# Proposal for Google Summer of Code 2020

RocketMQ Connect IoTDB

## Basic Information

**First name:** Tsung-Han (Preferred: Jack)
**Last name:** Tsai
**Email:** tsaitsunghan@apache.org
**Github:** Jack Tsai
**Website:** https://jack870131.github.io/

## Background

### Project name

Apache RocketMQ - [COMDEV-350] RocketMQ Connect IoTDB

### Synopsis

The goal of this issue is to implement the IoTDB sink connector based on OpenMessaging Connect API, which it enables writing data from Apache RocketMQ topics to IoTDB.

### Motivation

The problem could be traced back to the reason of applying RocketMQ Connect. To connect RocketMQ with IoTDB, the traditional method is applying the plain Producer/Consumer model, e.g. the producer side send (produce) data with some manipulation languages into RocketMQ brokers directly, then the consumer client retrieves (consume) the information from the message queue and store it into IoTDB.

However, there exists several problems while applying the plain Producer/Consumer model in some situations (RocketMQ 4.X), since the method is not allowed brokers in different clusters perform data backup and message communication:

1. Message reliability and availability
   If a RocketMQ cluster has been deployed in a single region has unforeseen major problems such as damage to the server room, disk failure or server damage, the reliability of the single cluster will fail.
2. Backup & replication
   The original RocketMQ 4.X does not enable “backup and replication” between clusters, which users cannot really have full backup content in other clusters or other regions.
3. Cross-region message consumption
   For example, System A is deployed as a Producer in Region A, and System B is deployed as a Consumer in Region B. System A cannot sent the data for System B to consume.

### Solution

The implementation of the Message Connector could be applied regarding to those problems. The Connector instance is responsible for maintaining the relevant configuration of a specific data system, such as the link address and the data needs to be synchronized. After the connector instance is started, it can split partitions into several tasks based on the configuration information. The Message Connector is mainly used to synchronize messages between different regions to ensure data consistency and replication.

RocketMQ Connect, which is designed based on the concept of the Message Connector, allows the communication between multiple clusters, and effectively implements the synchronous replication of messages in RocketMQ clusters distributed between different regions. To be specific, it is responsible for obtaining data and sends into RocketMQ, then write messages from the message queue to other systems.

In conclusion, the development of the connector for IoTDB could help solving those restrictions mentioned above while applying the plain Producer/Consumer model. Besides, RocketMQ Connect provides a nicer API, which enables the tasks distribution for parallel processing. For instance, if there are some data combined with multiple topics needed to be synchronized through IoTDB, they could be separated into several task threads and be allocated to corresponding worker processes (the Connector Runtime environment). If a worker process has been damaged, the tasks on it would be transferred automatically to those are still survived, which provides a failover mechanism to enhance the fault tolerance.

## Key Modules Implementation Design

To implement the corresponding connector, there are some key components should be clarified:

| Component | Description |
|---|---|
| Source Connector | The connector responsible for obtaining data from other systems and ingesting to RocketMQ topics. |
| Sink Connector | The connector responsible for delivering data from RocketMQ topics into target systems. |
| Runtime | The runtime environment for connectors. |
| OpenMessaging Connect | OpenMessaging Connect is an API for message connect. The Source Connector or Sink Connector which implements OpenMessaging Connect could both be loaded by RocketMQ Connect Runtime. |

Due to the task description of this issue, the sink connector should be developed for RocketMQ connecting IoTDB. The implementation details would be based on OpenMessaging Connect API and divided into several modules provided below.

## Basic configuration

| Field | Type | Description |
|---|---|---|
| Host | String | The host address of IoTDB. |
| Port | String | The port of IoTDB. |
| Username | String | The username for IoTDB. |
| Password | String | The password for IoTDB. |
| Topic | String | The targeted topic name. |
| Mode | String | The mode of sink connector. Only “bulk” mode is supported now. |
| Task divide strategy | Integer | The strategy of dividing tasks. The value is “0” in default, which represents dividing tasks by topics. Each table includes a topic. |
| Task parallelism | Integer | The parallelism of tasks. The value is “1” in default, which represents dividing topics into specified number of tasks for executing. |
| Source record converter | String | The parser of the source data. |
| Name server | String | The name server which sink connector connect to. |
| Broker cluster | String | The broker cluster which sink connector connect to. |
| Refresh Interval | Long | The refresh interval of the runtime environment. |

## Task divide strategy

Determining the suited strategy of splitting topics into multiple tasks is needed for the sink connector. There are two kinds of strategies listed below, which had already been implemented in RocketMQ connecting JDBC and could applied in the project of connecting IoTDB.

| Divide By | Description |
|---|---|
| Topic | Each task is responsible for multiple topics. Each topic belongs to one task. |
| Queue | Each topic owns multiple queues. Queues are separated through multiple tasks. |

## Parser/Converter

The parser is a series of utilities which are responsible for converting the SQL statement in the message queue into data with corresponding type. There are multiple data types (e.g. Boolean, Integer, Long Integer, Float, Single Precision Floating Point, Double Precision Floating Point, String) converters which IoTDB supported should be implemented.

## Sink task

For this part, development should implement the OpenMessaging Connect API. The detail of the implementation is provided below:

| Method | Description |
| --- | --- |
| void start(KeyValue config) | Start the task and load it with a series of configurations, including the information for building connections with IoTDB. |
| void stop() | Stop the task and close the connection with the target system. |
| void pause() | Pause the task. |
| void resume() | Resume the task. |
| void put(Collection<SinkDataEntry> message) | Put the data entries to the sink, which takes records loaded from RocketMQ and sends them to IoTDB. |

## Sink connector

This is the main part of this issue, which the development should follow the OpenMessaging Connect as well. There are multiple methods should be implemented presented below:

| Method | Description |
| --- | --- |
| void verifyAndSetConfig(KeyValue config) | The configuration which had been set before should be loaded at this step and the method should be invoked before starting the connector. |
| void start() | Start the connector. To be specific, it would build the route connecting to the expected broker cluster for listening events and boot the administration tool of RocketMQ. |
| void stop() | Stop the connector. |
| void pause() | Pause the connector. |
| void resume() | Resume the connector. |
| Class<? extends Task> taskClass() | Return the task implementation for this connector, which is the sink task stated above. |
| List<KeyValue> taskConfigs() | Return a set of configurations for the sink task. |

## Consumer

In this part, the RocketMQ push consumer would be applied. There are some basic IoTDB operation methods such as data definition language or data manipulation language need to be implemented, e.g. create, update or delete targeted timeseries.

## Deliverables & Timetable

| Date | Time | Description |
|---|---|---|
| Before - 18 May | N/A | • The most important work of this stage is to build the connection with the community. The mailing list is the main method and the fastest way to engaging in the development process of RocketMQ.<br>• Learn how the RocketMQ works. Start reading issue related part of the code on Github. |
| 18 May - 1 Jun. | 2 week | • There would have the final exam of my college near the end of May as I need to spend some time prepare for it. With my experiences, working 40 hours a week on project (including weekend) could still be ensured and the exam won't affect my schedule.<br>• Gain enough background knowledge for message queue, especially on how OpenMessaging Connect work. This could be reached on reading the code of "rocketmq-connect-jdbc" project in "rocketmq-externals.<br>• Discuss and determine the configuration details with mentors.<br>• Implement the needed configuration for RocketMQ connecting IoTDB. |
| 1 Jun. - 15 Jun. | 2 week | • Determine and build task divide strategies with corresponding configurations.<br>• Implement needed parsers for IoTDB converting data into corresponding type.<br>• Prepare for the first-round evaluation. |
| 15 Jun. - 29 Jun. | 2 week | • Discuss and determine the implementation details about the sink connector main part.<br>• Implement the sink task with the configuration set before. |
| 29 Jun. - 13 Jul. | 2 week | • Build the sink connector part.<br>• Discuss and determine how consumer part should be implemented and the testing plan with mentors.<br>• Prepare for the second-round evaluation. |
| 13 Jul. - 27 Jul. | 2 week | • Build the consumer part and the user documentation.<br>• Build tests with the complete coverage. |
| 27 Jul. - 10 Aug. | 2 week | • Fix remaining bugs due to the testing results.<br>• Prepare for the final evaluation. |

# About me

## Education

**University of Liverpool**, Liverpool, UK

BSc in Computer Science with Software Development (expected graduation: 29 May 2020)

## Skills

- Proficient in Java
- Familiar with database development, e.g. MySQL, Redis and Apache IoTDB
- Familiar with development tools, e.g. Git, Maven and Checkstyle
- Familiar with web or big data frameworks, e.g. Spring, Apache Hadoop and Apache Storm

## Related experience

- Being elected as the Apache committer of IoTDB (id: tsaitsunghan) on Nov. 2019
- Software engineering intern at National Engineering Laboratory for Big Data Software in Tsinghua University from Jun. 2019 to Aug. 2019
- Participated in Hacktoberfest held by DigitalOcean (promoted the concept of open source, contributed at least 5 pull requests, wrote and modified documentations) on Oct. 2018
- Web developer intern at Tokyo Keiso Taiwan Co.,Ltd. from Jun. 2018 to Aug. 2018

## Open source contribution & Issues fixed or participated

### Apache IoTDB

IOTDB-418 Refactor query process in IoTDB PR#713

IOTDB-279 Merge TsDigest into Statistics PR#553

IOTDB-191 Enrich Session interfaces PR#375

IOTDB-13 Support batched ingestion PR#331

IOTDB-174 Add interfaces for querying device or timeseries number PR#435

IOTDB-73 A new encoding method for regular timestamp column PR#231

### Checkstyle

#5879 New Filter SuppressionXpathSingleFilter PR#6368

#5832 Code samples for Naming Checks PR#6086, PR#6102, PR#6103, PR#6105, PR#6408, PR#6425, PR#6426, PR#6431, PR#6434, PR#6778, PR#6779