示例代码比较旧，不能直接在参数中传递 OSS 的 AccessKeyId、AccessKeySecret， 而是需要通过 SparkConf 进行设置，如下所示。
>
>   ```
>   trait RunLocally {
>   val conf = new SparkConf().setAppName(getAppName).setMaster("local[4]")
>   conf.set("spark.hadoop.fs.oss.impl", "com.aliyun.fs.oss.nat.NativeOssFileSystem")
>   conf.set("spark.hadoop.mapreduce.job.run-local", "true")
>   conf.set("spark.hadoop.fs.oss.endpoint", "YourEndpoint")
>   conf.set("spark.hadoop.fs.oss.accessKeyId", "YourId")
>   conf.set("spark.hadoop.fs.oss.accessKeySecret", "YourSecret")
>   conf.set("spark.hadoop.job.runlocal", "true")
>   conf.set("spark.hadoop.fs.oss.impl", "com.aliyun.fs.oss.nat.NativeOssFileSystem")
>   conf.set("spark.hadoop.fs.oss.buffer.dirs", "/mnt/disk1")
>   val sc = new SparkContext(conf)
>   def getAppName: String
>   }
>   ```
>
> - 在本地调试时，需要把 `loghubStream.foreachRDD(rdd => rdd.saveAsObjectFile("/mysqlbinlog") )` 中的 `/mysqlbinlog` 修改成本地 HDFS的地址。

ii. 代码编译。

在本地调试完成后，我们可以通过如下命令进行打包编译。

```
mvn clean install
```

iii. 上传 jar 包。

请先在 OSS 上建立 bucket 为*qiaozhou-EMR/jar*的目录，然后通过OSS 控制台或 OSS 的 SDK 将*/target/shaded*目录下的*examples-1.1-shaded.jar*上传到 OSS 的这个目录下。上传后的 jar 包地址为*oss://qiaozhou-EMR/jar/examples-1.1-shaded.jar*，这个地址在后面会用上，如下图所示。

5. 搭建 EMR 集群，创建任务并运行执行计划。

i. 通过 EMR 控制台创建一个 EMR 集群，大约需要 10 分钟左右，请耐心等待。

ii. 创建一个类型为 Spark 的作业。

请根据您具体的配置将 `SLS_endpoint` `$SLS_access_id` `$SLS_secret_key` 替换成真实值。请注意参数的顺序，否则可能会报错。

```
--master yarn --deploy-mode client --driver-memory 4g --executor-memory 2g --executor-cores 2 --class com.aliyun.EMR.example.LoghubSample ossref://EMR-test/jar/examples-1.1-shaded.jar canal test canal sparkstreaming $SLS_endpoint $SLS_access_id $SLS_secret_key 1
```

iii. 创建执行计划，将作业和 EMR 集群绑定后，开始运行。

iv. 查询 Master 节点的IP，如图所示。

通过 SSH 登录后，执行以下命令。

```
hadoop fs -ls /
```

可以看到 mysqlbinlog 开头的目录，再通过以下命令查看 mysqlbinlog 文件。

```
hadoop fs -ls /mysqlbinlog
```

还可以通过 `hadoop fs -cat /mysqlbinlog/part-00000` 命令查看文件内容。

6. 错误排查。

如果没有看到正常的结果，可以通过 EMR 的运行记录，来进行问题排查，如图所示。

2.10. Gateway节点运行Flume进行数据同步

本文介绍阿里云EMR-3.17.0及后续版本，如何使用Gateway节点运行Flume从而进行数据同步。

背景信息

EMR-3.16.0及后续版本支持Apache Flume。EMR-3.17.0及后续版本提供默认监控等特性。

在Gateway节点运行Flume可以避免对E-MapReduce Hadoop集群产生影响。使用Gateway节点部署Flume Agent的基本数据流如下图所示。

环境准备

本示例在华北1（杭州）进行测试，版本选择EMR- 3.17.0。

- 创建Hadoop集群，在**可选服务**中选择**Flume**。

  

- 创建Gateway节点，关联已创建的Hadoop集群。

实施步骤

- 运行Flume，请参见使用说明。
- 查看监控信息。

  默认情况下，集群服务页面提供了Flume Agent的监控信息。通过在集群与服务管理页面单击 **Flume** 服务进行访问，如下图所示。

  

  > **注意**
  >
  > 监控数据以Agent组件（Source、Channel或Sink）的名称命名。例如，CHANNEL.channel1表示名称为channel1的Channel组件的监控指标，所以在配置不同的Agent时请避免使用相同的组件名称。
  >
  > 如果您想通过Ganglia等方式查看Flume Agent的监控数据，可以参考Flume官网进行配置。

- 查看日志 。

  默认情况下，Flume agent日志的存放路径为*/mnt/disk1/log/flume/${flume-agent-name}/flume.log*。您可以通过修改 */etc/ecm/flume-conf/log4j.properties*进行配置（不建议修改日志路径）。

  > **注意** 日志路径包含了Flume Agent的名称，所以配置不同的Agent时请勿使用相同的Agent名称，以免日志混在一起。

2.11. 在EMR上使用Sqoop与数据库同步数据时的网络配置

如果您的E-MapReduce（EMR）集群需要和集群之外的数据库同步数据，确保网络是联通的。本文以RDS、ECS自建和云下私有数据库三种情况为例，分别介绍如何配置网络。

云数据库RDS

Sqoop是用map任务同步数据，可以在任意节点上运行，而Sqoop任务需要配置连接RDS的内网地址来连接，所以，需要确保EMR集群的内网IP在RDS白名单里。

EMR集群和RDS需要在同一个VPC网络内，以便于可以直接访问RDS地址。如果在不同的VPC网络下，需要通过高速通道打通网络连接。

ECS自建数据库

访问VPC网络的自建数据库跟VPC网络的RDS类似，EMR集群需要使用VPC网络，并且数据库ECS实例和EMR集群实例需要在同一个安全组内。

云下私有数据库

有两种方式访问云下私有数据库，一种是绑定弹性IP（EIP）访问数据库的公网地址，一种是将云下数据库通过高速通道和VPC网络互联。

- 绑定EIP

  如果云下私有数据库可以通过公网访问，则创建一个VPC网络类型的EMR集群。创建后如果您需要使用公网IP地址访问，请在ECS上申请开通公网IP地址，详情请参见弹性公网IP。

- 高速通道

  如果私有数据库不能在公网暴露，可以创建一个VPC网络类型的 EMR 集群，通过高速通道连接私有 IDC 和阿里云上的 VPC 集群。

  高速通道详情请参见高速通道。

2.12. 通过Spark Streaming作业处理Kafka数据

本文介绍如何使用阿里云E-MapReduce创建的Hadoop和Kafka集群，运行Spark Streaming作业以消费Kafka数据。

前提条件

- 已注册阿里云账号，详情请参见阿里云账号注册流程。
- 已开通E-MapReduce服务。
- 已完成云账号的授权，详情请参见角色授权。
- 本地安装了PuTTY和文件传输工具（SSH Secure File Transfer Client）。

步骤一：创建Hadoop集群和Kafka集群

创建同一个安全组下的Hadoop和Kafka集群。创建详情请参见创建集群。

1. 登录阿里云E-MapReduce控制台。
2. 创建Hadoop集群。
3. 创建Kafka集群。

步骤二：获取JAR包并上传到Hadoop集群

1. 获取JAR包（examples-1.2.0-shaded-2.jar.zip）。
2. 使用文件传输工具，上传JAR包至Hadoop集群Master节点的*/home/hadoop*路径下。

步骤三：在Kafka集群上创建Topic

本示例将创建一个分区数为10、副本数为2、名称为test的Topic。

1. 登录Kafka集群的Master节点，详情请参见使用SSH连接主节点。
2. 通过如下命令创建Topic。

```
/usr/lib/kafka-current/bin/kafka-topics.sh --partitions 10 --replication-factor 2 --zookeeper emr-header-1:2181 /kafka-1.0.0 --topic test --create
```

> **说明** 创建Topic后，请保留该登录窗口，后续步骤仍将使用。

步骤四：运行Spark Streaming作业

本示例将运行一个流式单词统计（WordCount）的作业。

1. 登录Hadoop集群的Master节点，详情请参见使用SSH连接主节点。
2. 执行如下作业命令，进行流式单词统计（WordCount）。

```
spark-submit --class com.aliyun.emr.example.spark.streaming.KafkaSample /home/hadoop/examples-1.2.0-shaded-2.jar 192.168.xxx.xxx:9092 test 5
```

关键参数说明如下：

参数	描述
192.168.xxx.xxx	Kafka集群中任一**Kafka Broker**组件的内网IP地址。IP地址如Kafka集群组件所示。
test	Topic名称。
5	时间间隔。

Kafka集群组件

步骤五：使用Kafka发布消息

1. 在Kafka集群的命令行窗口，执行如下命令运行Kafka的生产者。

```
/usr/lib/kafka-current/bin/kafka-console-producer.sh --topic test --broker-list emr-worker-1:9092
```

2. 在Kafka集群的登录窗口中输入文本，在Hadoop集群的登录窗口中，会实时显示文本的统计信息。

   例如，在Kafka集群的登录窗口输入如下信息。

Hadoop集群的登录窗口会输出如下信息。

步骤六：查看Spark Streaming作业状态

1. 在E-MapReduce控制台的**集群管理**页面。
2. 单击创建的Hadoop集群所在行的**详情**。
3. 在左侧导航栏，单击**访问链接与端口**。
4. 单击**Spark History Server UI**所在行的链接。

5. 在History Server页面，单击待查看的App ID。您可以查看Spark Streaming作业的状态。

2.13. 通过Kafka Connect进行数据迁移

在流式数据处理过程中，E-MapReduce经常需要在Kafka与其他系统间进行数据同步或者在Kafka集群间进行数据迁移。本文向您介绍如何在E-MapReduce上通过Kafka Connect快速的实现Kafka集群间的数据迁移。

前提条件

- 已注册云账号，详情请参见阿里云账号注册流程。
- 已开通E-MapReduce服务。
- 已完成云账号的授权，详情请参见角色授权。

背景信息

Kafka Connect是一种可扩展的、可靠的、用于Kafka与其他系统间进行数据同步或者在Kafka集群间进行流式数据传输的工具。例如，Kafka Connect可以获取数据库的binlog数据，将数据库数据同步至Kafka集群，从而达到迁移数据库数据的目的。由于Kafka集群可对接流式处理系统，所以还可以间接实现数据库对接下游流式处理系统的目的。同时，Kafka Connect还提供了REST API接口，方便您创建和管理Kafka Connect。

Kafka Connect分为Standalone和Distributed两种运行模式。在Standalone模式下，所有的worker都在一个进程中运行。相比于Standalone模式，Distributed模式更具扩展性和容错性，是最常用的方式，也是生产环境推荐使用的模式。

本文介绍如何在E-MapReduce上使用Kafka Connect的REST API接口在Kafka集群间进行数据迁移，Kafka Connect使用distributed模式。

步骤一：创建Kafka集群

在EMR上创建源Kafka集群和目的Kafka集群。Kafka Connect安装在Task节点上，所以目的Kafka集群必须创建Task节点。集群创建好后，Task节点上Kafka Connect服务会默认启动，端口号为8083。

1. 登录阿里云E-MapReduce控制台。
2. 创建源Kafka集群和目的Kafka集群，详情请参见创建集群。

> 注意
> - 源Kafka集群和目的Kafka集群创建在同一个安全组下。
>
>   如果源Kafka集群和目的kafka集群不在同一个安全组下，则两者的网络默认是不互通的，您需要对两者的安全组分别进行相关配置，以使两者的网络互通。
> - 创建目的Kafka集群后，需要扩容Task节点，详情请参见新增机器组。

步骤二：准备待迁移数据Topic

在源Kafka集群上创建一个名称为**connect**的Topic。

1. 以SSH方式登录到源Kafka集群的Master节点（本例为**emr-header-1**）。

2. 以root用户运行如下命令，创建一个名称为**connect**的Topic。

```
kafka-topics.sh --create --zookeeper emr-header-1:2181 --replication-factor 2 --partitions 10 --topic co
nnect
```

> **说明** 完成上述操作后，请保留该登录窗口，后续仍将使用。

步骤三：创建Kafka Connect的connector

在目的Kafka集群的Task节点上，使用 curl 命令，通过JSON数据创建一个Kafka Connect的connector。

1. 自定义Kafka Connect配置。

   进入阿里云E-MapReduce控制台目的Kafka集群**Kafka**服务的**配置**页面，在**connect-distributed.properties**中自定义**offset.storage.topic**、**config.storage.topic**和**status.storage.topic**三个配置项，详情请参见组件参数配置。

   Kafka Connect会将offsets、configs和任务状态保存在Topic中，Topic名对应**offset.storage.topic**、**config.storage.topic**和**status.storage.topic**三个配置项。Kafka Connect会自动使用默认的partition和replication factor创建这三个Topic，其中partition和repication factor配置项保存在*/etc/ecm/kafka-conf/connect-distributed.properties*文件中。

2. 以SSH方式登录到目的Kafka集群的Master节点（本例为**emr-header-1**）。
3. 切换至Task节点（本例为**emr-worker-3**）。

4. 以root用户运行如下命令创建一个Kafka Connect。

```
curl -X POST -H "Content-Type: application/json" --data '{"name": "connect-test", "config": { "connector.
class": "EMRReplicatorSourceConnector", "key.converter": "org.apache.kafka.connect.converters.ByteArra
yConverter", "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter", "src.kafka.bo
otstrap.servers": "${src-kafka-ip}:9092", "src.zookeeper.connect": "${src-kafka-curator-ip}:2181", "dest.z
ookeeper.connect": "${dest-kafka-curator-ip}:2181", "topic.whitelist": "${source-topic}", "topic.rename.f
ormat": "${dest-topic}", "src.kafka.max.poll.records": "300" } }' http://emr-worker-3:8083/connectors
```

在JSON数据中，name字段代表创建的Kafka Connect的名称，本例为*connect-test*；config字段需要根据实际情况进行配置，关键变量的说明如下。

变量	说明
${source-topic}	源Kafka集群中需要迁移的Topic，多个Topic需用英文逗号（,）隔开，例如**connect**。
${dest-topic}	目的Kafka集群中迁移后的Topic，例如**connect.replica**。
${src-kafka-curator-hostname}	源Kafka集群中安装了ZooKeeper服务的节点的内网IP地址。

变量	说明
${dest-kafka-curator-hostname}	目的Kafka集群中安装了ZooKeeper服务的节点的内网IP地址。

> **说明** 完成上述操作后，请保留该登录窗口，后续仍将使用。

步骤四：查看Kafka Connect和Task节点状态

查看Kafka Connect和Task节点信息，确保两者的状态正常。

1. 返回到目的Kafka集群的Task节点（本例为**emr-worker-3**）的登录窗口。
2. 以root用户运行如下命令查看所有的Kafka Connect。

   ```
   curl emr-worker-3:8083/connectors
   ```

   
3. 以root用户运行如下命令查看本例创建的Kafka Connect（本例为**connect-test**）的状态。

   ```
   curl emr-worker-3:8083/connectors/connect-test/status
   ```

   

   确保Kafka Connect（本例为**connect-test**）的状态为**RUNNING**。
4. 以root用户运行如下命令查看Task节点信息。

   ```
   curl emr-worker-3:8083/connectors/connect-test/tasks
   ```

   

   确保Task节点的返回信息中无错误信息。

步骤五：生成待迁移数据

通过命令向源集群中的**connect** Topic发送待迁移的数据。

1. 返回到源Kafka集群的header节点（本例为**emr-header-1**）的登录窗口。
2. 以root用户运行如下命令向**connect**的Topic发送数据。

   ```
   kafka-producer-perf-test.sh --topic connect --num-records 100000 --throughput 5000 --record-size 100
   0 --producer-props bootstrap.servers=emr-header-1:9092
   ```

   当提示如下信息，则表示已经成功生成待迁移数据。

   

步骤六：查看数据迁移结果

生成待迁移数据后，Kafka Connect会自动将这些数据迁移到目的集群的相应文件（本例为**connect.replica**）中。

1. 返回到目的Kafka集群的Task节点（本例为**emr-worker-3**）的登录窗口。
2. 以root用户运行如下命令查看数据迁移是否成功。

```
kafka-consumer-perf-test.sh --topic connect.replica --broker-list emr-header-1:9092 --messages 10000
0
```

从上述返回结果可以看出，在源Kafka集群发送的100000条数据已经迁移到了目的Kafka集群。

小结

本文介绍并演示了使用Kafka Connect在Kafka集群间进行数据迁移的方法。如果需要了解Kafka Connect更详细的使用方法，请参见Kafka官网资料和REST API。

2.14. 通过JDBC连接HiveServer2来访问Hive数据

本文介绍如何通过JDBC连接HiveServer2访问Hive数据。适用于无法通过Hive Client和HDFS访问Hive数据的场景。

前提条件

- 已对Hive进行权限配置，详情请参见Hive配置。
- 因为HiveServer2默认不校验用户和密码，所以当您需要用户和密码认证时，请进行用户认证配置，详情请参见如何设置HiveServer2的认证方式为LDAP？。

背景信息

JDBC连接HiveServer2的方法如下：

- Beeline：通过HiveServer2的JDBC客户端进行连接。
- Java：编写Java代码进行连接。

> **说明** E-MapReduce集群中，Hue通过HiveServer2方式来访问Hive数据。

在E-MapReduce集群中，HiveServer2的JDBC连接地址如下：

- 标准集群：*jdbc:hive2://emr-header-1:10000*
- 高安全集群：*jdbc:hive2://${master1_fullhost}:10000/;principal=hive/${master1_fullhost}@EMR.$id.COM*

Beeline客户端连接HiveServer2

1. 登录集群主节点，详情请参见使用SSH连接主节点。
2. 执行如下命令，进入Beeline客户端。

   ```
   [root@emr-header-1 ~]# beeline
   ```

   返回如下信息。

   ```
   Beeline version 1.2.1.spark2 by Apache Hive
   ```

3. 执行如下命令，连接HiveServer2。

```
beeline> !connect jdbc:hive2://emr-header-1:10000
```

返回如下信息。

```
Connecting to jdbc:hive2://emr-header-1:10000
```

4. 输入用户名和密码。

```
Enter username for jdbc:hive2://emr-header-1:10000: your_username
Enter password for jdbc:hive2://emr-header-1:10000: your_password
```

返回如下信息。

```
log4j:WARN No such property [datePattern] in org.apache.log4j.RollingFileAppender.
18/12/17 18:09:16 INFO Utils: Supplied authorities: emr-header-1:10000
18/12/17 18:09:16 INFO Utils: Resolved authority: emr-header-1:10000
18/12/17 18:09:16 INFO HiveConnection: Will try to open client transport with JDBC Uri: jdbc:hive2://emr
-header-1:10000
Connected to: Apache Hive (version 2.3.3)
Driver: Hive JDBC (version 1.2.1.spark2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
```

5. 查询Hive数据。

```
0: jdbc:hive2://emr-header-1:10000> select * from emrusers limit 10;
+------------------+-------------------+------------------+--------------------+--------------+--+
| emrusers.userid  | emrusers.movieid  | emrusers.rating  | emrusers.unixtime  | emrusers.dt  |
+------------------+-------------------+------------------+--------------------+--------------+--+
| 196          | 242          | 3           | 881250949       | 2018100102   |
| 186          | 302          | 3           | 891717742       | 2018100102   |
| 22           | 377          | 1           | 878887116       | 2018100102   |
| 244          | 51           | 2           | 880606923       | 2018100102   |
| 166          | 346          | 1           | 886397596       | 2018100102   |
| 298          | 474          | 4           | 884182806       | 2018100102   |
| 115          | 265          | 2           | 881171488       | 2018100102   |
| 253          | 465          | 5           | 891628467       | 2018100102   |
| 305          | 451          | 3           | 886324817       | 2018100102   |
| 6            | 86           | 3           | 883603013       | 2018100102   |
+------------------+-------------------+------------------+--------------------+--------------+--+
10 rows selected (1.455 seconds)
```

Java连接HiveServer2

在执行本操作前，确保您已安装Java环境和Java编程工具，并且已配置环境变量。

1. 在*pom.xml*文件中配置项目依赖（hadoop-common和hive-jdbc）。本示例新增的项目依赖如下：

```
<dependencies>
    <dependency>
      <groupId>org.apache.hive</groupId>
      <artifactId>hive-jdbc</artifactId>
      <version>1.2.1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-common</artifactId>
      <version>2.7.2</version>
    </dependency>
</dependencies>
```

2. 编写代码，连接HiveServer2并操作Hive数据。示例如下。

```
import java.sql.*;

public class App
{
   private static String driverName = "org.apache.hive.jdbc.HiveDriver";

   public static void main(String[] args) throws SQLException {

      try {
         Class.forName(driverName);
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      }

      Connection con = DriverManager.getConnection(
            "jdbc:hive2://emr-header-1:10000", "root", "");  //代码打包后，运行JAR包的环境需要在hosts文件中把emr-header-1映射到E-MapReduce集群的公网IP地址（或内网IP地址）。

      Statement stmt = con.createStatement();

      String sql = "select * from emrusers limit 10";
      ResultSet res = stmt.executeQuery(sql);

      while (res.next()) {
         System.out.println(res.getString(1) + "\t" + res.getString(2));
      }

   }
}
```

3. 打包项目工程（即生成JAR包），并上传JAR包至运行环境。

   > 注意 JAR包的运行需要依赖hadoop-common和hive-jdbc。如果运行环境的环境变量中未包含这两个依赖包，则您需要下载依赖包并配置环境变量，或者直接一起打包这两个依赖包。运行JAR包时，如果缺少这两个依赖包，则会提示以下错误：
   >
   > - 缺失*hadoop-common*：提示java.lang.NoClassDefFoundError: org/apache/hadoop/conf/Configuration
   > - 缺失*hive-jdbc*：提示java.lang.ClassNotFoundException: org.apache.hive.jdbc.HiveDriver

   本示例生成的JAR包为*emr-hiveserver2-1.0.jar*，上传到E-MapReduce集群的**emr-header-1**。

4. 运行JAR包，测试是否可正常运行。

> **注意** 运行JAR包的服务器与E-MapReduce集群需要在同一个VPC和安全组下，并且网络可达。如果两者的VPC不同或网络环境不同，则需要通过公网地址访问，或先使用网络产品打通两者的网络，再通过内网访问。网络连通性测试方法：
>
> - 公网：telnet *emr-header-1的公网IP地址 10000*
> - 内网：telnet *emr-header-1的内网IP地址 10000*

```
[root@emr-header-1 xxx-test]# java -jar emr-hiveserver2-1.0.jar
```

返回如下信息。

```
log4j:WARN No appenders could be found for logger (org.apache.hive.jdbc.Utils).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
196    242
186    302
22     377
244    51
166    346
298    474
115    265
253    465
305    451
6      86
```