ALIBABA CLOUD

Alibaba Cloud

Tablestore

Compute-Analysis

Document Version: 20201023

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 because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should 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 this document based on the "status quo", "being defective", and "existing functions" of its products and services. 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 take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, 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 contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, 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 this document 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 directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice 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.
Warning	A warning notice 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 restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.MaxCompute

1.1. Allow MaxCompute to access Table Store across accounts

This article describes how Table Store and MaxCompute on different accounts be seamlessly connected. For information about enabling communication between Table Store and MaxCompute within the same account, see Allow MaxCompute to access Table Store using one account.

Preparation

Grant the account owner of MaxCompute (account B) access permissions to table data owned by the Table Store account owner (account A). An example scenario is as follows:

> **Note** The following information is for reference only.

Item	Table Store	MaxCompute
Primary Account Name	Account A	Account B
UserId	12345	56789

Preparation for enabling MaxCompute to access Table Store owned by a different account:

1. Log on as account B, activate the MaxCompute service and then create a MaxCompute project.
2. Create an AccessKey for Account A and for Account B.
3. Use Account A to log on to the RAM console, and create a user role on the **Roles** page.

   In this example, the created role is named AliyunODPSRoleForOtherUser.
4. Click the role name to enter **Role Details** page. Click **Edit basic information** to set the policy content. The policy content is specified as follows.

```
{
 "Statement": [
  {
   "Action": "sts:AssumeRole",
   "Effect": "Allow",
   "Principal": {
    "Service": [
     "1xxxx@odps.aliyuncs.com"
    ]
   }
  }
 ],
 "Version": "1"
}
```

> **Note** Replace 1xxxx with your UID in the preceding policy content.

5. On the **Role Details** page, view the role Arn.
6. Create an authorization policy in **Policies** page.

   In this example, the authorization policy is named AliyunODPSRolePolicyForOtherUser.

   The policy content is specified as follows.

```
  {
"Version": "1",
"Statement": [
    {
      "Action": [
        "ots:ListTable",
        "ots:DescribeTable",
        "ots:GetRow",
        "ots:PutRow",
        "ots:UpdateRow",
        "ots:DeleteRow",
        "ots:GetRange",
        "ots:BatchGetRow",
        "ots:BatchWriteRow",
        "ots:ComputeSplitPointsBySize"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
]
  }
```

> **Note** You can also specify other permissions, such as CreateTable.

7. Grant the permission AliyunODPSRolePolicyForOtherUser to the role AliyunODPSRoleForOtherUser in **Roles** page.

8. Create an instance and create a table in the Table Store console.

   In this example, the Table Store instance and the table are created as follows:

   - Instance name: cap1
   - Data table name: vehicle_track
   - Primary key information: vid (integer), gt (integer)
   - Endpoint: `https://cap1.cn-hangzhou.ots-internal.aliyuncs.com`

     > **Note** We recommend that you use the Table Store intranet address when accessing Table Store using MaxCompute.

   - Set the network type of the instance to **Any Network**.

Allow MaxCompute to Access Table Store across accounts

Repeat the steps described in Allow MaxCompute to access Table Store under the same account. You must specify a roleArn when creating an external table for cross-account access.

Log on as account B and create an external table in MaxCompute. When creating the external table, specify the role Arn that you created in the preceding Preparation step.

For detailed steps, see Allow MaxCompute to access Table Store using one account. Use the following code when creating the external table in Step 2:

```
CREATE EXTERNAL TABLE ads_log_ots_pt_external
(
vid bigint,
gt bigint,
longitude double,
latitude double,
distance double ,
speed double,
oil_consumption double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler'
WITH SERDEPROPERTIES (
'tablestore.columns.mapping'=':vid, :gt, longitude, latitude, distance, speed, oil_consumption',
'tablestore.table.name'='vehicle_track',
'odps.properties.rolearn'='acs:ram::12345:role/aliyunodpsroleforotheruser'
)
LOCATION 'tablestore://cap1.cn-hangzhou.ots-internal.aliyuncs.com'
USING 'odps-udf-example.jar'
```

1.2. Allow MaxCompute to access Table Store using AccessKeys

In addition to account authorization, you can access data in Table Store using AccessKeys on MaxCompute.

Preparation

Get the AccessKey credentials (that is, AccessKeyId and AccessKeySecret) for the account that owns the target Table Store resources. If the AccessKey is for a Resource Access Management (RAM) user, the RAM user must be granted with at least the following permissions to operate the Table Store resources:

```
{
 "Version": "1",
 "Statement": [
  {
   "Action": [
    "ots:ListTable",
    "ots:DescribeTable",
    "ots:GetRow",
    "ots:PutRow",
    "ots:UpdateRow",
    "ots:DeleteRow",
    "ots:GetRange",
    "ots:BatchGetRow",
    "ots:BatchWriteRow",
    "ots:ComputeSplitPointsBySize"
   ],
   "Resource": "*",
   "Effect": "Allow"
  }
 ]
}

-- You can also define other permissions
```

Allow MaxCompute to access Table Store using AccessKeys

Unlike account authorization, you must specify the AccessKey information in the `LOCATION` clause when creating an external table.

```
LOCATION 'tablestore://${AccessKeyId}:${AccessKeySecret}@${InstanceName}. ${Region}.ots-internal.aliyuncs.com'
```

Assume that the following information is what MaxCompute must access:

AccessKeyId	AccessKeySecret	Instance name	Region	Network mode
abcd	1234	cap1	cn-hangzhou	Intranet access

The statement to create the external table is:

```
CREATE EXTERNAL TABLE ads_log_ots_pt_external
(
vid bigint,
gt bigint,
longitude double,
latitude double,
distance double ,
speed double,
oil_consumption double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler'
WITH SERDEPROPERTIES (
'tablestore.columns.mapping'=':vid, :gt, longitude, latitude, distance, speed, oil_consumption',
'tablestore.table.name'='vehicle_track'
)
LOCATION 'tablestore://abcd:1234@cap1.cn-hangzhou.ots-internal.aliyuncs.com'
```

For more information about accessing data, see the following section in Allow MaxCompute to access Table Store using one account: Step 3. Access Table Store data through external tables

1.3. Process data by using UDF

If your data stored in Table Store is uniquely structured and you want to define development logic to process each line of data (for example, parsing a specific JSON string), you can use User Defined Function (UDF).

Procedure

1. Follow the MaxCompute Studio to install MaxCompute-Java/MaxCompute-Studio plug-in in IntelliJ. Development may be started when the plug-in is installed.

   The following figure shows a simple UDF definition, which connects two strings. MaxCompute supports more complex UDF, for example, user-defined window execution logic. For more information, see Develop and debug UDF.

2. Upload the resource to MaxCompute after packaging.

   Select **File > Project Structure > Artifacts**. Enter the *Name* and the *Output directory*, then click + to select the output module. After packaging, upload the resource and create a function using ODPS Project Explorer, and then you can call it in SQL.

3. Run **bin/odpscmd.bat**.

```
// Select a line of data, and pass name/name into the UDF. A connection of the two strings is retu
rned.
select cloud_metric_extract_md5(name, name) as udf_test from test_table limit 1;
```

The returned result is displayed as follows.

1.4. Common errors troubleshooting

- Symptoms: FAILED: ODPS-0010000:System internal error - fuxi job failed, WorkerPackageNotExist

  Resolution: Set set odps.task.major.version=unstructured_data.

- Symptoms: FAILED: ODPS-0010000:System internal error - std::exception:Message: a timeout was reached

  Resolution: The Table Store endpoint is incorrect, which makes MaxCompute unaccessible. Enter the correct endpoint.

- Symptoms: logview invalid end_point

  Resolution: The logview URL is returned during the execution. If an error is returned when a browser is used to access the address, it may be due to an incorrect configuration. Check MaxCompute configuration.

If the problem persists, contact the after-sales technical support.

2.Spark/SparkSQL

2.1. Environment preparations

Access Table Store tables with Spark or Spark SQL

You can use Spark and Spark SQL to access data in Table Store directly by using the dependency package released by Table Store and E-MapReduce.

Install Spark/Spark SQL

1. Download the Spark installation package that complies with the following requirements:
   - Release version: 1.6.2
   - Package type: Pre-built for Hadoop 2.6
   - Download type: Direct Download
2. Unpack the installation package as follows.

```
$ cd /home/admin/spark-1.6.2
$ tar -zxvf spark-1.6.2-bin-hadoop2.6.tgz
```

Install JDK-7+

1. Download and install the installation package of JDK-7+.
   - Linux/MacOS: Use the package installation manager.
   - Windows: Click to download.
2. Check the installation status as follows.

```
$ java -version
java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
Java HotSpot(TM) 64-Bit Server VM (build 25.77-b03, mixed mode)
```

Download Java SDK for Table Store

1. Download the Java SDK dependency package (version 4.1.0 or later).

> Note The SDK dependency package is updated with Java SDK. Download the dependency package according to the latest Java SDK.

2. Copy the SDK to the Spark directory as follows.

```
$ mv tablestore-4.1.0-jar-with-dependencies.jar /home/admin/spark-1.6.2/
```

Download EMR dependency package

- Download the Alibaba Cloud EMR dependency package.

  > Note For more information on EMR, click here.

- Rename the *emr-sdk_2.10-1.3.0-20161025.065936-1.jar* file.

```
mv emr-sdk_2.10-1.3.0-20161025.065936-1.jar /home/admin/spark-1.6.2/emr-sdk_2.10-1.3.0-SNAPSHOT.jar
```

Run Spark SQL

```
$ cd /home/admin/spark-1.6.2/
$ bin/spark-sql --master local --jars tablestore-4.3.1-jar-with-dependencies.jar,emr-tablestore-1.4.2.jar
```

2.2. Tutorial

Data preparation

Name a table in Table Store as pet and import the data. Note that column `name` is the only Primary Key.

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	
Puffball	Diane	hamster	f	1999-03-30	

> Note As Table Store is schema-free , you do not need to input anything (such as `NULL`) into blank cells.

Example for accessing by Spark SQL

Preparations

Prepare the environment for Spark, JDK, and dependency package of Table Store SDK and EMR as Prerequisites.

Examples

```
$ bin/spark-sql --master local --jars tablestore-4.3.1-jar-with-dependencies.jar,emr-tablestore-1.4.2.ja
r
spark-sql> CREATE EXTERNAL TABLE pet
 (name STRING, owner STRING, species STRING, sex STRING, birth STRING, death STRING)
 STORED BY 'com.aliyun.openservices.tablestore.hive.TableStoreStorageHandler'
 WITH SERDEPROPERTIES(
  "tablestore.columns.mapping"="name,owner,species,sex,birth,death")
 TBLPROPERTIES (
  "tablestore.endpoint"="YourEndpoint",
  "tablestore.access_key_id"="YourAccessKeyId",
  "tablestore.access_key_secret"="YourAccessKeySecret",
  "tablestore.table.name"="pet");
spark-sql> SELECT * FROM pet;
Bowser  Diane   dog    m     1979-08-31     1995-07-29
Buffy   Harold  dog    f     1989-05-13     NULL
Chirpy  Gwen    bird   f     1998-09-11     NULL
Claws   Gwen    cat    m     1994-03-17     NULL
Fang    Benny   dog    m     1990-08-27     NULL
Fluffy  Harold  cat    f     1993-02-04     NULL
Puffball      Diane   hamster f     1999-03-30     NULL
Slim    Benny   snake  m     1996-04-29     NULL
Whistler      Gwen    bird   NULL   1997-12-09     NULL
Time taken: 5.045 seconds, Fetched 9 row(s)
spark-sql> SELECT * FROM pet WHERE birth > "1995-01-01";
Chirpy  Gwen    bird   f     1998-09-11     NULL
Puffball      Diane   hamster f     1999-03-30     NULL
Slim    Benny   snake  m     1996-04-29     NULL
Whistler      Gwen    bird   NULL   1997-12-09     NULL
Time taken: 1.41 seconds, Fetched 4 row(s)
```

Parameters explanation

- WITH SERDEPROPERTIES

- tablestore.columns.mapping (optional): By default, the field names of the external tables (written in lower case according to Hive conventions) are the same as the column names (names of Primary Key or Attribute columns) in Table Store. However, due to case-sensitivity or charsets, the names may be different. In this case, tablestore.columns.mapping needs to be specified. This parameter is a comma-separated string. A blank space cannot be added before or after a comma. Each item is a column name and the order is the same as the field names of the external tables.

  > Note Table Store supports column names with blank characters. So a blank space is considered part of the column name.

- TBLPROPERTIES
  - tablestore.endpoint (required): The endpoint. You can view the endpoint information of the instance on the Table Store console.
  - tablestore.instance (optional): The instance name. If it is not specified, it is the first field of tablestore.endpoint.
  - tablestore.table.name (required): The table name in Table Store.
  - tablestore.access_key_id and tablestore.access_key_secret (required): See Access control.
  - tablestore.sts_token (optional): See Security token.

Example for accessing by Spark

The following example illustrates how to count rows in `pet` by Spark.

```
private static RangeRowQueryCriteria fetchCriteria() {
    RangeRowQueryCriteria res = new RangeRowQueryCriteria("YourTableName");
    res.setMaxVersions(1);
    List<PrimaryKeyColumn> lower = new ArrayList<PrimaryKeyColumn>();
    List<PrimaryKeyColumn> upper = new ArrayList<PrimaryKeyColumn>();
    lower.add(new PrimaryKeyColumn("YourPkeyName", PrimaryKeyValue.INF_MIN));
    upper.add(new PrimaryKeyColumn("YourPkeyName", PrimaryKeyValue.INF_MAX));
    res.setInclusiveStartPrimaryKey(new PrimaryKey(lower));
    res.setExclusiveEndPrimaryKey(new PrimaryKey(upper));
    return res;
}

public static void main(String[] args) {
    SparkConf sparkConf = new SparkConf().setAppName("RowCounter");
    JavaSparkContext sc = new JavaSparkContext(sparkConf);

    Configuration hadoopConf = new Configuration();
    TableStoreInputFormat.setCredential(
        hadoopConf,
        new Credential("YourAccessKeyId", "YourAccessKeySecret"));
    TableStoreInputFormat.setEndpoint(
        hadoopConf,
        new Endpoint("https://YourInstance.Region.ots.aliyuncs.com/"));
    TableStoreInputFormat.addCriteria(hadoopConf, fetchCriteria());

    try {
        JavaPairRDD<PrimaryKeyWritable, RowWritable> rdd = sc.newAPIHadoopRDD(
            hadoopConf,
            TableStoreInputFormat.class,
            PrimaryKeyWritable.class,
            RowWritable.class);
        System.out.println(
            new Formatter().format("TOTAL: %d", rdd.count()).toString());
    } finally {
        sc.close();
    }
}
```

Note If you use scala, replace JavaSparkContext with SparkContext, and replace JavaPairRDD with PairRDD.

Run the program

```
$ bin/spark-submit --master local --jars hadoop-connector.jar row-counter.jar
TOTAL: 9
```

Data type conversion

Table Store and Hive/Spark support the different sets of data types.

The following table indicates data type conversion support from Table Store (rows) to Hive (columns).

	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	BOOLEAN	STRING	BINARY
INTEGER	Yes, with limited precision	Yes, with limited precision	Yes, with limited precision	Yes	Yes, with limited precision	Yes, with limited precision			
DOUBLE	Yes, with limited precision	Yes, with limited precision	Yes, with limited precision	Yes, with limited precision	Yes, with limited precision	Yes			
BOOLEAN							Yes		
STRING								Yes	
BINARY									Yes

3.Hive/HadoopMR

3.1. Tutorial

Data preparation

Name a table in Table Store as pet and import the following data. Note that column `Name` is the only Primary Key.

Name	Owner	Species	Sex	Birth	Death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	
Puffball	Diane	hamster	f	1999-03-30	

> **Note** As Table Store is schemafree (according to the Overview topic), you do not need to input anything (such as `NULL`) into blank cells.

Example for accessing by Hive

Preparations

Prepare the environment for Hadoop, Hive, JDK, and dependency package of Table Store SDK and EMR as Prerequisites.

Example

```
# You can add HADOOP_HOME and HADOOP_CLASSPATH into /etc/profile
$ export HADOOP_HOME=${Your Hadoop Path}
$ export HADOOP_CLASSPATH=emr-tablestore-1.4.2.jar:tablestore-4.3.1-jar-with-dependencies.jar:joda-time-2.9.4.jar
$ bin/hive
hive> CREATE EXTERNAL TABLE pet
 (name STRING, owner STRING, species STRING, sex STRING, birth STRING, death STRING)
 STORED BY 'com.aliyun.openservices.tablestore.hive.TableStoreStorageHandler'
 WITH SERDEPROPERTIES(
  "tablestore.columns.mapping"="name,owner,species,sex,birth,death")
 TBLPROPERTIES (
  "tablestore.endpoint"="YourEndpoint",
  "tablestore.access_key_id"="YourAccessKeyId",
  "tablestore.access_key_secret"="YourAccessKeySecret",
  "tablestore.table.name"="pet");
hive> SELECT * FROM pet;
Bowser  Diane   dog    m     1979-08-31     1995-07-29
Buffy   Harold  dog    f     1989-05-13     NULL
Chirpy  Gwen    bird   f     1998-09-11     NULL
Claws   Gwen    cat    m     1994-03-17     NULL
Fang    Benny   dog    m     1990-08-27     NULL
Fluffy  Harold  cat    f     1993-02-04     NULL
Puffball      Diane   hamster f     1999-03-30     NULL
Slim    Benny   snake  m     1996-04-29     NULL
Whistler      Gwen    bird   NULL  1997-12-09     NULL
Time taken: 5.045 seconds, Fetched 9 row(s)
hive> SELECT * FROM pet WHERE birth > "1995-01-01";
Chirpy  Gwen    bird   f     1998-09-11     NULL
Puffball      Diane   hamster f     1999-03-30     NULL
Slim    Benny   snake  m     1996-04-29     NULL
Whistler      Gwen    bird   NULL  1997-12-09     NULL
Time taken: 1.41 seconds, Fetched 4 row(s)
```

Parameters explanation

- WITH SERDEPROPERTIES

tablestore.columns.mapping (optional): By default, the field names of the external tables (written in lower case according to Hive conventions) are the same as the column names (names of Primary Key or Attribute columns) in Table Store. However, due to case-sensitivity or charsets, the names may be different. In this case, tablestore.columns.mapping needs to be specified. This parameter is a comma-separated string. A blank space cannot be added before or after a comma. Each item is a column name and the order is the same as the field names of the external tables.

> **Note** Table Store supports column names with blank characters, which means a blank space is considered part of the column name.

- TBLPROPERTIES
  - tablestore.endpoint (required): The endpoint. You can view the endpoint information of the instance on the Table Store console.
  - tablestore.instance (optional): The instance name. If it is not specified, it is the first field of tablestore.endpoint.
  - tablestore.table.name (required): The table name in Table Store.
  - tablestore.access_key_id, tablestore.access_key_secret (required): See Access control.
  - tablestore.sts_token (optional): See Security Token.

Example for accessing by HadoopMR

The following example illustrates how to count rows in pet using HadoopMR.

Code examples

- Construct Mappers and Reducers.

```
public class RowCounter {
public static class RowCounterMapper
extends Mapper<PrimaryKeyWritable, RowWritable, Text, LongWritable> {
   private final static Text agg = new Text("TOTAL");
   private final static LongWritable one = new LongWritable(1);

   @Override
   public void map(
      PrimaryKeyWritable key, RowWritable value, Context context)
      throws IOException, InterruptedException {
      context.write(agg, one);
   }
}

public static class IntSumReducer
extends Reducer<Text,LongWritable,Text,LongWritable> {

   @Override
   public void reduce(
      Text key, Iterable<LongWritable> values, Context context)
      throws IOException, InterruptedException {
      long sum = 0;
      for (LongWritable val : values) {
         sum += val.get();
      }
      context.write(key, new LongWritable(sum));
   }
}
}
```

Each time HadoopMR fetches a row from pet, it calls map() of the mapper. The first two parameters, PrimaryKeyWritable and RowWritable, correspond to the row's Primary Key and the contents of this row, respectively. You can get the Primary Key object and the row object defined by Table Store JAVA SDK by invoking PrimaryKeyWritable.getPrimaryKey() and RowWritable.getRow().

- Configure Table Store as data source of mapper.

```
private static RangeRowQueryCriteria fetchCriteria() {
    RangeRowQueryCriteria res = new     RangeRowQueryCriteria("YourTableName");
    res.setMaxVersions(1);
    List<PrimaryKeyColumn> lower = new ArrayList<PrimaryKeyColumn>();
    List<PrimaryKeyColumn> upper = new ArrayList<PrimaryKeyColumn>();
    lower.add(new PrimaryKeyColumn("YourPkeyName", PrimaryKeyValue.INF_MIN));
    upper.add(new PrimaryKeyColumn("YourPkeyName", PrimaryKeyValue.INF_MAX));
    res.setInclusiveStartPrimaryKey(new PrimaryKey(lower));
    res.setExclusiveEndPrimaryKey(new PrimaryKey(upper));
    return res;
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "row count");
    job.addFileToClassPath(new Path("hadoop-connector.jar"));
    job.setJarByClass(RowCounter.class);
    job.setMapperClass(RowCounterMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LongWritable.class);
    job.setInputFormatClass(TableStoreInputFormat.class);
    TableStoreInputFormat.setEndpoint(job, "https://YourInstance.Region.ots.aliyuncs.com/");
    TableStoreInputFormat.setCredential(job, "YourAccessKeyId", "YourAccessKeySecret");
    TableStoreInputFormat.addCriteria(job, fetchCriteria());
    FileOutputFormat.setOutputPath(job, new Path("output"));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

In the preceding example, job.setInputFormatClass(TableStoreInputFormat.class) is used to set Table Store as the data source. To complete the example, the following steps are also required:

- Deploy hadoop-connector.jar to the cluster and add it to classpath. The local path of hadoop-connector.jar is specified by addFileToClassPath(). In the code example, is assumes that hadoop-connector.jar is in the current path.
- Specify the endpoint and Access Key when accessing Table Store. They can be set using TableStoreInputFormat.setEndpoint() and TableStoreInputFormat.setCredential().

- Specify a table to count.

> Note
> - TableStoreInputFormat.addCriteria() can be invoked multiple times. Each invocation adds a RangeRowQueryCriteria object.
> - Set setFilter() and addColumnsToGet() to filter unrequired rows and columns in server-side to reduce the cost and improve the performance of Table Store.
> - Add RangeRowQueryCriterias to multiple tables to merge them.
> - Add multiple RangeRowQueryCriterias to a single table to tune the splits. TableStore-Hadoop Connector can then split the range of the user's input based on specified requirements.

Run the program

```
$ HADOOP_CLASSPATH=hadoop-connector.jar bin/hadoop jar row-counter.jar
...
$ find output -type f
output/_SUCCESS
output/part-r-00000
output/. _SUCCESS.crc
output/.part-r-00000.crc
$ cat out/part-r-00000
TOTAL   9
```

Data type conversion

Table Store and Hive/Spark support different sets of data types.

The following table indicates data type conversion support from Table Store (rows) to Hive (columns).

	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	BOOLEAN	STRING	BINARY
INTEGER	Yes, with limited precision	Yes, with limited precision	Yes, with limited precision	Yes	Yes, with limited precision	Yes, with limited precision			
DOUBLE	Yes, with limited precision	Yes, with limited precision	Yes, with limited precision	Yes, with limited precision	Yes, with limited precision	Yes			

	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	BOOLEAN	STRING	BINARY
BOOLEAN							Yes		
STRING								Yes	
BINARY									Yes

3.2. Environment preparations

Access Table Store tables with Hive and HadoopMR

You can use Hive and HadoopMR to access data in Table Store directly by using the dependency package released by Table Store and E-MapReduce.

You can use Hive and HadoopMR to access data in Table Store directly by using the dependency package released by Table Store and E-MapReduce.

Install JDK-7+

1. Download and install the relevant installation package of JDK-7+.
   - Linux/MacOS: Use the package installation manager.
   - Windows: Click to download.
2. Check the installation status as follows.

```
$ java -version
java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
Java HotSpot(TM) 64-Bit Server VM (build 25.77-b03, mixed mode)
```

Install Hadoop

1. Download Hadoop (version 2.6.0 or later).
2. Unpack the installation package and install Hadoop for your cluster.
3. Run Hadoop as follows.

```
$ bin/start-all.sh
# Check the hadoop service
$ jps
24017 NameNode
24835 Jps
24131 DataNode
24438 ResourceManager
5114 HMaster
24287 SecondaryNameNode
24527 NodeManager
```

4. Add the path of Hadoop to `/etc/profile` and run `source /etc/profile` to make the configuration take effect.

```
export HADOOP_HOME=/data/hadoop/hadoop-2.6.0
export PATH=$PATH:$HADOOP_HOME/bin
```

Install Hive

1. Download Hive, specifically the bin.tar.gz type.
2. Unpack the installation package as follows.

```
$ mkdir /home/admin/hive-2.1.0
$ tar -zxvf apache-hive-2.1.0-bin.tar.gz -C /home/admin/
$ mv /home/admin/apache-hive-2.1.0-bin /home/admin/hive-2.1.0/
```

3. Initialize the schema as follows.

```
# Enter the specified directory
$ cd /home/admin/hive-2.1.0/

# Initialization, Derby can be replaced directly with mysql if it is MySQL
# If an error occurs, you can delete it by running rm -rf metastore_db/ and execute again.
$ ./bin/schematool -initSchema -dbType derby
```

4. Run Hive as follows.

```
$ ./bin/hive
# check hive
hive> show databases;
OK
default
Time taken: 0.207 seconds, Fetched: 1 row(s)
```

Download Java SDK for Table Store

1. Download the Java SDK dependency package (version 4.1.0 or later).

   > **Note** The SDK dependency package is updated with Java SDK. Download the dependency package according to the latest Java SDK.

2. Copy the SDK to the Hive directory as follows.

```
$ mv tablestore-4.1.0-jar-with-dependencies.jar /home/admin/hive-2.1.0/
```

Download EMR dependency package

1. Download the Alibaba Cloud EMR dependency package.

   > **Note** For more information on EMR, click here.

2. Rename the *emr-sdk_2.10-1.3.0-20161025.065936-1.jar* file.

```
mv emr-sdk_2.10-1.3.0-20161025.065936-1.jar /home/admin/hive-2.1.0/emr-sdk_2.10-1.3.0-SNAPSHOT.jar
```

4.Function Compute

4.1. Use Function Compute

You can use Function Compute to perform real-time computing on incremental data stored in Tablestore.

Context

Alibaba Cloud Function Compute is an event-driven computing service that allows you to focus on writing and uploading code without the need to manage servers. Function Compute prepares computing resources for you and runs your code in an elastic and reliable way. You need to pay only for the resources that are consumed when the code is run. For more information, see What is Function Compute? For examples on how to use Function Compute, see Examples on how to use Function Compute in Tablestore.

Tablestore Stream is a data channel used to obtain incremental data in Tablestore tables. By creating Tablestore triggers, Tablestore Stream and Function Compute can be automatically docked. This allows the custom program logic in the computing function to automatically process changed data in Tablestore tables.

Scenarios

The following figure shows the tasks that you can use Function Compute to perform.

- Data synchronization: You can use Function Compute to synchronize real-time data stored in Tablestore to data caches, search engines, or other database instances.
- Data archiving: You can use Function Compute to incrementally archive data stored in Tablestore to OSS for cold archiving.
- Event-driven application: You can create triggers to trigger functions to call API operations provided by IoT suites or cloud-based applications or send notifications.

Configure a Tablestore trigger

You can create a Tablestore trigger in the Tablestore console to process the real-time data stream generated by the incremental data in a Tablestore table.

1. Create a table and enable the stream feature for the table.
   i. Long on to the Tablestore console and create an instance.
   ii. Create a table in the created instance and enable the stream feature for the table.
2. Create a function.
   i. Log on to the Function Compute console.
   ii. In the left-side navigation pane, click **Service/Function**.
   iii. On the **Service/Function** page, click **Create Function**.
   iv. On the **Create Function** page, click **Event Function** and then click **Next**.
   v. Configure the parameters based on your requirements, and then click **Create**.

   
3. Configure services.
   i. In the left-side navigation pane, click **Service/Function**.

ii. On the **Service/Function** page, click **Service Configurations**.

iii. On the **Service Configurations** tab, configure roles used to authorize the function to collect logs and access other resources. For more information, see Permissions.

4. Create and test a Tablestore trigger.

> **Note** If you use Function Compute in the Tablestore console for the first time, authorize Tablestore to send event notifications in the previous version of the Tablestore console.

i. On the **Trigger** tab in the details page of the table, click **Use Existing Function Compute**.

ii. In the **Create Triggers** dialog box that appears, select the Function Compute service and function, and enter the name of the trigger.

> **Note** When you create a trigger in the previous version of the Tablestore console for the first time, click **Grant Tablestore the permission to send event notifications.**
>
> After authorization, you can see the AliyunTableStoreStreamNotificationRole role that is automatically created in the RAM console.

iii. Click **OK**.

Data Processing

- Data format

  A Tablestore trigger encodes the incremental data in the CBOR format to construct a Function Compute events. The following code provides an example of the format of incremental data:

```
{
  "Version": "string",
  "Records": [
    {
      "Type": "string",
      "Info": {
        "Timestamp": int64
      },
      "PrimaryKey": [
        {
          "ColumnName": "string",
          "Value": formated_value
        }
      ],
      "Columns": [
        {
          "Type": "string",
          "ColumnName": "string",
          "Value": formated_value,
          "Timestamp": int64
        }
      ]
    }
  ]
}
```

- Elements

  The following table describes the elements included in the preceding format.

Element	Description
Version	The version of the payload, which is Sync-v1. Data type: string.
Records	The array that contains the incremental data in the table. This element includes the following members: ○ Type: the type of the columns. Valid values: PutRow, UpdateRow, and DeleteRow. Data type: string. ○ Info: includes the Timestamp member, which indicates the last time when the row was modified. The value of Timestamp must be in the UTC format. Data type: int64.

Element	Description
PrimaryKey	The array that stores the primary key column. This element includes the following members: ○ ColumnName: the name of the primary key column. Data type: string. ○ Value: the content of the primary key column. Data type: formated_value. Valid values: integer, string, and blob.
Colunms	The array that stores the attribute columns. This element includes the following members: ○ Type: the type of the attribute column. Valid values: PutOneVersion, DeleteOneVersion, and DeleteAllVersions. Data type: string. ○ ColumnName: the name of the attribute column. Data type: string. ○ Value: the content of the attribute column. Data type: formated_value. Valid values: integer, boolean, double, string, and blob. ○ Timestamp: the last time when the attribute column was modified. The value of Timestamp must be in the UTC format. Data type: int64.

- Sample data

```
{
  "Version": "Sync-v1",
  "Records": [
    {
      "Type": "PutRow",
      "Info": {
        "Timestamp": 1506416585740836
      },
      "PrimaryKey": [
        {
          "ColumnName": "pk_0",
          "Value": 1506416585881590900
        },
        {
          "ColumnName": "pk_1",
          "Value": "2017-09-26 17:03:05.8815909 +0800 CST"
        },
        {
          "ColumnName": "pk_2",
          "Value": 1506416585741000
        }
      ],
      "Columns": [
        {
          "Type": "Put",
          "ColumnName": "attr_0",
          "Value": "hello_table_store",
          "Timestamp": 1506416585741
        },
        {
          "Type": "Put",
          "ColumnName": "attr_1",
          "Value": 1506416585881590900,
          "Timestamp": 1506416585741
        }
      ]
    }
  ]
}
```

Online debugging

Function Compute supports online debugging for functions. You can construct an event to trigger the function and test whether the function logic is as expected.

Tablestore events that trigger Function Compute are in the CBOR format, which is a JSON-like binary format. Therefore, you can perform online debugging in the following methods:

1. Add both "import CBOR" and "import JSON" in the code.
2. On the **Service/Function** page, click the name of the function that you want to debug.
3. On the details page of the function, click **Code**.
4. On the **Code** tab, click **Event**. On the Test Event panel that appears, select **Custom**. Copy the preceding sample data to the edit box and modify the data based on your requirements. Click **OK**.
5. Test the function.
   i. Add `records = json.loads(event)` in the code to process custom events. Click **Save and Invoke**. Check whether the results are as expected.
   ii. After you test the function by using `records=json.loads(event)`, you can modify this line of code to `records = cbor.loads(event)` and click **Save**. This way, the corresponding function logic is triggered when data is written to the Tablestore table.

   Sample code:

```
import logging
import cbor
import json
def get_attrbute_value(record, column):
    attrs = record[u'Columns']
    for x in attrs:
        if x[u'ColumnName'] == column:
            return x['Value']
def get_pk_value(record, column):
    attrs = record[u'PrimaryKey']
    for x in attrs:
        if x['ColumnName'] == column:
            return x['Value']
def handler(event, context):
    logger = logging.getLogger()
    logger.info("Begin to handle event")
    #records = cbor.loads(event)
    records = json.loads(event)
    for record in records['Records']:
        logger.info("Handle record: %s", record)
        pk_0 = get_pk_value(record, "pk_0")
        attr_0 = get_attrbute_value(record, "attr_0")
    return 'OK'
```

4.2. Use Function Compute to cleanse data

Featured by highly concurrent write performance and low storage cost, Tablestore is suitable for the storage of IoT data, logs, and monitoring data. You can write data to Tablestore, use Function Compute to cleanse the data, and then write the cleansed data back to Tablestore. In addition, you can access the original data and cleansed data in a real-time manner.

Scenario

You want to write log data that includes three fields to Tablestore. To efficiently query the logs, you need to write the logs in which the value of the level field is larger than 1 to another table. The following table describes the fields included in the logs.

Field	Type	Description
id	Integer	The ID of the log.
level	Integer	The level of the log. A larger value indicates a higher level.
message	String	The content of the log.

Create a function and a trigger

Before your create a trigger, you must enable the stream feature for the table to allow the function to process incremental data written to the table. A trigger can be bound only to an existing function. Therefore, you must create the function in the same region as that of the Tablestore instance.

1. Create a table and enable the stream feature for the table.
   i. Long on to the Tablestore console and create an instance.
   ii. Create a table in the created instance and enable the stream feature for the table.
2. Create a function.
   i. Log on to the Function Compute console.
   ii. In the left-side navigation pane, click **Service/Function**.
   iii. On the **Service/Function** page, click **Create Function**.
   iv. On the **Create Function** page, click **Event Function** and then click **Next**.
   v. Configure the function based on your requirements. Click **Create**.
      - In this example, Function Name is set to etl_test and Runtime is set to python2.7.
      - Funtion Handler is set to etl_test.handler.

      
3. Configure services.
   i. In the left-side navigation pane, click **Service/Function**.
   ii. On the **Service/Function** page, click **Service Configurations**.
   iii. On the **Service Configurations** tab, configure the role for the service. For more information, see Permissions.The function needs to write logs to Log Service and perform read and write operations on Tablestore tables. Therefore, you must authorize Function Compute to perform these operations. In this example, the AliyunOTSFullAccess and AliyunLogFullAccess permissions are granted to the role. We recommend that you follow the principle of least privilege (PoLP) when you grant permissions to the role.

      
4. Modify the function code.
   i. On the **Service/Function** page, click the name of the function you want to modify.
   ii. On the details page of the function, click **code**.
   iii. On the **Code** tab, modify and save the function code.Set the following parameters to actual values: INSTANCE_NAME, REGION, and ENDPOINT.

      

      The following code provides an example of the function used to cleanse log data:

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
import cbor
import json
import tablestore as ots
```

```
INSTANCE_NAME = 'distribute-test'
REGION = 'cn-shanghai'
ENDPOINT = 'http://%s.%s.vpc.tablestore.aliyuncs.com'%(INSTANCE_NAME, REGION)
RESULT_TABLENAME = 'result'
def _utf8(input):
    return str(bytearray(input, "utf-8"))
def get_attrbute_value(record, column):
    attrs = record[u'Columns']
    for x in attrs:
        if x[u'ColumnName'] == column:
            return x['Value']
def get_pk_value(record, column):
    attrs = record[u'PrimaryKey']
    for x in attrs:
        if x['ColumnName'] == column:
            return x['Value']
# The obtained credentials can be used to access Tablestore because the AliyunOTSFullAcce
ss permission is granted to the role.
def get_ots_client(context):
    creds = context.credentials
    client = ots.OTSClient(ENDPOINT, creds.accessKeyId, creds.accessKeySecret, INSTANCE_NA
ME, sts_token = creds.securityToken)
    return client
def save_to_ots(client, record):
    id = int(get_pk_value(record, 'id'))
    level = int(get_attrbute_value(record, 'level'))
    msg = get_attrbute_value(record, 'message')
    pk = [(_utf8('id'), id),]
    attr = [(_utf8('level'), level), (_utf8('message'), _utf8(msg)),]
    row = ots.Row(pk, attr)
    client.put_row(RESULT_TABLENAME, row)
def handler(event, context):
    records = cbor.loads(event)
    #records = json.loads(event)
    client = get_ots_client(context)
    for record in records['Records']:
        level = int(get_attrbute_value(record, 'level'))
        if level > 1:
            save_to_ots(client, record)
        else:
            print "Level <= 1, ignore."
```

```
        print "Level<= 1, ignore."
```

5. Create and test a Tablestore trigger.

> **Note** If you use Function Compute in the Tablestore console for the first time, authorize Tablestore to send event notifications in the previous version of the Tablestore console.

   i. On the **Trigger** tab in the details page of the table, click **Use Existing Function Compute**.

   ii. In the **Create Triggers** dialog box that appears, select the Function Compute service and function, and enter the name of the trigger.

> **Note** When you create a trigger in the previous version of the Tablestore console for the first time, click **Grant Tablestore the permission to send event notifications.**
>
> After authorization, you can see the AliyunTableStoreStreamNotificationRole role that is automatically created in the RAM console.

   iii. Click **OK**.

Test the function

After you create the function and trigger, write data to Tablestore and query the data to verify whether the data is cleansed as expected.

Write data to the table named source_data. Enter the values of the id, level, and message fields and query the cleansed data in the table named result. Fore more information about how to write and query data, see Read and write data in the console.

- When you write a log in which the value of the level field is larger than 1 to the source_data table, the log is synchronized to the result table.
- When you write a log in which the value of the level field is equal to or smaller than 1 to the source_data table, the log is not synchronized to the result table.