ALIBABA CLOUD

阿里云

Databricks 数据洞察

最佳实践

文档版本：20201019

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.使用Databricks Delta优化Spark作业

本文介绍如何使用Databricks Delta进行Spark作业的优化。

前提条件

已创建集群，详情请参见创建集群。

集群应满足以下配置：

区域	详情
地域（Region）	华北2（北京）
集群规模	1个Master节点，5个Worker节点
ECS实例配置	配置如下： • CPU：32核 • 内存：128GiB • ECS规格：ecs.g6.8xlarge • 数据盘配置：ESSD云盘300GB X 4块 • 系统盘配置：ESSD云盘120GB X 1块 说明 ECS实例会因库存等原因和实际售卖页有出入。此处参数仅供参考，具体请您根据实际情况选择相应的实例规格进行测试。
OSS宽带	10Gbps

背景信息

Databricks数据洞察内置了Databricks商业版引擎，您可以利用Databricks数据洞察创建集群，实现在秒级响应时间内处理PB级别的数据。本文示例制造100亿条数据，利用Databricks Delta的Data Skipping和ZOEDER Clustering特性，对Spark作业进行改造，达到优化性能的目的。Databricks Delta详情请参见Processing Petabytes of Data in Seconds with Databricks Delta。

配置Spark

1. 使用阿里云账号登录Databricks数据洞察控制台。
2. 在Databricks数据洞察控制台页面，选择所在的地域（Region）。 创建的集群将会在对应的地域内，一旦创建后不能修改。
3. 在左侧导航栏中，单击集群。
4. 单击待配置集群所在行的详情。
5. 在集群详情页面，单击上方的Spark配置。
6. 配置以下参数。
   - 修改以下配置。

参数	描述
spark.driver.cores	4
spark.driver.memory	8G
spark.executor.memory	23G

- 新增以下配置。
  a. 在配置区域，单击spark-defaults页签。
  b. 单击右侧的自定义配置。

  设置以下配置。

参数	描述
spark.executor.cores	3
spark.executor.instances	22
spark.yarn.executor.memoryOverhead	default

示例

1. 准备数据。
   - 准备测试数据和query脚本。

     在集群中生成数据预计需要5小时，生成测试数据详情请参见Processing Petabytes of Data in Seconds with Databricks Delta。
   - 准备五张表：
     - conn_random：delta格式表
     - conn_random_parquet：parquet格式表
     - conn_optimize：经过OPTIMIZE的表，主要是Compaction
     - conn_zorder_only_ip：ZORDER BY (src_ip, dst_ip)
     - conn_zorder：ZORDER BY (src_ip, src_port, dst_ip, dst_port)
2. 使用OPTIMIZE命令进行优化。 详细代码如下：

```
import spark.implicits._

val seed = 0
val numRecords = 10*1000*1000*1000L
val numFiles = 1000*1000

val baseLocation = "oss://mytest/records-10m(1000)3-(1000)2/data/random/"

val dbName = s"mdc_random_$numFiles"
val connRandom = "conn_random"
```

```
val connRandomParquet = "conn_random_parquet"
// val connSorted = "conn_sorted"
val connOptimize = "conn_optimize"
val connZorderOnlyIp = "conn_zorder_only_ip"
val connZorder = "conn_zorder"

spark.conf.set("spark.sql.shuffle.partitions", numFiles)
spark.conf.get("spark.sql.shuffle.partitions")

sql(s"drop database if exists $dbName cascade")
sql(s"create database if not exists $dbName")
sql(s"use $dbName")
sql(s"show tables").show(false)

import scala.util.Random

case class ConnRecord(src_ip: String, src_port: Int, dst_ip: String, dst_port: Int)

// 生成数据
def randomIPv4(r: Random) = Seq.fill(4)(r.nextInt(256)).mkString(".")
def randomPort(r: Random) = r.nextInt(65536)

def randomConnRecord(r: Random) = ConnRecord(
  src_ip = randomIPv4(r), src_port = randomPort(r),
  dst_ip = randomIPv4(r), dst_port = randomPort(r))

val df = spark.range(0, numFiles, 1, numFiles).mapPartitions { it =>
  val partitionID = it.toStream.head
  val r = new Random(seed = partitionID)
  Iterator.fill((numRecords / numFiles).toInt)(randomConnRecord(r))
}

// 生成数据表
df.write
.mode("overwrite")
.format("delta")
.option("path", baseLocation + connRandom)
.saveAsTable(connRandom)

df.write
```

```
.mode("overwrite")
.format("parquet")
.option("path", baseLocation + connRandomParquet)
.saveAsTable(connRandomParquet)

spark.read.table(connRandom)
.write
.mode("overwrite")
.format("delta")
.option("path", baseLocation + connOptimize)
.saveAsTable(connOptimize)

spark.read.table(connRandom)
.write
.mode("overwrite")
.format("delta")
.option("path", baseLocation + connZorderOnlyIp)
.saveAsTable(connZorderOnlyIp)

spark.read.table(connRandom)
.write
.mode("overwrite")
.format("delta")
.option("path", baseLocation + connZorder)
.saveAsTable(connZorder)

spark.conf.set("spark.databricks.io.skipping.mdc.addNoise", "false")

// OPTIMIZE优化命令
sql(s"OPTIMIZE '${baseLocation + connOptimize}'")
sql(s"OPTIMIZE '${baseLocation + connZorderOnlyIp}' ZORDER BY (src_ip, dst_ip)")
sql(s"OPTIMIZE '${baseLocation + connZorder}' ZORDER BY (src_ip, src_port, dst_ip, dst_port)")
```

3. 验证Spark SQL。

```
select count(*) from conn_random where src_ip like '157%' and dst_ip like '216.%';

select count(*) from conn_random_parquet where src_ip like '157%' and dst_ip like '216.%';

select count(*) from conn_optimize where src_ip like '157%' and dst_ip like '216.%';

select count(*) from conn_zorder_only_ip where src_ip like '157%' and dst_ip like '216.%';

select count(*) from conn_zorder where src_ip like '157%' and dst_ip like '216.%';
```

测试结论

本示例各表情况如下。

表名称	时间（s）
conn_random_parquet	2504
conn_random	2324
conn_optimize	112
conn_zorder	65
conn_zorder_only_ip	46

说明 通过以上示例，可以发现：

- 经过OPTIMIZE的表，文件大小会在1G左右，而且进行了delta元数据的优化，提高了data-skipping的效率，在性能上提升约20倍（2504/112=22X）。
- Zorder使得data-skipping的优化效果进一步深化，性能提升约40倍（2504/65=38X）。
- 当Zorder列是查询列时，优化效果会更加明显，实验显示性能提升约50倍（2504/46=54X）。

问题反馈

您在使用阿里云Databricks数据洞察过程中有任何疑问，欢迎用钉钉扫描下面的二维码加入钉钉群进行反馈。

Databricks数据洞察产品群

2.Databricks Delta vs Open-Source Delta Lake

本文介绍Databricks数据洞察产品中Databricks Runtime Delta和社区开源版本Delta Lake在性能优化方面的差异点。

Performance Optimization

1. Compaction

Delta Lake on Databricks can improve the speed of read queries from a table by coalescing small files into larger ones.

Official Document: https://docs.databricks.com/delta/optimizations/file-mgmt.html#compaction-bin-packing

Syntax

```
OPTIMIZE delta.`/data/events`

Or

OPTIMIZE events

Specify an optional partition predicate

OPTIMIZE events WHERE date >= '2017-01-01'
```

```
# Databricks notebook source

# DBTITLE 1,Clean up Parquet tables
#  %fs rm -r /tmp/flights_parquet

# DBTITLE 1,Clean up Databricks Delta tables
#  %fs rm -r /tmp/flights_delta
# DBTITLE 1,Step 0: Read flights data

flights = spark.read.format("csv") \
  .option("header", "true") \
  .option("inferSchema", "true") \
  .load("/databricks-datasets/asa/airlines/2008.csv")



# DBTITLE 1,Step 1: Write a Parquet based table using flights data
```

```
flights.write.format("parquet").mode("overwrite").partitionBy("Origin").save("/tmp/flights_parquet")

#  Once step 1 completes, the "flights" table contains details of US flights for a year.
#  Next in Step 2, we run a query that get top 20 cities with the highest monthly total flights on the firs
t day of week.

# DBTITLE 1,Step 2: Run a query
from pyspark.sql.functions import count

flights_parquet = spark.read.format("parquet").load("/tmp/flights_parquet")

display(flights_parquet.filter("DayOfWeek = 1").groupBy("Month","Origin").agg(count("*").alias("TotalF
lights")).orderBy("TotalFlights", ascending=False).limit(20))

#  Once step 2 completes, you can observe the latency with the standard "flights_parquet" table.
#  In step 3 and step 4, we do the same with a Databricks Delta table. This time, before running the qu
ery, we run the `OPTIMIZE` command to ensure data is optimized for faster retrieval.

# DBTITLE 1,Step 3: Write a Databricks Delta based table using flights data
flights.write.format("delta").mode("overwrite").partitionBy("Origin").save("/tmp/flights_delta")

# DBTITLE 1,Step 3 Continued: OPTIMIZE the Databricks Delta table
display(spark.sql("DROP TABLE  IF EXISTS flights"))

display(spark.sql("CREATE TABLE flights USING DELTA LOCATION '/tmp/flights_delta'"))

display(spark.sql("OPTIMIZE flights"))

# DBTITLE 1,Step 4 : Rerun the query from Step 2 and observe the latency
flights_delta = spark.read.format("delta").load("/tmp/flights_delta")

display(flights_delta.filter("DayOfWeek = 1").groupBy("Month","Origin").agg(count("*").alias("TotalFlig
hts")).orderBy("TotalFlights", ascending=False).limit(20))

# The query over the Databricks Delta table runs much faster after `OPTIMIZE` is run.
# How much faster the query runs can depend on the configuration of the cluster you are running on,
# however should be **15-20X faster**compared to the standard parquet table.
```

Benchmark

Test Environment	Query over Parquet Table	Query over Delta Table after `OPTIMIZE` is run
	50.95 seconds	1.93 seconds

2. Data Skipping

Data skipping information is collected automatically when you write data into a Delta table. Delta Lake on Databricks takes advantage of this information (minimum and maximum values) at query time to provide faster queries. You do not need to configure data skipping - the feature is activated whenever applicable.

Official Document: https://docs.databricks.com/delta/optimizations/file-mgmt.html#compaction-bin-packing

Blog and User Cases: https://databricks.com/blog/2018/07/31/processing-petabytes-of-data-in-seconds-with-databricks-delta.html?_ga=2.29295480.552083878.1584501563-968665100.1584501563

3. Z-Ordering

Z-Ordering is a technique

Syntax

```
OPTIMIZE events
WHERE date >= current_timestamp() - INTERVAL 1 day
ZORDER BY (eventType)
```

Demo

```
All the script for data generation and test code can be found
in: https://drive.google.com/drive/u/1/folders/1ZQKoku9zFoC4qwYY3jlKlhYlvF8I2SpI

-- Dataset schema:

CREATE TABLE `conn_zorder` (`src_ip` STRING, `src_port`INT, `dst_ip` STRING, `dst_port` INT)
USING delta
OPTIONS (
 `serialization.format`
 '1',
 path 'hdfs://rhodium-tests-1:8020/tmp/data/random/conn_zorder')

-- Optimize command:

OPTIMIZE '${baseLocation + connZorder}' ZORDER BY(src_ip, src_port, dst_ip, dst_port)

-- Query Tested:

select count(*) from conn_zorder where src_ip like '157%'and dst_ip like '216.%'
```

Benchmark

Product	Data Set Size	Runtime of Predicate Query on optimized Columns (s)	DBR Delta faster than OSS Parquet by factor of
DBR Delta	10B Row, 1M files (267GB)	11	16.9090909
OSS Parquet	10B Row, 1M files (267GB)	186	

3.Databricks数据洞察与Delta vs Open-Source Delta Lake功能对比

本文提供Databricks数据洞察中的Databricks Runtime Delta与社区开源版本Delta Lake。

Databricks Runtime vs Apache Spark

下表中的 feature 列表来自 Databricks 官网(https://databricks.com/spark/comparing-databricks-to-apache-spark

Feature	Apache Spark	Databricks数据洞察
Built-in file system optimized for cloud storage access (AWS S3, Redshift, Azure Blob	No	Yes
Spark-native fine grained resource sharing for optimum utilization	No	Yes
Fault isolation of compute resources	No	Yes
Faster writes to OSS	No	Yes
Compute optimization during joins and filters	No	Yes
Rapid release cycles	No	Yes
Auto-scaling compute	No	即将发布
High availability for cluster	No	即将发布

Databricks Delta vs Open-source Delta Lake

Feature	Open SourceDelta Lake	Databricks Delta
Snapshot Isolation / Transactional Guarantees	Yes	Yes
Efficient directory / File listing	Yes	Yes
Version history and time travel	Yes	Yes
Schema evolution & enforcement	Yes	Yes
Hidden partitions / Partitioning by expressions	In Roadmap	In Roadmap
HDFS Support	Yes	Yes

Feature	Open SourceDelta Lake	Databricks Delta
Object Storage Support	Yes	Yes
Streaming Data Sink	Yes	Yes
Streaming Data Source	Yes	Yes
Basic Upsert (merge into)	Yes	Yes
Scalable Upsert (merge into)	No	Yes
Data skipping based on stats	No	Yes
Compact small files	Yes	Yes
Optimize (efficiently compact small files)	No	Yes
Auto Optimize	No	Yes
Native Parquet Reader	No	Yes
Local SSD Caching	No	No
Read from Presto	Yes	Yes
Read from Hive	In Roadmap	In Roadmap

4.Databricks 数据洞察访问E-MapReduce数据源

本文介绍如何使用阿里云 Databricks 数据洞察创建的集群去访问外部数据源 E-MapReduce，并运行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。
- 已开通对象存储 OSS服务。
- 已开通 Databricks数据洞察服务。
- 已完成云账号的授权，详情请参见角色授权。

步骤一：创建Kafka集群和Databricks 数据洞察集群

1. 登录阿里云E-MapReduce控制台。
2. 创建Kafka集群。
3. 登陆Databricks数据洞察控制台。
4. 创建集群，详情参见创建集群。

步骤二：Databricks 数据洞察集群添加外部数据源

1. 登陆Databricks数据洞察控制台。
2. 单击左侧**集群**按钮，选择已创建的集群。
3. 进入集群详情页面，单击上方**数据源**按钮。
4. 在数据源页面，单击**添加**按钮，选择Aliyun EMR KAFKA
5. 填入**描述**，选择**kafka集群**。

步骤三：获取JAR包并上传到对象存储 OSS

1. 登陆OSS管理控制台。
2. 创建Bucket存储空间，详情请参见存储空间。
3. 获取JAR包（spark-kafka-sample-1.0-SNAPSHOT-jar-with-dependencies.jar）。
4. 上传JAR，详情请参见上传文件。

步骤四：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。

步骤五：运行Spark Streaming作业

1. 登陆Databricks数据洞察控制台。
2. 新建项目空间，详情请参见新建项目。
3. 在所属项目空间创建作业，详情请参见管理作业。
4. 执行如下作业命令，进行流式单词统计。

```
--class com.aliyun.scala.ScalaKafkaStream oss://xxx/xxx/spark-kafka-sample-1.0-SNAPSHOT-jar-with-dependencies.jar 192.168.xxx.xxx:9092 test
```

关键参数说明如下：

参数	说明
oss://xxx/xxx/spark-kafka-sample-1.0-SNAPSHOT-jar-with-dependencies.jar	oss对象存储上JAR包的位置
192.168.xxx.xxx:9092	Kafka集群中任一Kafka Broker组件的内网IP地址。IP地址如下图所示。
test	Topic名称

Kafka集群IP

步骤六：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
```

在命令行中输入数据

步骤七：查看结果

通过Yarn UI查看Spark Streaming作业的信息，详情请参见集群 Web UI。

1. 在Hadoop控制台，单击作业的ID。

详细信息如下。

点击Logs，查看详细信息。

5.Databricks 数据洞察pyspark实例

本文介绍如何使用阿里云 Databricks 数据洞察的Notebook进行pyspark开发。

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。
- 已开通对象存储 OSS服务。
- 已开通 Databricks数据洞察服务。
- 已完成云账号的授权，详情请参见角色授权。

> 注意
>
> 若要使用其他数据源进行数据开发，需开通相应服务。本示例采用OSS数据源。

步骤一：创建Databricks 数据洞察集群

- 登录阿里云E-MapReduce控制台。
- 创建Databricks 数据洞察集群，详情参见集群创建。

步骤二：添加依赖库并安装

根据开发需要，添加相应的依赖库

- 添加matplotlib库，本示例使用PyPI方式添加，详情参见管理库。
- 单击安装按钮，安装依赖到开发集群。
- 等待安装完成，可单击任务按钮查看。

步骤三：获取数据并上传对象存储 OSS

- 登陆OSS管理控制台。
- 创建Bucket存储空间，详情请参见存储空间。
- 获取测试文件（TEST.txt）。
- 上传文件，详情请参见上传文件。

步骤四：数据开发

- Notebook使用，详情参见Notebook使用。
- 数据开发，将以下代码写入note文件，如下图所示。

```
%pyspark
import matplotlib.pyplot as plt

data = sc.textFile('oss://xxx/xxx/TEST.txt')

data = data.flatMap(lambda line: line.split(" "))

resultRdd = data.map(lambda word: (word,1)).reduceByKey(lambda a,b:a+b)

result = resultRdd.sortBy(lambda x: x[1], False)

resultColl = resultRdd.collect()

x = result.keys().collect()
y = result.values().collect()
plt.bar(x, y, color="b", label="count")
plt.minorticks_on()
plt.title("WordCount")
plt.xlabel("word")
plt.ylabel("count")
plt.legend()
z.show(plt)
```

- 单击右上角运行按钮，等待任务结束查看结果

6.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。

操作步骤

1. 云账号登录RAM控制台。
2. 创建RAM用户。
3. 新建权限策略。
   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. 选择**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）

测试环境（test-bucket）	生产环境（prod-bucket）

```
{
"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/*"
]
}
]
}
```

```
{
"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/*"
]
}
]
}
```

测试环境（例如中国东部）	生产环境（例如中国北京）

按上述脚本示例进行权限隔离后，RAM用户在Databricks 数据洞察控制台的权限如下：

a. 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。

b. 只能看到被授权的bucket下的内容，无法看到其他bucket内的内容。

c. 作业中只能读写被授权的bucket，读写未被授权的bucket会报错。

4. 为RAM授权

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**。完成授权后，权限立即生效。