Alibaba Cloud E-MapReduce

Best Practice

Issue: 20181113

MORE THAN JUST CLOUD | Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, special, or punitive damages, including lost profits arising from the use or trust in this document, even if Alibaba Cloud has been notified of the possibility of such a loss.
5. By law, all the content of the Alibaba Cloud website, including but not limited to works, products, images, archives, information, materials, website architecture, website graphic layout, and webpage design, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade

secrets. No part of the Alibaba Cloud website, product programs, or content shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates).

6. Please contact Alibaba Cloud directly if you discover any errors in this document.

Generic conventions

Table -1: Style conventions

Style	Description	Example
	This warning information indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
	This warning information indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	**Note:** Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user.	**Note:** You can use **Ctrl** + **A** to select all files.
>	Multi-level menu cascade.	**Settings** > **Network** > **Set network type**
Bold	It is used for buttons, menus, page names, and other UI elements.	Click **OK**.
`Courier font`	It is used for commands.	Run the `cd /d C:/windows` command to enter the Windows system folder.
Italics	It is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	It indicates that it is a optional value, and only one item can be selected.	`ipconfig` *[-all|-t]*
{} or {a|b}	It indicates that it is a required value, and only one item can be selected.	`swich` *{stand | slave}*

Contents

1 Use EMR for real-time MySQL binlog transmission

This section describes how to use the SLS plug-in function of Alibaba Cloud and the EMR cluster to implement quasi-real-time transmission of MySQL binlog.

Basic architecture

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

The preceding links contain three processes:

1. How to collect RDS binlog to SLS.
2. How to read and analyze the logs in SLS through Spark Streaming.
3. How to save the logs read and processed in the second link to Spark HDFS.

Prepare the environment

1. Install a MySQL database (using MySQL protocol, such as RDS and DRDS), and enable the log-bin function. Configure the binlog type to ROW mode. (RDS is enabled by default.)
2. Enable the SLS service.

Procedure

1. Check the MySQL database environment.

    a. View whether the log-bin function is enabled.

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

    b. View the binlog type.

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

2. Add user permissions. You can also add user permissions directly from the RDS console.

    ```
    CREATE USER canal IDENTIFIED BY 'canal';
    ```

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

3. Add the corresponding configuration file for the SLS service, and check if the data is collected properly.

   a. Add the corresponding project and logstore in the SLS console. For example, create a project named canaltest and a logstore named canal.

   b. Configure SLS: create a file named user_local_config.json under the directory of /etc/ilogtail.

```
{
"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": {}
               }
           ]
       }
   }
}
}
```

   The information such as host and password in detail is MySQL database information, and the user information is the user name authorized previously. AliUid, defaultEndpoint, project_name, and category are information related with users and SLS. Fill in the informatio n according to your actual situation.

c. Wait about 2 minutes to see if the log data has been uploaded successfully in the SLS console.

   If the log data acquisition is not successful, view the acquisition log of SLS based on its prompt for troubleshooting.

4. Prepare and compile the code to jar package, and upload it to OSS.

   a. Copy the example code of EMR using Git and modify the code. The command is as follows: `git clone https://github.com/aliyun/aliyun-emapreduce-demo.git`. The example code includes the LoghubSample class, which is primarily used to capture and print data from SLS. The modified code is as below:

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

```
}
```

The main change is as follows: `loghubStream.foreachRDD(rdd => rdd.saveAsObjectFile("/mysqlbinlog") )`. When the example code is run in the EMR cluster, the data that flows out of Spark Streaming will be saved in HDFS of EMR.

Note:

- To run the example code locally, create a Hadoop cluster in the local environment in advance.
- Because the Spark SDK of EMR is updated, its example code is old and cannot directly transfer the AccessKey ID and AccessKey Secret of OSS in the parameter. You need to set the Spark SDK with the SparkConf constructor, as shown in the following figure:

```
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
}
```

- During local debugging, you need to change /mysqlbinlogloghubStream.foreachRDD(rdd => in rdd.saveAsObjectFile("/mysqlbinlog")) to the local HDFS address.

b. Compile code.

After local debugging is complete, you can run the following command to package and compile the code:

```
mvn clean install
```

c. Upload the jar package.

Create a directory on an OSS instance where the bucket is qiaozhou-EMR/jar, and upload examples-1.1-shaded.jar under the directory of /target/shaded to the OSS directory through the OSS console or the SDK of OSS. The uploaded jar package address is oss://qiaozhou-EMR/jar/examples-1.1-shaded.jar. This address will be used later.

5. Create an EMR cluster and tasks, and run the execution plans.

a. Create an EMR cluster in the EMR console, which takes about 10 minutes.

b. Create a job of the Spark type.

   Replace `SLS_endpoint` `$SLS_access_id` `$SLS_secret_key` with your actual values.

   Make sure that the order of the parameters is correct. Otherwise, errors may be reported.

```
--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
canaltest canal sparkstreaming $SLS_endpoint $SLS_access_id $
SLS_secret_key 1
```

c. After the execution plan is created, bind jobs to the EMR cluster. Start to run the jobs.

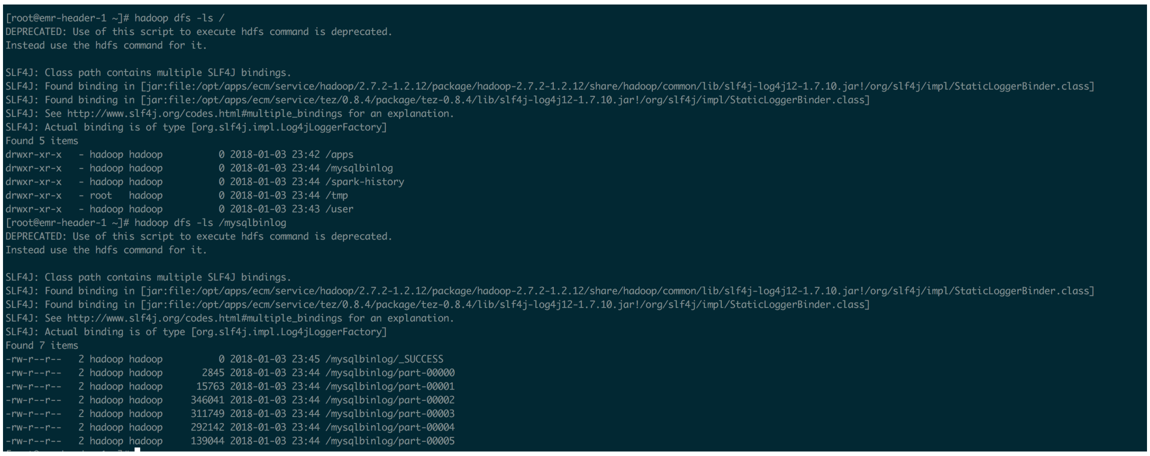
d. Search for the IP address of the master node.

   After you login through SSH, run the following command:

```
hadoop fs -ls /
```

   You can see the directory at the beginning of mysqlbinlog, and view the mysqlbinlog file with the following command:

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



   You can also run `hadoop fs -cat /mysqlbinlog/part-00000` command to view the file content.

6. Troubleshoot.

   If you don't see the normal results, you can troubleshoot problems in the running records of EMR.

2 Use E-MapReduce to process offline jobs

This section describes how to use E-MapReduce to read data from OSS, and a set of offline data processing operations, such as data collection and data clean-up.

Overview

E-MapReduce clusters can be used in various scenarios. E-MapReduce supports all the scenarios that the Hadoop ecosystem and Spark support. E-MapReduce is based on Hadoop and Spark clusters. You can use Alibaba Cloud ECS instances hosted by E-MapReduce clusters in the same way as you would on your physical machines.

Two popular kinds of big data processing that we use today are offline and online data processing.

- Offline data processing: You only want to obtain the analytical results of data without a major concern about the time it takes. For example, in a batch data processing scenario, you receive data from OSS and output processing results to OSS, using MapReduce, Hive, Pig, and Spark.
- Online data processing: You want to obtain the analytical results of data with a strict requirement on the time it takes, such as real-time streaming data processing. Deeply integrated with Spark MLlib, GrapX, and SQL, Spark Streaming can be used to process streaming messages.

This section describes how to run an offline job called word count in E-MapReduce.

Process

OSS -> EMR -> Hadoop MapReduce

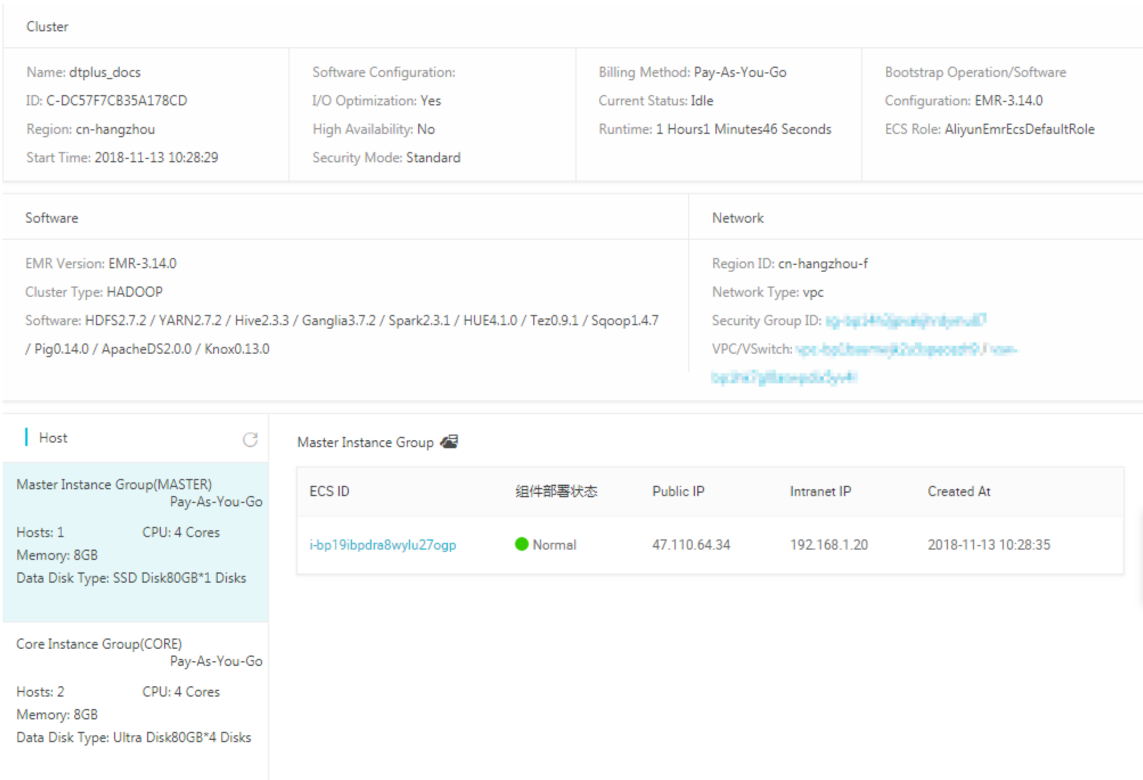
This process includes two steps:

1. Store data to OSS.
2. Read data from OSS and analyze the data by using E-MapReduce.

Prerequisites

- The following steps are performed in a Windows system. You need to ensure that Maven and Java have been installed and configured properly into your system.
- You can use E-MapReduce to automatically create a Hadoop cluster. For more information, see *Create a cluster*.
  - EMR Version: EMR-3.12.1
  - Cluster Type: HADOOP

- Software: HDFS2.7.2, YARN2.7.2, Hive2.3.3, Ganglia3.7.2, Spark2.3.1, HUE4.1.0, Zeppelin0.8.0, Tez0.9.1, Sqoop1.4.7, Pig0.14.0, ApacheDS2.0.0, and Knox0.13.0
- The network type of this Hadoop cluster is VPC in the China (Hangzhou) region. The master instance group is configured with a public IP and an internal network IP. The high availability mode is set to No (a non-HA mode). The following figure shows the details.



Procedures

1. Download sample code to your local disk.

   Open git bash in your system and execute the clone command as follows.

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

   Execute the `mvn install` command to compile the code.
2. For more information about how to create a bucket, see *Create a bucket*.

   Note:

   You must create a bucket and an E-MapReduce cluster in the same region.
3. Upload jar packages and resource files

   a. Log on to the *OSS console* and click the **Files** tab.

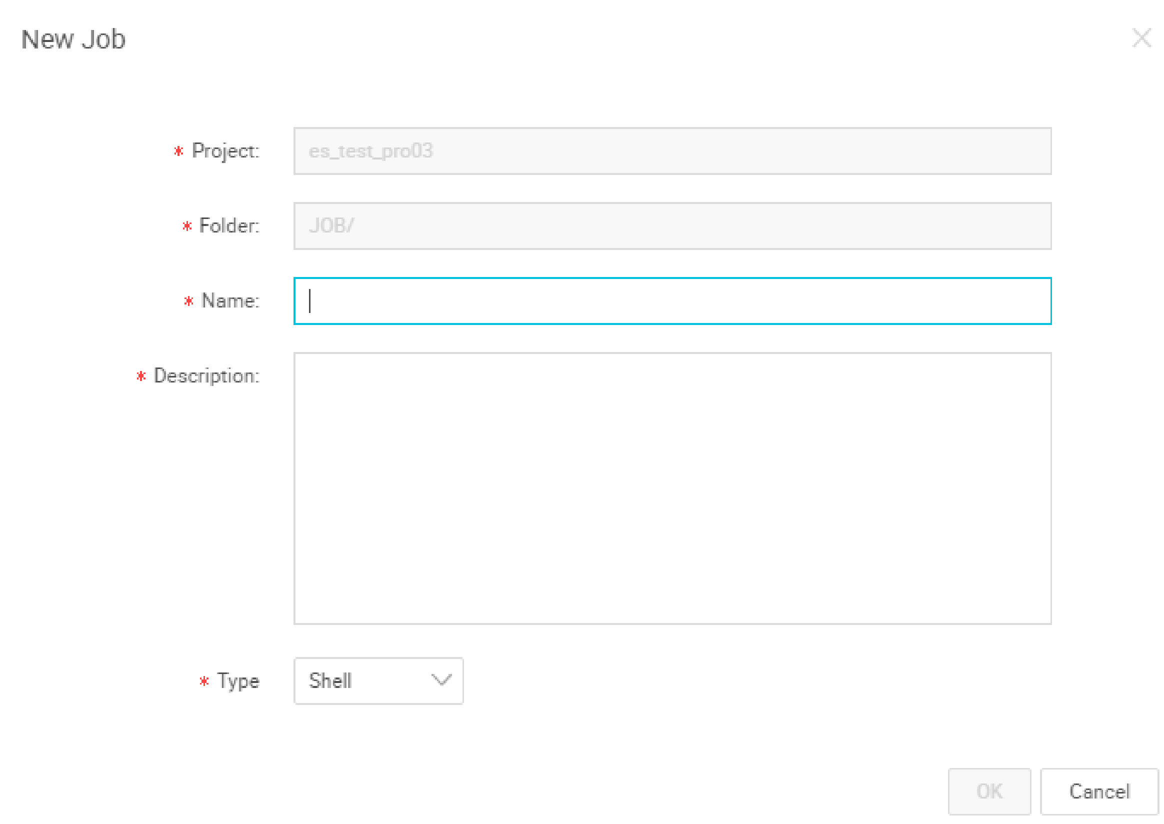
b. Click **Upload** to upload resources files in the `aliyun-emapreduce-demo/resources` directory and jar packages in the `aliyun-emapreduce-demo/target` directory.

4. Create a workflow project

   For more information, see *Workflow project management*.

5. Create a job

   For more information, see *Edit jobs*. Take a MapReduce job as an example.



6. After you configure a job, click **Run**. The following figure shows the details.

   - For more information about how to use OSS, see *OSS usage instructions*.
   - For more information about how to configure jobs, see the *Cite LeftjobCite Right* section of the E-MapReduce User Guide.

Note:

- If the OSS output URI already exists, an error occurs when you execute a job.
- When you click the Insert an OSS UNI button and select OSSREF as a File Prefix, E-MapReduce downloads OSS files to your cluster and add these files to a specified classpath.

- Currently, only OSS Standard storage is supported for all operations.

View logs

For more information about how to view logs of an execution plan, see *Connect to a cluster using SSH*.

3 Use E-MapReduce to collect metrics from a Kafka client

This section describes how to use E-MapReduce to collect metrics from a Kafka client to conduct effective performance monitoring.

Background

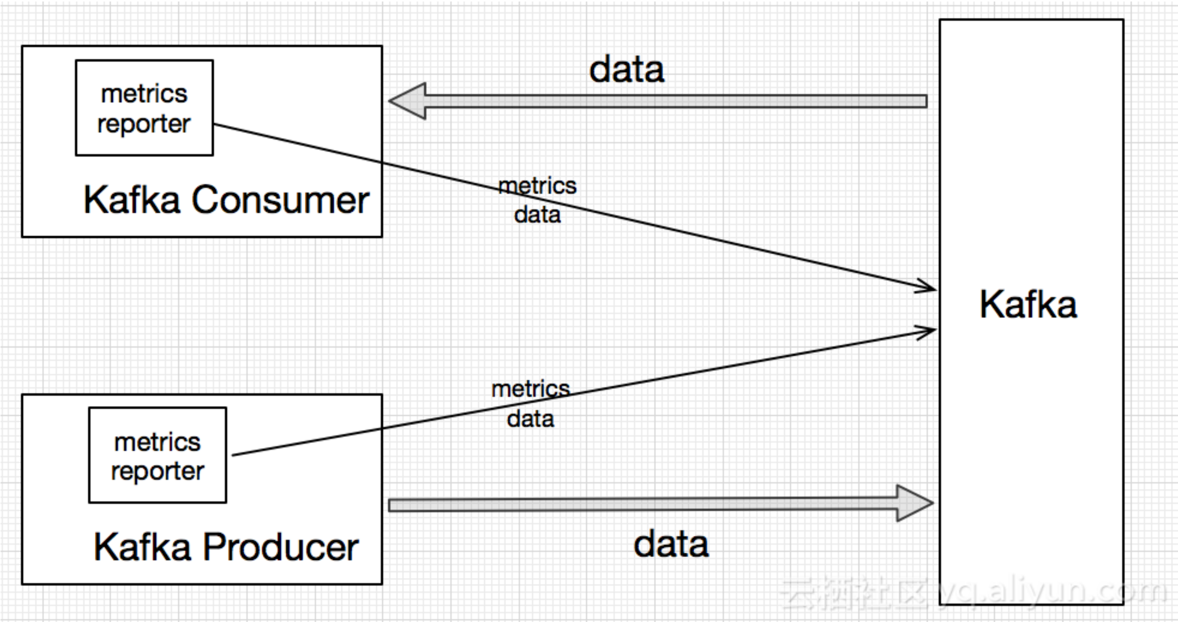
Kafka provides a collection of metrics that are used to measure the performance of Broker, Consumer, Producer, Stream, and Connect. E-MapReduce collects metrics for Kafka Broker by using Ganglia to monitor the running status of this Kafka Broker. A Kafka system consists of two roles: a Kafka Broker and multiple Kafka clients. When an issue of read/write performance occurs , you must perform an analysis on the both Kafka Broker and clients. Metrics from Kafka clients are important for performing the analysis.

Scenarios

- Collect metrics for Kafka performance

  Kafka supports multiple external Metrics Reporters. JMX Reporter is built in to Kafka by default . You can use the JMX tool to view metrics of Kafka. You can implement your own Metrics Reporter such as org.apache.kafka.common.metrics.MetricsReporter to collect custom metrics.
- Store metrics

  You can customize Kafka metrics. In addition, you need a data store to keep these metrics for later use and analysis. You can store metrics to Kafka without using a third-party data store as Kafka itself is a data store. In addition, Kafka can be easily integrated with other services. You can collect metrics from a client as the following figure shows:



E-MapReduce provides a sample emr-kafka-client-metrics. You can download the source code from the link: *source code*.

Test

Without compiling code by yourself, E-MapReduce has published the jar package in Maven. You can download the latest version from the *download link*.

- Configure metrics

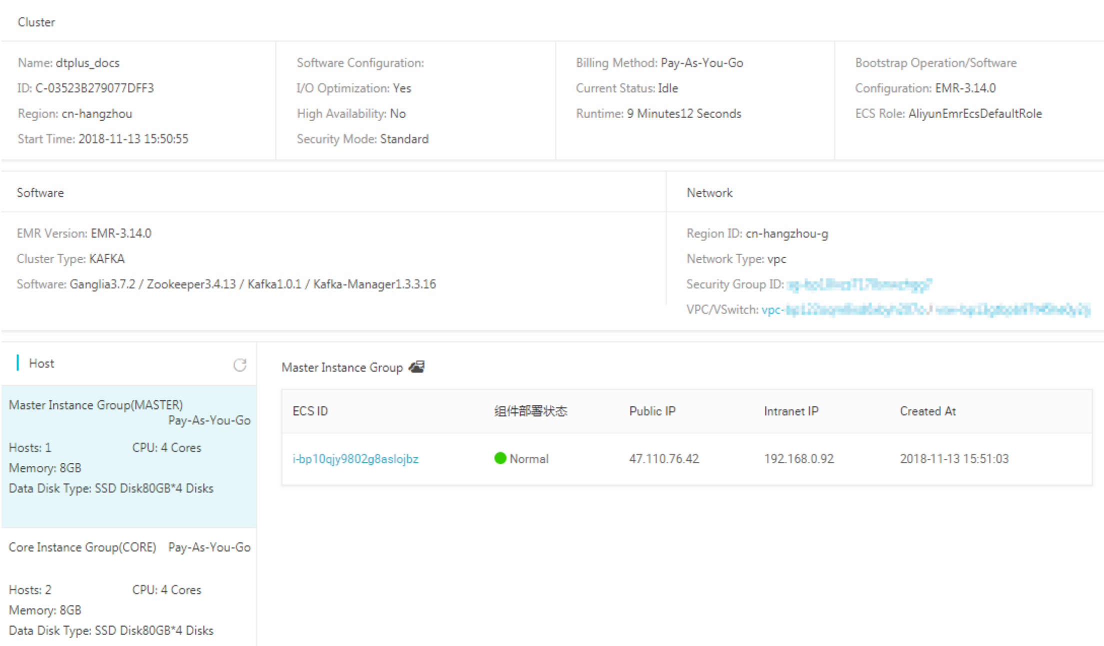
Metric	**Description**
metric.reporters	The sample Metrics Reporter: *org.apache.kafka.clients.reporter.EMRClientMetricsReporter*
emr.metrics.reporter.bootstrap.servers	The metrics that stores bootstrap.servers of a Kafka cluster.
emr.metrics.reporter.zookeeper.connect	The metrics that stores Zookeeper addresses of a Kafka cluster.

- Load metrics
  - Place the emr-kafka-client-metrics jar package on a client. Add the path of the jar package to the classpath of a client-side application.
  - Install the emr-kafka-client-metrics dependency on the jar package of a client-side applicatio n.
- Prerequisites

- In this section, we use E-MapReduce to automatically create a Kafka cluster. For more information, see *Create a cluster*.

  We use the following versions of E-MapReduce and Kafka:

  - EMR Version: EMR-3.12.1
  - Cluster Type: Kafka
  - Software: Kafka-Manager (1.3.3.16), Kafka (2.11-1.0.1), ZooKeeper (3.4.12), and Ganglia (3.7.2)
  - The network type of this Kafka cluster is VPC in the China (Hangzhou) region. The master instance group is configured with a public IP and an internal network IP. The following figure shows the details.



- Procedures

1. Download the latest emr-kafka-client-metrics package.

```
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. Copy the emr-kafka-client-metrics package to the lib directory of a Kafka client.

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

3. Create a test topic

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

4. Write data to a test topic. You can write the configurations of a Kafka Producer to the local client.conf file.

```
## client.conf:
metric.reporters=org.apache.kafka.clients.reporter.EMRClientM
etricsReporter
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
## Commnad:
kafka-producer-perf-test.sh --topic test-metrics --throughput 1000
 --num-records 100000
--record-size 1024 --producer.config client.conf
```

5. View the current metrics from a client. The default metrics topic is *_emr-client-metrics*.

```
Kafka-console-consumer.sh -- Topic _ emr-client-metrics --
Bootstrap-server emr-worker-1: 9092
--from-beginning
```

The returned message is shown as follows.

```
{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-metrics,
tag.client-id=producer-1}
```

Field name	Description:
client.ip	The IP address of a client host.
client.process	The process ID of a client-side application.
attribute	The attribute name of a metric.
value	The value of a metric.

Field name	Description:
timestamp	The timestamp when you collect a metric.
tag.xxx	Other tag information of a metric.

Note:

Restrictions

- Support for only Java applications
- Support for only clients of Kafka 0.10 or later

4 Submit Storm topologies to process data in Kafka on E-MapReduce

This topic describes how to deploy Storm clusters and Kafka clusters on E-MapReduce and run Storm topologies to consume data in Kafka.

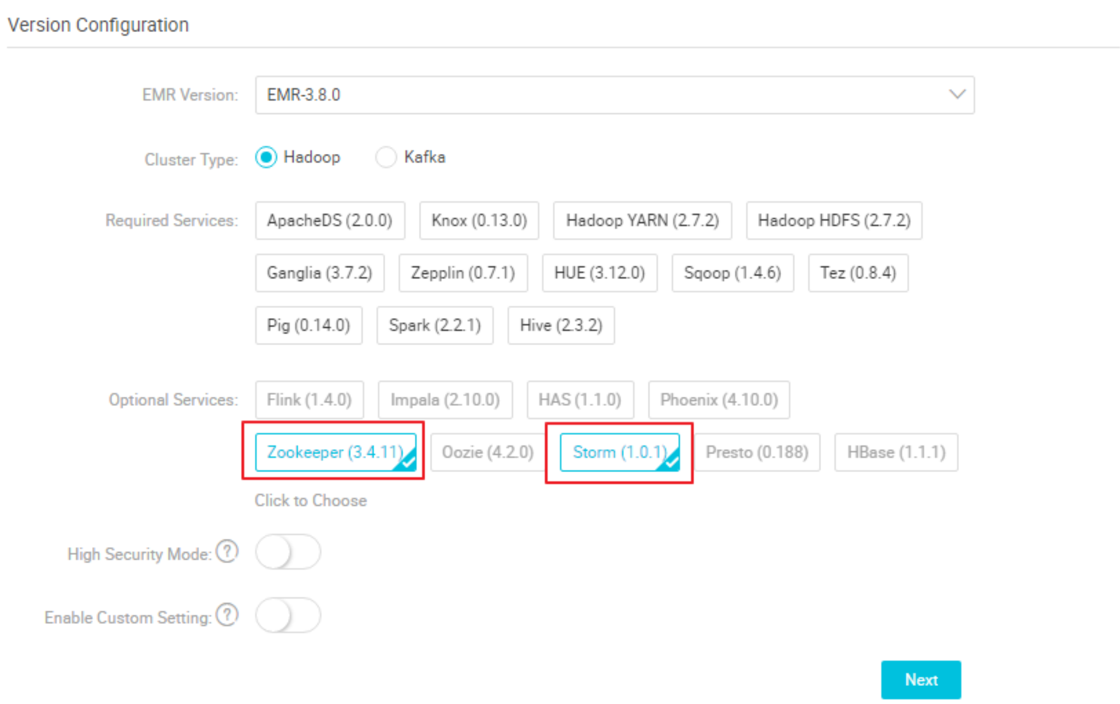
Prepare the environment

The test is performed using EMR that is deployed in the China East 1 (Hangzhou) region. The version of EMR is 3.8.0. The component versions required for this test are as follows.

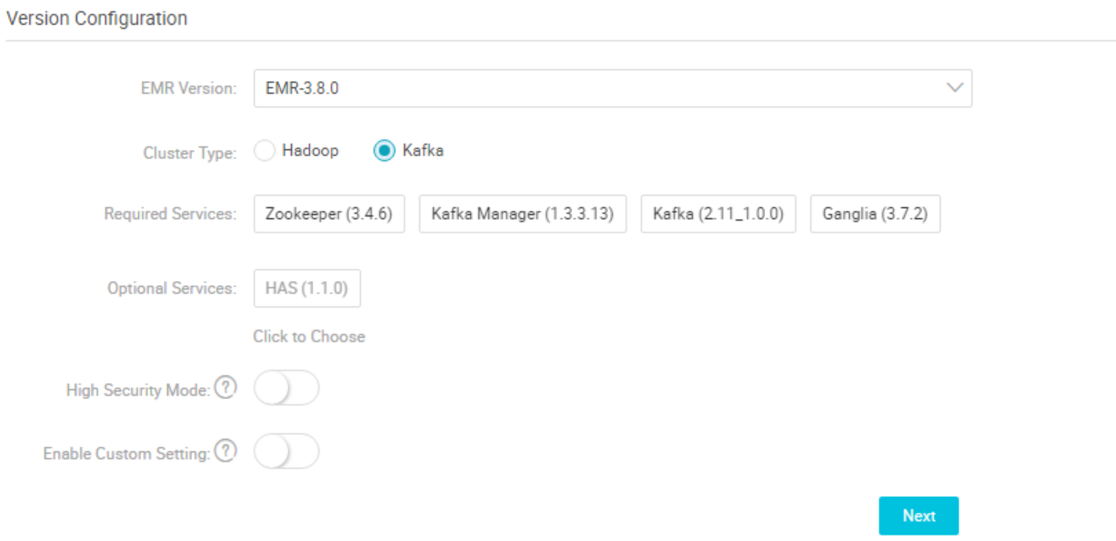
- Kafka: 2.11_1.0.0
- Storm: 1.0.1

In this topic, we use Alibaba Cloud E-MapReduce to create a Kafka cluster automatically. For more information, see *Create a cluster*.

- Create a Hadoop cluster



- Create a Kafka cluster



Note:

- If you choose classic network as the network type, put the Hadoop cluster and the Kafka cluster in the same security group to save time for configuring connections between instances.
- If you choose VPC as the network type, put the Hadoop cluster and the Kafka cluster in the same VPC and the same security group to save time for configuring a VPC peering connection.
- If you are familiar with networking and security groups for ECS, you can create configurat ions as needed.

- Configure the environment for Storm

  Consuming Kafka data fails if you run Storm topologies in the initial environment. To avoid such failures, you need to install the following dependencies for the Storm environment:

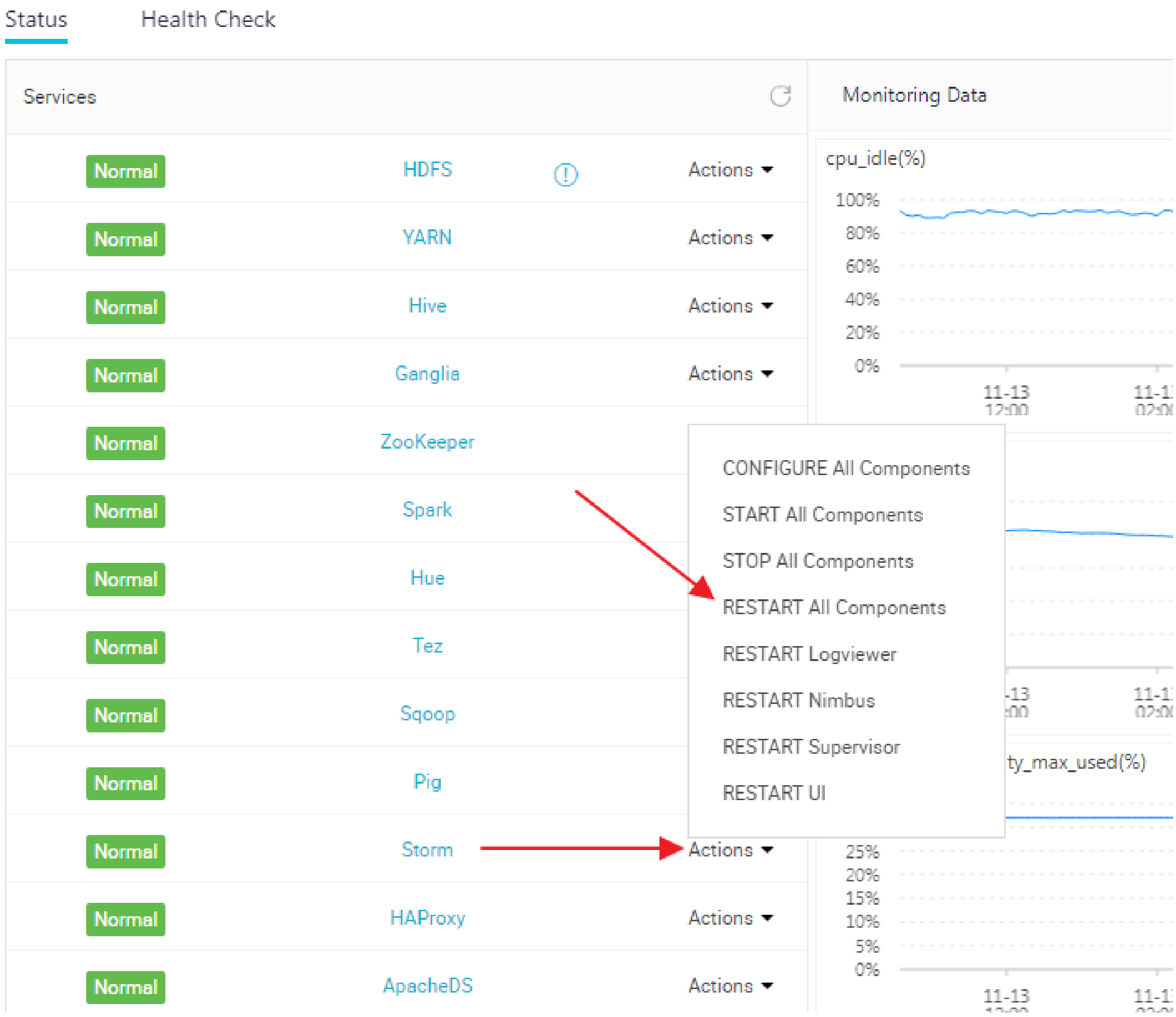
  - *curator-client*
  - *curator-framework*
  - *curator-recipes*
  - *json-simple*
  - *metrics-core*
  - *scala-library*
  - *zookeeper*
  - *commons-cli*

- *commons-collections*
- *commons-configuration*
- *htrace-core*
- *jcl-over-slf4j*
- *protobuf-java*
- *guava*
- *hadoop-common*
- *kafka-clients*
- *kafka*
- *storm-hdfs*
- *storm-kafka*

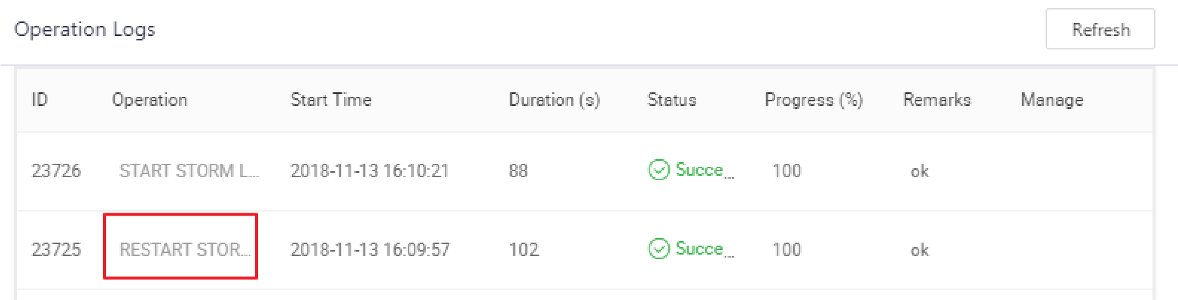
These dependencies have been tested. If you need additional dependencies, perform the following operations to add them to the lib folder of Storm.

```
[hadoop@emr-header-1 ~]$ ll
total 8524
-rw-rw-r-- 1 hadoop hadoop   52988 Jun 14  2015 commons-cli-1.3.1.jar
-rw-rw-r-- 1 hadoop hadoop  588337 Nov 13  2015 commons-collections-3.2.2.jar
-rw-rw-r-- 1 hadoop hadoop  298829 Feb  5  2009 commons-configuration-1.6.jar
-rw-r--r-- 1 root   root     73448 Feb  9 14:01 curator-client-2.10.0.jar
-rw-r--r-- 1 root   root    195437 Feb  9 14:01 curator-framework-2.10.0.jar
-rw-r--r-- 1 root   root    281476 Feb  9 14:01 curator-recipes-2.10.0.jar
-rw-rw-r-- 1 hadoop hadoop   31212 Apr 19  2014 htrace-core-3.0.4.jar
-rw-rw-r-- 1 hadoop hadoop   17289 Jun 11  2012 jcl-over-slf4j-1.6.6.jar
-rw-rw-r-- 1 hadoop hadoop   16046 Aug 13  2009 json-simple-1.1.jar
-rw-rw-r-- 1 hadoop hadoop   82123 Nov 27  2012 metrics-core-2.2.0.jar
-rw-rw-r-- 1 hadoop hadoop  533455 Mar  8  2013 protobuf-java-2.5.0.jar
-rw-r--r-- 1 root   root   5745606 Feb  9 14:01 scala-library-2.11.7.jar
-rw-rw-r-- 1 hadoop hadoop  792964 Feb 24  2014 zookeeper-3.4.6.jar
[hadoop@emr-header-1 ~]$ pwd
/home/hadoop
[hadoop@emr-header-1 ~]$ sudo cp ./* /usr/lib/storm-current/lib/
```

You need to perform the preceding operations on each node in the Hadoop cluster. After the operations are complete, restart Storm in the E-MapReduce console as shown in the following figure.



You can view operation logs to check the status of Storm:


Operation Logs
Refresh

ID	Operation	Start Time	Duration (s)	Status	Progress (%)	Remarks	Manage
23726	START STORM L...	2018-11-13 16:10:21	88	Succe...	100	ok	
23725	RESTART STOR...	2018-11-13 16:09:57	102	Succe...	100	ok	

Create Storm topologies and Kafka topics

- E-MapReduce provides sample code that you can use directly. The links are as follows:
  - *e-mapreduce-demo*
  - *e-mapreduce-sdk*
- Write data to topics

1. Log on to the Kafka cluster.
2. Create a test topic with 10 partitions and 2 replicas.

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

3. Write 100 records of data to the test topic.

```
/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
```

Note:

The preceding command is run on the emr-header-1 node in the Kafka cluster. You can also run the command on client nodes.

- Run a Storm topology.

  Log on to the Hadoop cluster, copy the *examples-1.1-shaded.jar* file (the test topic data is compiled to this file in step 2) to the emr-header-1 node. In this example, the file is stored in the HDFS root directory. Run the following command to submit the topology:

```
/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
```

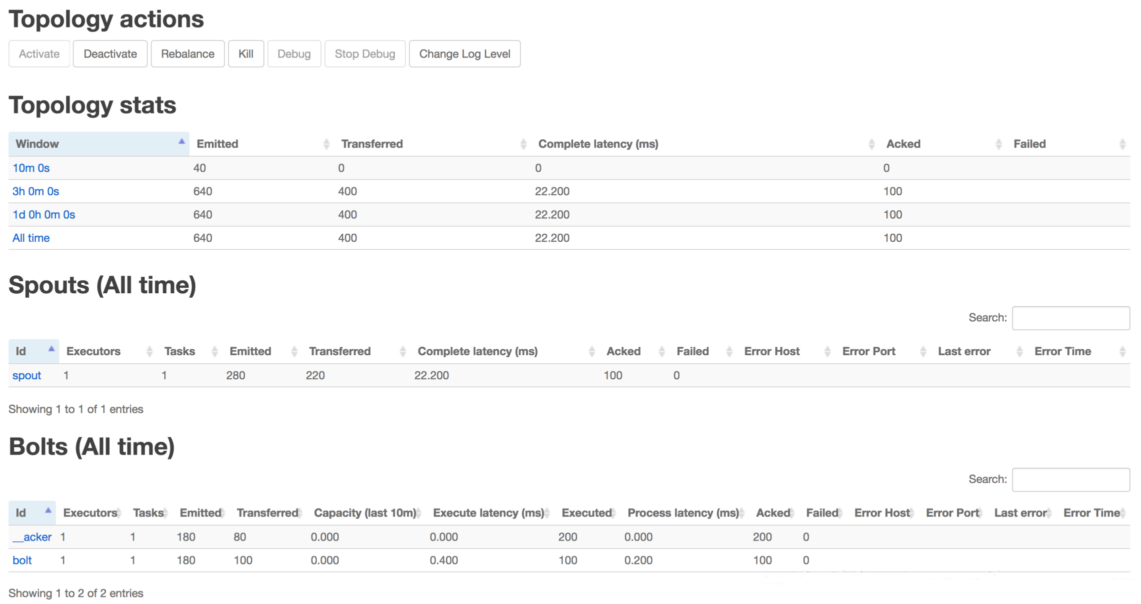
- View the running status of a topology

  - View the running status of Storm

    You can use the Web UI to view the services on a cluster in the following ways:

    - With Knox. For more information, see *Knox instructions*.
    - Use SSH. For more information, see *Use SSH to log on to a cluster*.

    In this topic, we use SSH to access the Web UI. The endpoint is *http://localhost:9999/index.html*. You can see the topology that we have submitted. Click the topology to view the running logs:



- View the output files in HDFS
    - View the output files in HDFS.

```
[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-2-0-1518327441
777.txt | wc -l
200
```

    - Write 120 records of data to the test topic in Kafka.

```
[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.
```

    - Output the line number of the HDFS file.

```
[root@emr-header-1 ~]# hadoop fs -cat /foo/bolt-2-0-1518327441
777.txt | wc -l
320
```

Summary

We have successfully deployed a Storm cluster and a Kafka cluster on E-MapReduce, run a Storm topology and consumed Kafka data. E-MapReduce also supports the Spark streaming and the Flink components, which can run in Hadoop clusters and process Kafka data.

Note:

E-MapReduce does not provide the Storm cluster option. Therefore, we have created a Hadoop cluster and have installed the Storm components. If you do not need to use other components, you can easily disable them in the E-MapReduce console. Then a Hadoop cluster is equivalent to a Storm cluster.

5 Use ES-Hadoop on E-MapReduce

ES-Hadoop is a tool used to connect the Hadoop ecosystem provided by Elasticsearch (ES). It enables users to use tools such as MapReduce (MR), Spark, and Hive to process data in ES (ES-Hadoop also supports taking a snapshot of ES indices and storing it in HDFS, which is not discussed in this topic).

Background

We know that the advantage of the Hadoop ecosystem is processing large data sets. But the disadvantage is also obvious: interactive analysis can be delayed. ES is adept at many types of queries, especially ad-hoc queries. Subsecond response time has been reached. ES-Hadoop has combined both advantages. With ES-Hadoop, users only need to make small changes to the code for quickly processing data stored in ES. ES also provides acceleration.

ES-Hadoop uses ES as the data source of data processing engines, such as MR, Spark, and Hive. ES plays the role of storage in architectures where compute and storage are separated. This is the same for other data sources of MR, Spark, and Hive. But ES has faster data filtering ability compared with other data sources. This ability is one of the most critical abilities of an analytics engine.

EMR has already integrated with ES-Hadoop. Users can use ES-Hadoop directly without any configurations. The following examples introduce ES-Hadoop on EMR.

Preparation

ES can automatically create indices and identify data types based on input data. In some cases, this feature is helpful, by avoiding many actions by users. However, it also cause problems. The biggest problem is that sometimes the data types identified by ES are not correct. For example, we define a field called `age`. The data type of this column is INT but it may be identified as LONG in the ES index. Users need to convert data types when performing some specified actions. We recommend that you create indices manually to avoid such problems.

In the following examples, we use the `company` index and the `employees'` type (you can consider an ES index as a database and a type as a table in the database). This type defines four fields (field types are defined by ES).

```
{
  "id": long,
  "name": text,
  "age": integer,
  "birth": date
```

```
}
```

Run the following commands to create an index in Kibana (you can also use cURL commands):

```
PUT company
{
  "mappings": {
    "employees": {
      "properties": {
        "id": {
          "type": "long"
        },
        "name": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "birth": {
          "type": "date"
        },
        "addr": {
          "type": "text"
        }
      }
    }
  },
  "settings": {
    "index": {
      "number_of_shards": "5",
      "number_of_replicas": "1"
    }
  }
}
```

Note:

Specify the index parameters in settings as needed. This step is optional.

Prepare a file where each row is a JSON object as follows:

```
{"id": 1, "name": "zhangsan", "birth": "1990-01-01", "addr": "No. 969
, wenyixi Rd, yuhang, hangzhou"}
{"id": 2, "name": "lisi", "birth": "1991-01-01", "addr": "No. 556,
xixi Rd, xihu, hangzhou"}
{"id": 3, "name": "wangwu", "birth": "1992-01-01", "addr": "No. 699
wangshang Rd, binjiang, hangzhou"}
```

Save the file to the specified directory in HDFS (for example, `/es-hadoop/employees.txt`).

Mapreduce

In the following example, we read the JSON files in the `/es-hadoop` directory in HDFS and write each row in the JSON files into ES as a document. Writing is finished in the map stage through EsOutputFormat.

Use the following options to set ES.

- es.nodes: ES nodes. The formats is host:port. For ES hosted on Alibaba Cloud, set the value to the endpoint of ES provided by Alibaba Cloud.
- es.net.http.auth.user: Username.
- es.net.http.auth.pass: Password.
- es.nodes.wan.only: For ES hosted on Alibaba Cloud, set the value to `true`.
- es.resource: The indices and types of ES.
- es.input.json: If the input file is in JSON format, set the value to `true`. Otherwise, you need to parse the input data using the map() function and output the corresponding Writable class.

Note:

Disable speculative execution for map tasks and reduce tasks

```
package com.aliyun.emr;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.elasticsearch.hadoop.mr.EsOutputFormat;

public class Test implements Tool {

  private Configuration conf;

  @Override
  public int run(String[] args) throws Exception {

    String[] otherArgs = new GenericOptionsParser(conf, args).
getRemainingArgs();

    conf.setBoolean("mapreduce.map.speculative", false);
    conf.setBoolean("mapreduce.reduce.speculative", false);
    conf.set("es.nodes", "<your_es_host>:9200");
    conf.set("es.net.http.auth.user", "<your_username>");
```

```
    conf.set("es.net.http.auth.pass", "<your_password>");
    conf.set("es.nodes.wan.only", "true");
    conf.set("es.resource", "company/employees");
    conf.set("es.input.json", "yes");

    Job job = Job.getInstance(conf);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(EsOutputFormat.class);
    job.setMapOutputKeyClass(NullWritable.class);
    job.setMapOutputValueClass(Text.class);
    job.setJarByClass(Test.class);
    job.setMapperClass(EsMapper.class);

    FileInputFormat.setInputPaths(job, new Path(otherArgs[0]));

    return job.waitForCompletion(true) ? 0 : 1;
  }

  @Override
  public void setConf(Configuration conf) {
    this.conf = conf;
  }

  @Override
  public Configuration getConf() {
    return conf;
  }

  public static class EsMapper extends Mapper<Object, Text, NullWritab
le, Text> {
    private Text doc = new Text();

    @Override
    protected void map(Object key, Text value, Context context) throws
 IOException, InterruptedException {
      if (value.getLength() > 0) {
        doc.set(value);
        context.write(NullWritable.get(), doc);
      }
    }
  }

  public static void main(String[] args) throws Exception {
    int ret = ToolRunner.run(new Test(), args);
    System.exit(ret);
  }
}
```

Compile and package the code into a JAR file called *mr-test.jar*. Submit it to an instance that has installed an EMR client program (such as a gateway, or any node in an EMR cluster).

Run the following commands on any node that has installed an EMR client to run the MapReduce program:

```
hadoop jar mr-test.jar com.aliyun.emr.Test -Dmapreduce.job.reduces=0 -
libjars mr-test.jar /es-hadoop
```

At this point, writing data to ES has finished. You can query the written data through Kibana (or by using the cURL commands).

```
GET
{
  "query": {
    "match_all": {}
  }
}
```

Spark

In this example, we write data to an index in ES using Spark instead of MapReduce. Spark persists a resilient distributed dataset (RDD) to ES using the JavaEsSpark class. Users also need to use the options mentioned above in the MapReduce section to set ES.

```
package com.aliyun.emr;

import java.util.Map;
import java.util.concurrent.atomic.AtomicInteger;
import org.apache.spark.SparkConf;
import org.apache.spark.SparkContext;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
import org.elasticsearch.spark.rdd.api.java.JavaEsSpark;
import org.spark_project.guava.collect.ImmutableMap;

public class Test {

  public static void main(String[] args) {
    SparkConf conf = new SparkConf();
    conf.setAppName("Es-test");
    conf.set("es.nodes", "<your_es_host>:9200");
    conf.set("es.net.http.auth.user", "<your_username>");
    conf.set("es.net.http.auth.pass", "<your_password>");
    conf.set("es.nodes.wan.only", "true");

    SparkSession ss = new SparkSession(new SparkContext(conf));
    final AtomicInteger employeesNo = new AtomicInteger(0);
    JavaRDD<Map<Object, ? >> javaRDD = ss.read().text("hdfs://emr-
header-1:9000/es-hadoop/employees.txt")
        .javaRDD().map((Function<Row, Map<Object, ? >>) row ->
ImmutableMap.of("employees" + employeesNo.getAndAdd(1), row.mkString
()));

    JavaEsSpark.saveToEs(javaRDD, "company/employees");
  }
```

```
}
```

Package the code in a JAR file called spark-test.jar. Run the following command to write data:

```
spark-submit --master yarn --class com.aliyun.emr.Test spark-test.jar
```

After the task has finished, you can query the results through Kibana or the cURL commands.

In addition to **`Spark RDD`**. ES-Hadoop also provides a Spark SQL component to read and write ES data. For more information, see the *official website* of ES-Hadoop.

Hive

This example introduces SQL statements to read and write ES data through Hive.

First, run the **`hive`**command to enter CLI and create a table:

```
CREATE DATABASE IF NOT EXISTS company;
```

Then create an external table that is stored in ES. Specify the option using TBLPROPERTIES.

```
CREATE EXTERNAL table IF NOT EXISTS employees(
  id BIGINT,
  name STRING,
  birth TIMESTAMP,
  addr STRING
)
STORED BY 'org.elasticsearch.hadoop.hive.EsStorageHandler'
TBLPROPERTIES(
     'es.resource' = 'tpcds/ss',
     'es.nodes' = '<your_es_host>',
     'es.net.http.auth.user' = '<your_username>',
     'es.net.http.auth.pass' = '<your_password>',
     'es.nodes.wan.only' = 'true',
     'es.resource' = 'company/employees'
);
```

Note:

We set the data type of the birth columns to TIMESTAMP in the Hive table. In ES, we set it to DATE. This is because Hive and EC handle data types differently. Parsing of converted date data can fail when Hive writes data to ES. In contrast, parsing of returned data can also fail when Hive reads ES data. For more information, click *here* .

Insert some data into the table:

```
INSERT INTO TABLE employees VALUES (1, "zhangsan", "1990-01-01","No.
969, wenyixi Rd, yuhang, hangzhou");
INSERT INTO TABLE employees VALUES (2, "lisi", "1991-01-01", "No. 556
, xixi Rd, xihu, hangzhou");
```

```
INSERT INTO TABLE employees VALUES (3, "wangwu", "1992-01-01", "No.
699 wangshang Rd, binjiang, hangzhou");
```

Execute queries to view the results:

```
SELECT * FROM employees LIMIT 100;
OK
1    zhangsan    1990-01-01    No. 969, wenyixi Rd, yuhang, hangzhou
2    lisi    1991-01-01    No. 556, xixi Rd, xihu, hangzhou
3    wangwu    1992-01-01    No. 699 wangshang Rd, binjiang, hangzhou
```

6 Use Mongo-Hadoop on E-MapReduce

Mongo-Hadoop is a component provided by MongoDB for Hadoop components to connect to MongoDB. Using Mongo-Hadoop is similar to using ES-Hadoop which is described in the previous topic. EMR has already integrated with Mongo-Hadoop. Users can directly use Mongo-Hadoop without any deployment configuration. This topic describes how to use Mongo-Hadoop using some examples.

Preparation

We use the same data model for the following examples:

```
{
  "id": long,
  "name": text,
  "age": integer,
  "birth": date
}
```

We write data into the specified collection (similar to a table in a database) in a MongoDB database. Therefore, we need to first ensure that the collection exists in the MongoDB database. First, run the MongoDB client program on a client node that can access the MongoDB database. You may need to download the client program from the MongoDB website and install it. Take the connection to ApsaraDB for MongoDB as an example:

```
mongo --host dds-xxxxxxxxxxxxxxxxxxxxx.mongodb.rds.aliyuncs.com:3717
 --authenticationDatabase admin -u root -p 123456
```

The hostname of the MongoDB database is dds-xxxxxxxxxxxxxxxxxxxxx.mongodb.rds.aliyuncs.com. The port number is 3717. The actual port number depends on the MongoDB cluster. For an external MongoDB cluster that you have deployed on your own, the default port number is 27017. In this example, the password is set to 123456 using the -p option. Run the following commands in CLI to create a collection named employees in the company database:

```
> use company;
> db.createCollection("employees")
```

Prepare a file where each row is a JSON object as follows.

```
{"id": 1, "name": "zhangsan", "birth": "1990-01-01", "addr": "No. 969
, wenyixi Rd, yuhang, hangzhou"}
{"id": 2, "name": "lisi", "birth": "1991-01-01", "addr": "No. 556,
xixi Rd, xihu, hangzhou"}
```

```
{"id": 3, "name": "wangwu", "birth": "1992-01-01", "addr": "No. 699
wangshang Rd, binjiang, hangzhou"}
```

Save the file to the specified directory on HDFS (for example, the file path can be */mongo-hadoop/employees.txt*).

Mapreduce

In the following example, we read JSON files in the */mongo-hadoop* directory on HDFS and write each row in the JSON files as a document to the MongoDB database.

```
package com.aliyun.emr;

import com.mongodb.BasicDBObject;
import com.mongodb.hadoop.MongoOutputFormat;
import com.mongodb.hadoop.io.BSONWritable;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class Test implements Tool {

  private Configuration conf;

  @Override
  public int run(String[] args) throws Exception {

    String[] otherArgs = new GenericOptionsParser(conf, args).
getRemainingArgs();

    conf.set("mongo.output.uri", "mongodb://<your_username>:<
your_password>@dds-xxxxxxxxxxxxxxxxxxxx.mongodb.rds.aliyuncs.com:3717
/company.employees? authSource=admin");

    Job job = Job.getInstance(conf);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(MongoOutputFormat.class);
    job.setOutputKeyClass(Text.class);
    job.setMapOutputValueClass(BSONWritable.class);

    job.setJarByClass(Test.class);
    job.setMapperClass(MongoMapper.class);

    FileInputFormat.setInputPaths(job, new Path(otherArgs[0]));

    return job.waitForCompletion(true) ? 0 : 1;
  }

  @Override
  public Configuration getConf() {
    return conf;
```

```
  }

  @Override
  public void setConf(Configuration conf) {
    this.conf = conf;
  }

  public static class MongoMapper extends Mapper<Object, Text, Text,
BSONWritable> {

    private BSONWritable doc = new BSONWritable();
    private int employeeNo = 1;
    private Text id;

    @Override
    protected void map(Object key, Text value, Context context) throws
 IOException, InterruptedException {
      if (value.getLength() > 0) {
        doc.setDoc(BasicDBObject.parse(value.toString()));
        id = new Text("employee" + employeeNo++);
        context.write(id, doc);
      }
    }
  }

  public static void main(String[] args) throws Exception {
    int ret = ToolRunner.run(new Test(), args);
    System.exit(ret);
  }
}
```

Compile and package the code into a JAR file called *mr-test.jar*. Run the following command:

```
hadoop jar mr-test.jar com.aliyun.emr.Test -Dmapreduce.job.reduces=0 -
libjars mr-test.jar /mongo-hadoop
```

After the execution is complete, you can view the results using the MongoDB client program:

```
> db.employees.find();
{ "_id" : "employee1", "id" : 1, "name" : "zhangsan", "birth" : "1990-
01-01", "addr" : "No. 969, wenyixi Rd, yuhang, hangzhou" }
{ "_id" : "employee2", "id" : 2, "name" : "lisi", "birth" : "1991-01-
01", "addr" : "No. 556, xixi Rd, xihu, hangzhou" }
{ "_id" : "employee3", "id" : 3, "name" : "wangwu", "birth" : "1992-01
-01", "addr" : "No. 699 wangshang Rd, binjiang, hangzhou" }
```

Spark

In this example, we write data to a MongoDB database using Spark instead of MapReduce.

```
package com.aliyun.emr;

import com.mongodb.BasicDBObject;
import com.mongodb.hadoop.MongoOutputFormat;
import java.util.concurrent.atomic.AtomicInteger;
import org.apache.hadoop.conf.Configuration;
import org.apache.spark.SparkContext;
```

```
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
import org.bson.BSONObject;
import scala.Tuple2;

public class Test {

  public static void main(String[] args) {

    SparkSession ss = new SparkSession(new SparkContext());

    final AtomicInteger employeeNo = new AtomicInteger(0);
    JavaRDD<Tuple2<Object, BSONObject>> javaRDD =
        ss.read().text("hdfs://emr-header-1:9000/mongo-hadoop/
employees.txt")
            .javaRDD().map((Function<Row, Tuple2<Object, BSONObject
>>) row -> {
          BSONObject bson = BasicDBObject.parse(row.mkString());
          return new Tuple2<>("employee" + employeeNo.getAndAdd(1),
bson);
        });

    JavaPairRDD<Object, BSONObject> documents = JavaPairRDD.fromJavaRD
D(javaRDD);

    Configuration outputConfig = new Configuration();
    outputConfig.set("mongo.output.uri", "mongodb://<your_username>:<
your_password>@dds-xxxxxxxxxxxxxxxxxxxx.mongodb.rds.aliyuncs.com:3717
/company.employees? authSource=admin");

    // It is saved as a "Hadoop file." Actually, the data is written
into the MongoDB database through the MongoOutputFormat class.
    documents.saveAsNewAPIHadoopFile(
        "file:///this-is-completely-unused",
        Object.class,
        BSONObject.class,
        MongoOutputFormat.class,
        outputConfig
    );
  }
}
```

Package the code into a JAR file named *spark-test.jar*. Run the following command to write data.

```
spark-submit --master yarn --class com.aliyun.emr.Test spark-test.jar
```

After the writing has finished, you can use the MongoDB client to view the results.

Hive

This example describes how to use Hive to read and write data in MongoDB databases through SQL statements.

First, run the **hive** command to enter CLI mode and create a table:

```
CREATE DATABASE IF NOT EXISTS company;
```

You need to create an external table that is stored in a MongoDB database. Before you do that, create a MongoDB collection named employees as described in the Preparation section.

Go back to CLI mode, execute the following SQL statements to create an external table. Connection to MongoDB is set through the TBLPROPERTIES clause.

```
CREATE EXTERNAL TABLE IF NOT EXISTS employees(
  id BIGINT,
  name STRING,
  birth STRING,
  addr STRING
)
STORED BY 'com.mongodb.hadoop.hive.MongoStorageHandler'
WITH SERDEPROPERTIES('mongo.columns.mapping'='{"id":"_id"}')
TBLPROPERTIES('mongo.uri'='mongodb://<your_username>:<your_password
>@dds-xxxxxxxxxxxxxxxxxxxx.mongodb.rds.aliyuncs.com:3717/company.
employees? authSource=admin');
```

Note:

Values of the *id* column in Hive are mapped to values of the *_id* column in MongoDB through SERDEPROPERTIES. You can map column values as needed. Note that the data type of the birth column is set to STRING. The reason is that Hive and MongoDB handle DATE format differently. After Hive sends data in DATE format to MongoDB, NULL may be returned when the data is queried in Hive.

Insert some data into the table:

```
INSERT INTO TABLE employees VALUES (1, "zhangsan", "1990-01-01","No.
969, wenyixi Rd, yuhang, hangzhou");
INSERT INTO TABLE employees VALUES (2, "lisi", "1991-01-01", "No. 556
, xixi Rd, xihu, hangzhou");
INSERT INTO TABLE employees VALUES (3, "wangwu", "1992-01-01", "No.
699 wangshang Rd, binjiang, hangzhou");
```

Execute the following statement to see the results:

```
SELECT * FROM employees LIMIT 100;
OK
1    zhangsan    1990-01-01    No. 969, wenyixi Rd, yuhang, hangzhou
2    lisi    1991-01-01    No. 556, xixi Rd, xihu, hangzhou
3    wangwu    1992-01-01    No. 699 wangshang Rd, binjiang, hangzhou
```

7 Deep learning with Analytics Zoo on E-MapReduce

Analytics Zoo is an analytics and AI platform that unites Apache Spark and Intel BigDL into an integrated pipeline. It helps users develop deep learning applications based on big data and end-to-end pipelines. This topic describes how to use Analytics Zoo to develop deep learning applications on Alibaba Cloud E-MapReduce.

Introduction

Analytics Zoo is an analytics and AI platform that unites Apache Spark and Intel BigDL into an integrated pipeline. It helps users develop deep learning applications based on big data and end-to-end pipelines.

System requirements

- JDK 8
- Spark cluster (Spark 2.x supported by EMR is recommended)
- Python 2.7(also Python 3.5 or Python 3.6), pip

Installation of Analytics Zoo

- The latest release of Analytics Zoo is 0.2.0.
- Installation for Scala users
  - Download the pre-build version.

    You can download the *Pre-build version* from the Analytics Zoo page on GitHub.
  - Build Analytics Zoo using the make-dist.sh script.

    Install Apache Maven and set the environment variable MAVEN_OPTS as follows:

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

    If you use ECS instances to compile code, we recommend that you modify the mirror of the Maven repository.

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

```
</mirror>
```

Download an *Analytics Zoo release*. Extract the file, move to the corresponding directory, and run the following command:

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

After building Analytics Zoo, you can find a dist directory, which contains all the needed files to run an Analytics Zoo program. Use the following command to copy the files in the dist directory to the directory of the EMR software stack:

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

- Installation for Python users

  Analytics Zoo can be installed either with pip or without pip. When you install Analytics Zoo with pip, PySpark and BigDL are installed. This may cause a software conflict because PySpark has already been installed on the EMR cluster. To avoid such conflicts, install Analytics Zoo without pip.

  - Installation without pip

    First, you need to run the following command:

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

    Change to the pyzoo directory and install Analytics Zoo:

    ```
    python setup.py install
    ```

- Setting environment variables

  After building Analytics Zoo, copy the dist directory to the directory of the EMR software stack and set the environment variable. Add the following lines to the *`/etc/profile.d/analytics_zoo.sh`* file.

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

  You do not need to set SPARK_HOME because it has already been set on EMR.

Using Analytics Zoo

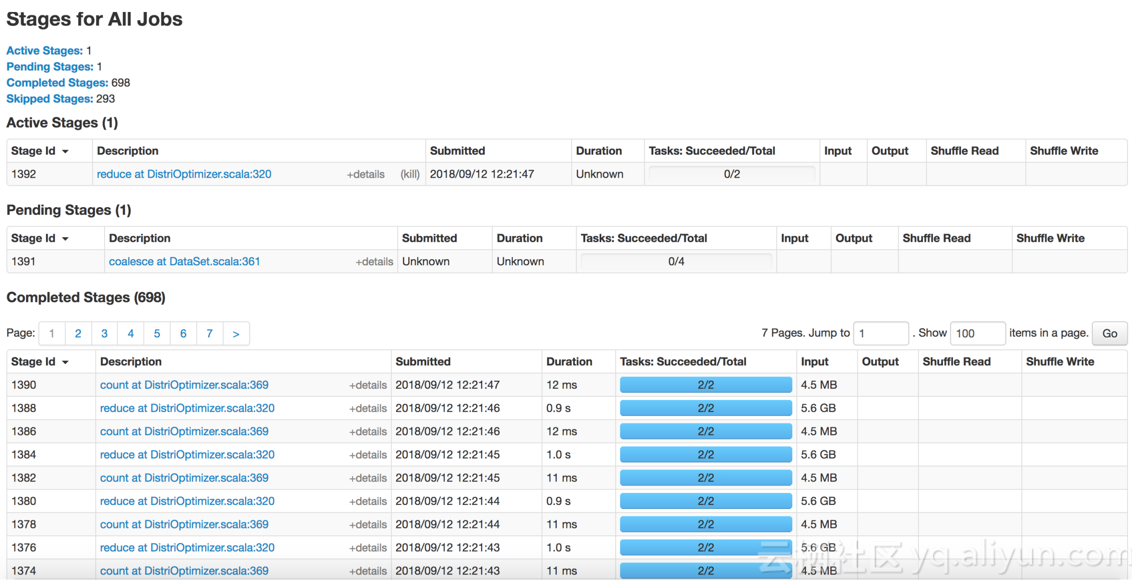
- Use Spark to train and test deep learning models.

- Use Analytics Zoo to do text classification. You can find the code and description on *GitHub*.

  Download the required data as required. Submit the following commands:

```
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
```

- You can log on to the instance of the Spark cluster through *ssh proxy* to view the status of the jobs.



  You can also view the accuracy of each epoch through logs.

```
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
.
```

- Use PySpark and Jupyter to train deep learning models on Analytics Zoo.

- Install Jupyter.

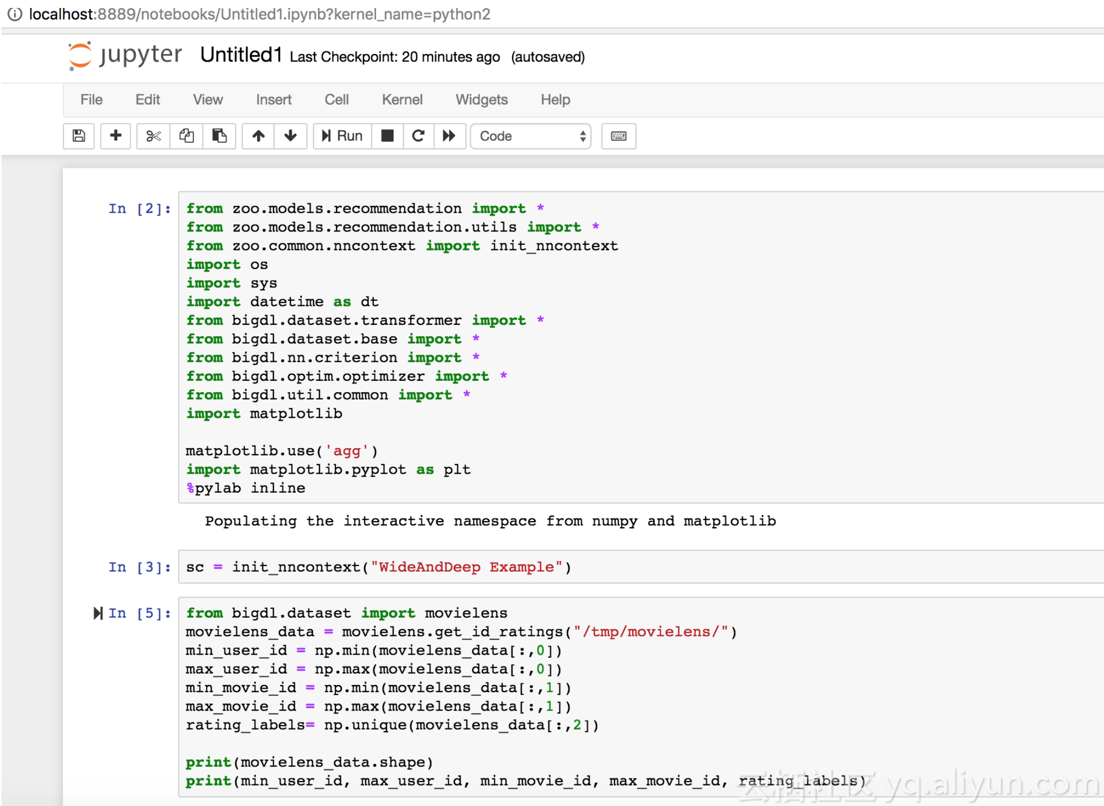
```
pip install jupyter
```

- Run the following command to start Jupyter.

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

- We recommend that you use the pre-defined Wide And Deep Learning models provided by Analytics Zoo.

1. Import data.



2. Build a model and create an optimizer.

```
In [10]: wide_n_deep = WideAndDeep(5, column_info, "wide_n_deep")

creating: createZooWideAndDeep
```

```
In [11]: # Create an Optimizer
batch_size = 8000

optimizer = Optimizer(
    model=wide_n_deep,
    training_rdd=train_data,
    criterion=ClassNLLCriterion(),
    optim_method=Adam(learningrate = 0.001, learningrate_decay=0.00005),
    end_trigger=MaxEpoch(10),
    batch_size=batch_size)

# Set the validation logic
optimizer.set_validation(
    batch_size=batch_size,
    val_rdd=test_data,
    trigger=EveryEpoch(),
    val_method=[Top1Accuracy(), Loss(ClassNLLCriterion())]
)
log_dir='/tmp/bigdl_summaries/'
app_name='wide_n_deep-'+dt.datetime.now().strftime("%Y%m%d-%H%M%S")
train_summary = TrainSummary(log_dir=log_dir,
                             app_name=app_name)
val_summary = ValidationSummary(log_dir=log_dir,
                                app_name=app_name)
optimizer.set_train_summary(train_summary)
optimizer.set_val_summary(val_summary)
print("saving logs to %s" % (log_dir + app_name))
```

云栖社区 yq.aliyun.com

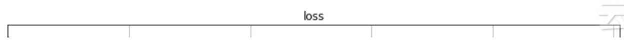
3. Start the training process.

```
In [12]: %%time
# Boot training process
optimizer.optimize()
print("Optimization Done.")

Optimization Done.
CPU times: user 85.9 ms, sys: 16.7 ms, total: 103 ms
Wall time: 2min 52s
```

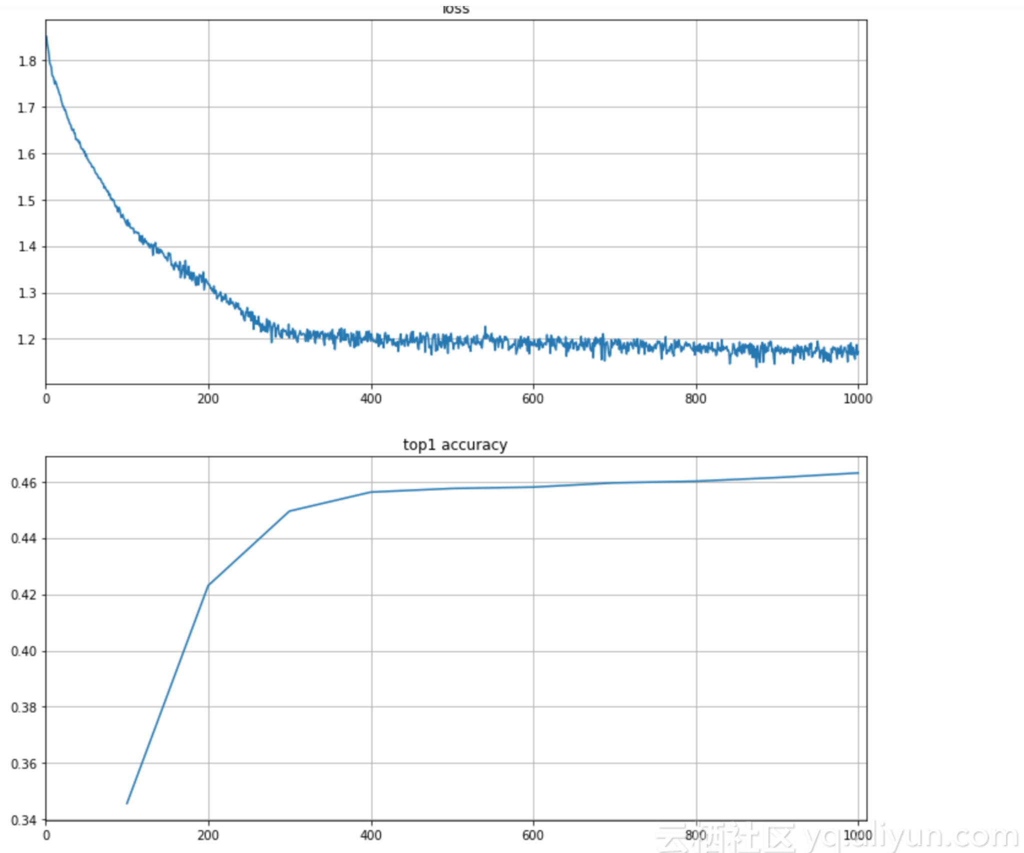
云栖社区 yq.aliyun.com

4. View training results.

```
In [16]: loss = np.array(train_summary.read_scalar("Loss"))
top1 = np.array(val_summary.read_scalar("Top1Accuracy"))

plt.figure(figsize = (12,12))
plt.subplot(2,1,1)
plt.plot(loss[:,0],loss[:,1],label='loss')
plt.xlim(0,loss.shape[0]+10)
plt.grid(True)
plt.title("loss")
plt.subplot(2,1,2)
plt.plot(top1[:,0],top1[:,1],label='top1')
plt.xlim(0,loss.shape[0]+10)
plt.title("top1 accuracy")
plt.grid(True)
```



云栖社区 yq.aliyun.com


loss
top1 accuracy

8 Adaptive execution of Spark SQL

Spark SQL of Alibaba Cloud Elastic MapReduce (E-MapReduce) 3.13.0 supports adaptive execution. It is used to set the number of reduce tasks automatically, solve data skew, and dynamically optimize execution plans.

Solved problems

Adaptive execution of Spark SQL solves the following problems:

- The number of shuffle partitions

  Currently, the number of tasks in the reduce stage in Spark SQL depends on the value of the **`spark.sql.shuffle.partition`** parameter (the default value is 200). Once this parameter has been specified for a job, the number of reduce tasks in all stages is the same value when the job is running.

  For different jobs, and for different reduce stages of one job, the actual data size can be quite different. For example, data to be processed in the reduce stage may have a size of 10 MB or 100 GB. If the parameter is specified using the same value, it has a significant impact on the actual processing efficiency. For example, 10 MB of data can be processed using only one task. If the value of the **`spark.sql.shuffle.partition`** parameter is set to the default value of 200, then 10 MB of data is partitioned to be processed by 200 tasks. This increases scheduling overheads and lowers processing efficiency.

  By setting the range of the shuffle partition number, the adaptive execution framework of Spark SQL can dynamically adjust the number of reduce tasks in the range for different stages of different jobs.

  This significantly reduces the costs for optimization (no need to find a fixed value). Additionally, the numbers of reduce tasks in different stages of one job can be dynamically adjusted.

  Parameter:

Attribute	Default value	Description
spark.sql.adaptive.enabled	false	Enables or disables adaptive execution.
spark.sql.adaptive.minNumPostShufflePartitions	1	The minimum number of reduce tasks.
spark.sql.adaptive.maxNumPostShufflePartitions	500	The maximum number of the reduce tasks.

Attribute	Default value	Description
spark.sql.adaptive.shuffle.targetPostShuffleInputSize	67108864	Dynamically adjusts the number of reduce tasks based on the partition size. For example, if the value is set to 64 MB, then each task in the reduce stage processes more than 64 MB data.
spark.sql.adaptive.shuffle.targetPostShuffleRowCount	20000000	Dynamically adjusts the number of reduce tasks based on the row number in the partition. For example, if the value is set to 20000000, then each task in the reduce stage processes more than 20,000,000 rows of data.

- Data skew

  Data skew is a common issue in SQL join operations. It refers to the scenario where certain tasks involve too much data in the processing, which leads to long tails. Currently, Spark SQL does not perform optimization for skewed data.

  The Adaptive Execution framework of Spark SQL can automatically detect skewed data and perform optimization for it at runtime.

  SparkSQL optimizes skewed data as follows: it splits the data that is in the skewed partition, processes the data through multiple tasks, and then combines the results through SQL union operations.

  Supported join types:

Type	Description
Inner	Skewed data can be handled in both tables.
Cross	Skewed data can be handled in both tables.
LeftSemi	Skewed data can only be handled in the left table.
LeftAnti	Skewed data can only be handled in the left table.
LeftOuter	Skewed data can only be handled in the left table.

Type	Description
RightOuter	Skewed data can only be handled in the right table.

Parameter:

Attribute	Default value	Description
spark.sql.adaptive.enabled	false	Enables or disables the adaptive execution framework.
spark.sql.adaptive.skewedJoin.enabled	false	Enables or disables the handling of skewed data.
spark.sql.adaptive.skewedPartitionFactor	10	A partition is identified as a skewed partition only when the following scenarios occur. First, the size of a partition is greater than this value (median size of all partitions) and the value of the spark.sql.adaptive.skewedPartitionSizeThreshold parameter. Second, the rows in a partition are greater than this value (median rows in all partitions) and the value of the spark.sql.adaptive.skewedPartitionSizeThreshold parameter.
spark.sql.adaptive.skewedPartitionSizeThreshold	67108864	The size threshold for a skewed partition.
spark.sql.adaptive.skewedPartitionRowCountThreshold	10000000	The row number threshold for a skewed partition.
spark.shuffle.statistics.verbose	false	When the value of this parameter is true, MapStatus collects information about the number of rows in each partition for handling skewed data.

- Execution plan optimization at runtime

  Catalyst optimizer of Spark SQL converts logical plans that are converted from SQL statements into physical execution plans and executes those physical execution plans. However, the physical execution plan produced by Catalyst may not be optimal because of lack or inaccuracy of statistics. For example, Spark SQL may choose SortMergeJoinExec instead of BroadcastJoin, while BroadcastJoin is the optimal option in the scenario.

  The Adaptive Execution framework of Spark SQL determines whether to use BroadcastJoin instead of SortMergeJoin to improve query performance based on the size of the shuffle write in the shuffle stage.

  Parameter:

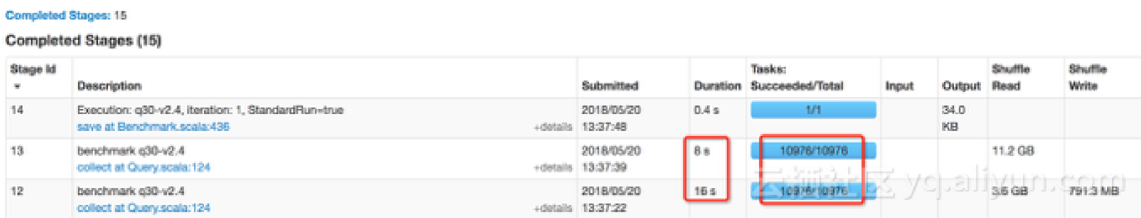
Attribute	Default value	Description
spark.sql.adaptive.enabled	false	Enables or disables the adaptive execution framework .
spark.sql.adaptive.join.enabled	true	Whether to determine a better join strategy at runtime.
spark.sql.adaptiveBroadcastJoinThreshold	Equals to spark.sql.autoBroadcastJoinThreshold.	Determines whether to use broadcast join to optimize join queries.

Test

Take some TPC-DS queries as test samples.

- Shuffle partition number
  - query30

    Native Spark:

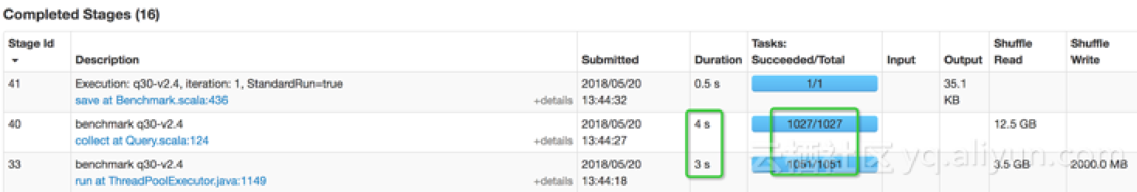


Completed Stages: 15

Completed Stages (15)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
14	Execution: q30-v2.4, iteration: 1, StandardRun=true save at Benchmark.scala:436 +details	2018/05/20 13:37:48	0.4 s	1/1		34.0 KB		
13	benchmark q30-v2.4 collect at Query.scala:124 +details	2018/05/20 13:37:39	8 s	10976/10976			11.2 GB	
12	benchmark q30-v2.4 collect at Query.scala:124 +details	2018/05/20 13:37:22	16 s	10976/10976			3.6 GB	791.3 MB

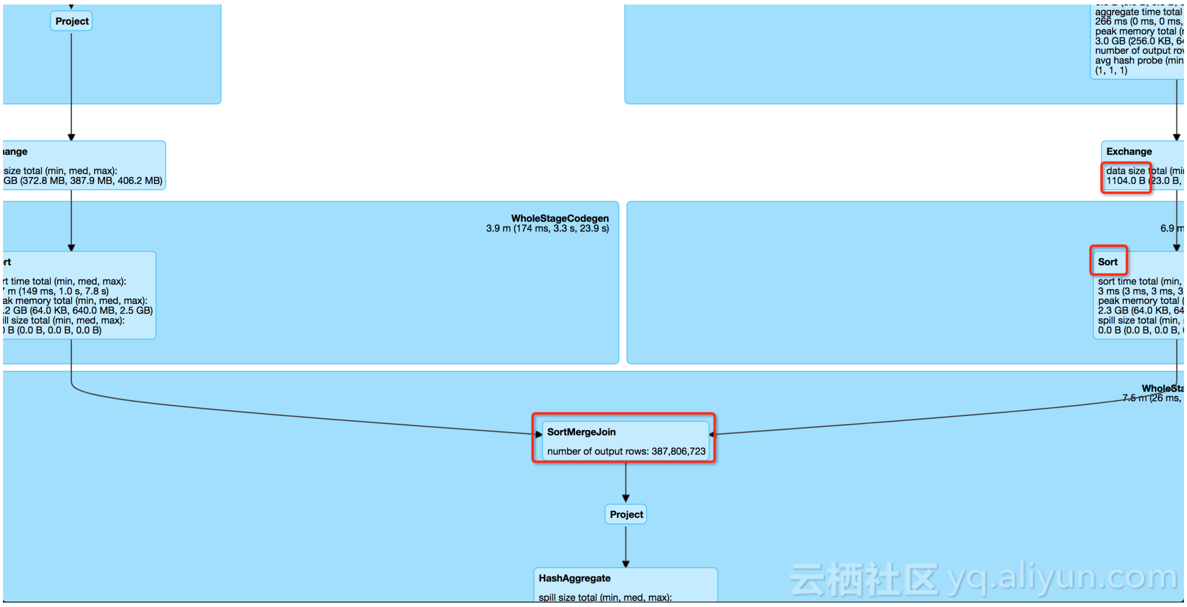
  - Adjusts the number of reduce tasks adaptively.



Completed Stages (16)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
41	Execution: q30-v2.4, iteration: 1, StandardRun=true save at Benchmark.scala:436 +details	2018/05/20 13:44:32	0.5 s	1/1		35.1 KB		
40	benchmark q30-v2.4 collect at Query.scala:124 +details	2018/05/20 13:44:27	4 s	1027/1027			12.5 GB	
33	benchmark q30-v2.4 run at ThreadPoolExecutor.java:1149 +details	2018/05/20 13:44:18	3 s	1051/1051			3.5 GB	2000.0 MB

- Execution plan optimization at runtime (SortMergeJoin to BroadcastJoin).



Uses BroadcastJoin adaptively.