# 背景

应用于生产环境的MYSQL数据库集群，为传统的互为主从的复制拓扑架构。随着业务处理的数据量增加，MYSQL节点间的复制链路延迟加大，如果此时发生代理IP切换，可能会发生数据错乱，影响正常业务。为解决数据库节点间复制延迟问题，需对数据库进行物理拆分，以达到降低单节点的数据处理压力，从而减缓复制链路延迟。

在进行MYSQL实例物理拆分后，需按照拆分规则进行迁移数据，并且尽可能保证较短的停机时间。

# 目标

基于MYSQL物理拆分后的热迁移，实现较短的停机时间，将是本文探索的目标。

# 总体设计

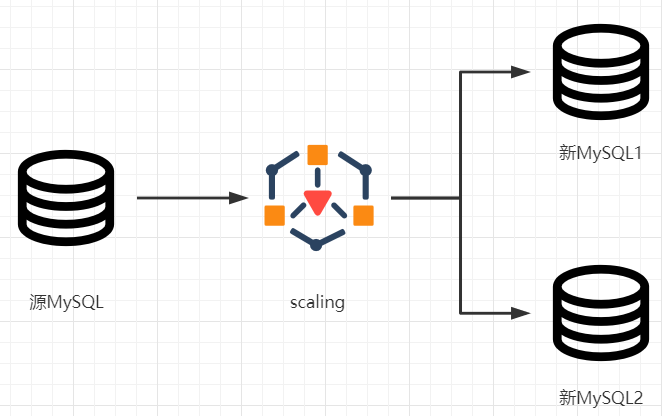


图 1 - 总体设计

1. 按照拆分规则，根据需要创建MySQL实例及库表，包括广播表和默认数据源表；
2. 按照拆分策略，配置并启动scaling任务；
3. 监控scaling任务进度情况，在确认无数据变更的情况下，完成应用数据源的切割。

**注意**：

1. 源MySQL、新MySQL实例均为高可用架构集群；新MySQL实例可能不止2个；
2. 如涉及到广播表，或默认数据源表，则需单独处理，不在本文档讨论范围内。

# 环境说明

|  |  |  |
| --- | --- | --- |
| 角色 | 物理IP | 部署服务 |
| 源MySQL | 88.88.16.126 | MySQL 5.7  SHARDING-PROXY 4.1.0  SHARDING-SCAILING 4.1.0  ZOOKEEPER 3.4.9 |
| 新MySQL1 | 88.88.16.112 | MySQL 5.7 |
| 新MySQL2 | 88.88.16.113 | MySQL 5.7 |

表 1 - 环境说明

如果条件允许，建议proxy, scaling组件分开部署。

# 迁移准备

## 安装zookeeper

zookeeper组件是sharding-proxy、sharding-scaling组件的前置条件。因proxy和scaling组件为迁移数据临时使用，故zookeeper采取单机模式进行部署。

进入https://archive.apache.org/dist/zookeeper/，下载zookeeper 3.4.9安装包。

tar -zxf zookeeper-3.4.9.tar.gz -C /opt/

进入解压后的目录，启动zk：

bin/zkServer.sh start

## 安装数据库

按照既定的拆分规则，创建MySQL物理实例，请注意，这里的MySQL物理实例可能不止2个，可能是多个。

在新MySQL1上安装MySQL实例，略。

在新MySQL2上安装MySQL实例，略。

## 准备sharding安装包

进入链接：<https://shardingsphere.apache.org/document/4.1.1/cn/downloads/>



图 2 - sharding下载

此处，我们仅需下载sharding-proxy和sharding-scaling的二进制包。

## 创建分片表（\*）

数据分片规则如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 源MySQL | 新MySQL1 | 新MySQL2 | 分库规则 | 分表规则 |
| t\_order | t\_order\_0  t\_order\_1 | t\_order\_0  t\_order\_1 | user\_id % 2 | order\_id % 2 |
| t\_order\_item | t\_order\_item\_0  t\_order\_item\_1 | t\_order\_item\_0  t\_order\_item\_1 | user\_id % 2 | order\_id % 2 |
| t\_dt | t\_dt\_0  t\_dt\_1 | t\_dt\_0  t\_dt\_1 | id % 2 | id % 2 |

表 2 - 分片规则

这里的分片规则，只是作为演示所用。具体分片规则，包括建表语句和配置信息，由开发同事提供。

在准备分片表之前，确保新MySQL实例依照生产部署要求部署完毕，在新MySQL1上创建schema和表：

create database demo\_ds\_0 default character set utf8mb4;

use demo\_ds\_0;

create table t\_order\_0

(

order\_id bigint not null auto\_increment primary key,

user\_id bigint not null,

name varchar(100)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

create table t\_order\_1

(

order\_id bigint not null auto\_increment primary key,

user\_id bigint not null,

name varchar(100)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `t\_order\_item\_0` (

`order\_id` bigint(20) NOT NULL,

`item` varchar(100) DEFAULT NULL,

`user\_id` bigint(20) NOT NULL,

PRIMARY KEY (`order\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `t\_order\_item\_1` (

`order\_id` bigint(20) NOT NULL,

`item` varchar(100) DEFAULT NULL,

`user\_id` bigint(20) NOT NULL,

PRIMARY KEY (`order\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE t\_dt\_0 (

id bigint(20) NOT NULL AUTO\_INCREMENT,

context varchar(2048) DEFAULT NULL,

dt datetime DEFAULT NULL,

ts timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (id)

);

CREATE TABLE t\_dt\_1 (

id bigint(20) NOT NULL AUTO\_INCREMENT,

context varchar(2048) DEFAULT NULL,

dt datetime DEFAULT NULL,

ts timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (id)

);

在新MySQL2上创建schema和表：

create database demo\_ds\_1 default character set utf8mb4;

use demo\_ds\_1;

create table t\_order\_0

(

order\_id bigint not null auto\_increment primary key,

user\_id bigint not null,

name varchar(100)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

create table t\_order\_1

(

order\_id bigint not null auto\_increment primary key,

user\_id bigint not null,

name varchar(100)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `t\_order\_item\_0` (

`order\_id` bigint(20) NOT NULL,

`item` varchar(100) DEFAULT NULL,

`user\_id` bigint(20) NOT NULL,

PRIMARY KEY (`order\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `t\_order\_item\_1` (

`order\_id` bigint(20) NOT NULL,

`item` varchar(100) DEFAULT NULL,

`user\_id` bigint(20) NOT NULL,

PRIMARY KEY (`order\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE t\_dt\_0 (

id bigint(20) NOT NULL AUTO\_INCREMENT,

context varchar(2048) DEFAULT NULL,

dt datetime DEFAULT NULL,

ts timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (id)

);

CREATE TABLE t\_dt\_1 (

id bigint(20) NOT NULL AUTO\_INCREMENT,

context varchar(2048) DEFAULT NULL,

dt datetime DEFAULT NULL,

ts timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (id)

);

## 安装proxy

Proxy为热迁移数据准备，将[准备sharding安装包](#_准备sharding安装包)进行安装部署。

mkdir -p /opt/shardingsphere

tar -zxf apache-shardingsphere-4.1.1-sharding-proxy-bin.tar.gz -C /opt/shardingsphere/

mv /opt/shardingsphere/apache-shardingsphere-4.1.1-sharding-proxy-bin /opt/shardingsphere/sharding-proxy-4.1.1

# 拷贝MySQL驱动包到proxy lib目录下

cp mysql-connector-java-5.1.47.jar /opt/shardingsphere/sharding-proxy-4.1.1/lib/

### 配置分片规则（\*）

vim /opt/shardingsphere/sharding-proxy-4.1.1/conf/config-sharding.yaml

schemaName: sharding\_db

dataSources:

ds\_0:

url: jdbc:mysql://88.88.16.112:3306/demo\_ds\_0?serverTimezone=Asia/Shanghai&useSSL=false&characterEncoding=utf8&character\_set\_server=utf8mb4&connectionCollation=utf8mb4\_bin

username: root

password: root

connectionTimeoutMilliseconds: 30000

idleTimeoutMilliseconds: 60000

maxLifetimeMilliseconds: 1800000

maxPoolSize: 50

ds\_1:

url: jdbc:mysql://88.88.16.113:3306/demo\_ds\_1?serverTimezone=Asia/Shanghai&useSSL=false&characterEncoding=utf8&character\_set\_server=utf8mb4&connectionCollation=utf8mb4\_bin

username: root

password: root

connectionTimeoutMilliseconds: 30000

idleTimeoutMilliseconds: 60000

maxLifetimeMilliseconds: 1800000

maxPoolSize: 50

shardingRule:

tables:

t\_order:

actualDataNodes: ds\_${0..1}.t\_order\_${0..1}

tableStrategy:

inline:

shardingColumn: order\_id

algorithmExpression: t\_order\_${order\_id % 2}

keyGenerator:

type: SNOWFLAKE

column: order\_id

t\_order\_item:

actualDataNodes: ds\_${0..1}.t\_order\_item\_${0..1}

tableStrategy:

inline:

shardingColumn: order\_id

algorithmExpression: t\_order\_item\_${order\_id % 2}

keyGenerator:

type: SNOWFLAKE

column: order\_item\_id

t\_dt:

actualDataNodes: ds\_${0..1}.t\_dt\_${0..1}

tableStrategy:

inline:

shardingColumn: id

algorithmExpression: t\_dt\_${id % 2}

databaseStrategy:

inline:

shardingColumn: id

algorithmExpression: ds\_${id % 2}

keyGenerator:

type: SNOWFLAKE

column: id

bindingTables:

- t\_order,t\_order\_item

defaultDatabaseStrategy:

inline:

shardingColumn: user\_id

algorithmExpression: ds\_${user\_id % 2}

defaultTableStrategy:

none:

注意：该处配置信息，在生产环境中进行配置时，需由研发同事提供正式的生产配置信息。

### 配置治理及权限

该部分配置依赖于zookeeper服务，确保[安装zookeeper](#_安装zookeeper)正确并且成功启动。

vi /opt/shardingsphere/sharding-proxy-4.1.1/conf/server.yaml

orchestration:

orchestration\_ds:

orchestrationType: registry\_center,config\_center,distributed\_lock\_manager

instanceType: zookeeper

serverLists: localhost:2181

namespace: orchestration

props:

overwrite: true

retryIntervalMilliseconds: 500

timeToLiveSeconds: 60

maxRetries: 3

operationTimeoutMilliseconds: 500

authentication:

users:

root:

password: root

sharding:

password: sharding

authorizedSchemas: sharding\_db

props:

max.connections.size.per.query: 1

acceptor.size: 16 # The default value is available processors count \* 2.

executor.size: 16 # Infinite by default.

proxy.frontend.flush.threshold: 128 # The default value is 128.

# LOCAL: Proxy will run with LOCAL transaction.

# XA: Proxy will run with XA transaction.

# BASE: Proxy will run with B.A.S.E transaction.

proxy.transaction.type: LOCAL

proxy.opentracing.enabled: false

proxy.hint.enabled: false

query.with.cipher.column: true

sql.show: true

allow.range.query.with.inline.sharding: false

### 启动proxy

完成相关配置后，启动proxy。

cd /opt/shardingsphere/sharding-proxy-4.1.1/bin

# 指定启动端口，该端口用于程序连接的代理端口

./start.sh 13308

tail -n 20 ../logs/stdout.log

检查日志，查看是否启动成功。

使用客户端连接proxy, `mysql -uroot -proot -h88.88.16.126 -P13308`，登录成功，即代理可用。

通过show tables可以正确查看经过proxy代理后的表。

mysql> use sharding\_db;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

sDatabase changed

mysql> show tables;

+---------------------+

| Tables\_in\_demo\_ds\_0 |

+---------------------+

| t\_dt |

| t\_order |

| t\_order\_item |

+---------------------+

提示：如果仍然看到分片表，说明config-sharding.yaml配置有问题。

## 安装scaling

Sharding-scaling组件为迁移实施的核心组件，其主要职责是监控源端数据变更，并应用到新MySQL实例，保证源端和目标端的数据完全一致。

tar -zxf apache-shardingsphere-4.1.1-sharding-scaling-bin.tar.gz -C /opt/shardingsphere

mv /opt/shardingsphere/apache-shardingsphere-4.1.1-sharding-scaling-bin /opt/shardingsphere/sharding-scaling

# 拷贝MySQL驱动包

cp /opt/shardingsphere/sharding-proxy-4.1.1/lib/mysql-connector-java-5.1.47.jar /opt/shardingsphere/sharding-scaling/lib/

cd /opt/shardingsphere/sharding-scaling/

# 根据需要调整server.yaml，4.1.1无需配置zk， 5+版本需要配置zk

vim conf/server.yaml

sh bin/start.sh

tail -n 50 logs/stdout.log

日志一般显示ACTIVE状态，则表示scaling组件启动成功。

[INFO ] 11:19:22.879 [main] o.a.s.shardingscaling.Bootstrap - ShardingScaling is server on http://127.0.0.1:8888/

[INFO ] 11:19:22.880 [nioEventLoopGroup-2-1] i.n.handler.logging.LoggingHandler - [id: 0x708f386c, L:/0.0.0.0:8888] ACTIVE

使用curl命令再次确认正常运行。

curl -X GET http://localhost:8888/shardingscaling/job/list

应答应为：

{"success":true,"errorCode":0,"errorMsg":null,"model":[]}

# 迁移实施

在经过[迁移准备](#_迁移准备)工作后，还需确保scaling组件部署所在的服务器，到源MySQL和新MySQL间的网络关系畅通。

## 确认迁移表（\*）

结合[配置分片规则（\*）](#_配置分片规则（*）)，再次询问研发同事需进行迁移的表，这部分待迁移的表，需提前在Proxy的“分片规则”（config-sharding.yaml）中配置完毕。

此处，我们迁移表为：t\_order, t\_order\_item, t\_dt三张表。

## 启动迁移任务

在scaling组件所在的服务器上，创建迁移任务。

curl -X POST --url http://127.0.0.1:8888/shardingscaling/job/start \

--header 'content-type: application/json' \

--data '{

"ruleConfiguration": {

"sourceDatasource": "ds\_0: !!org.apache.shardingsphere.orchestration.core.configuration.YamlDataSourceConfiguration\n dataSourceClassName: com.zaxxer.hikari.HikariDataSource\n properties:\n jdbcUrl: jdbc:mysql://88.88.16.126:3306/scaling?serverTimezone=Asia/Shanghai&useSSL=false&characterEncoding=utf8&character\_set\_server=utf8mb4&connectionCollation=utf8mb4\_bin\n driverClassName: com.mysql.jdbc.Driver\n username: root\n password: root\n connectionTimeout: 30000\n idleTimeout: 60000\n maxLifetime: 180000\n maxPoolSize: 2\n minPoolSize: 1\n maintenanceIntervalMilliseconds: 30000\n readOnly: false\n",

"sourceRule": "tables:\n t\_order\_item:\n actualDataNodes: ds\_0.t\_order\_item\n t\_order:\n actualDataNodes: ds\_0.t\_order\n t\_dt:\n actualDataNodes: ds\_0.t\_dt",

"destinationDataSources": {

"name": "dt\_1",

"password": "root",

"url": "jdbc:mysql://127.0.0.1:13308/sharding\_db?serverTimezone=UTC&useSSL=false&characterEncoding=UTF-8",

"username": "root"

}

},

"jobConfiguration": {

"concurrency": 1

}

}'

出现“{"success":true,"errorCode":0,"errorMsg":null,"model":null}”，则表示迁移任务创建成功。

## 监控迁移任务

在scaling组件所在的服务器上，通过rest http接口获取任务列表，

curl -X GET \

<http://localhost:8888/shardingscaling/job/list>

获取当前任务的状态：

curl -X GET \

<http://localhost:8888/shardingscaling/job/progress/1>

任务ID来自任务列表。通常来说，通过多次获取任务的状态，状态是不一样的，尤其binlog的文件位置。

curl -X GET http://localhost:8888/shardingscaling/job/progress/1

{"success":true,"errorCode":0,"errorMsg":null,"model":{"id":1,"jobName":"Local Sharding Scaling Job","status":"RUNNING","syncTaskProgress":[{"id":"127.0.0.1-3306-scaling","status":"SYNCHRONIZE\_REALTIME\_DATA","historySyncTaskProgress":[{"id":"history-scaling-t\_order\_item#0","estimatedRows":957,"syncedRows":957},{"id":"history-scaling-t\_order#0","estimatedRows":12,"syncedRows":12},{"id":"history-scaling-t\_dt#0","estimatedRows":6,"syncedRows":6}],"realTimeSyncTaskProgress":{"id":"realtime-scaling","delayMillisecond":1084,"logPosition":{"filename":"mysql-bin.000022","position":332278,"serverId":0}}}]}}

historySyncTaskProgress表示历史数据的同步情况，realTimeSyncTaskProgress表示实时数据的同步进度。

## 验证迁移数据

虽然可通过[监控迁移任务](#_监控迁移任务)可以查看迁移的具体进度，但仍需进行数据验证。

鉴于数据量较大，不会对全量数据进行校验，基于分片键对个例数据进行历史数据和热数据校验。

### 历史数据校验

这里，以t\_order\_item为例，该表以user\_id分库、trade\_id分表。以部分user\_id为抽检对象，对一定历史时间范围内的数据分别在源MySQL和Proxy中进行计数统计，查看计数结果是否一致。如果一致，则认为历史数据无问题。

select count(\*) from t\_order\_item where user\_id = 9999 and ts >= '2021-09-01' and ts < '2021-10-01';

### 热数据校验

在完成充分的历史数据校验后，需对实时迁移的热数据进行抽检。这里，同样以部分近一个小时有记录的user\_id为抽检对象，对近一个小时范围的数据分别在源MySQL和Proxy中进行计数统计，查看计数结果是否一致。如果一致，则认为热迁移数据无问题。

select count(\*) from t\_order\_item where user\_id = 8888 and ts >= '2021-09-28 10:00:00' and ts <= '2021-09-28 11:00:00';

## 完成迁移任务

通常来说，在完成应用的数据源切割后，即应用完全不再操作源MySQL，而是操作分片后的数据源，才算完成迁移任务。

在进行应用数据源切割前，迁移进度应趋近于实时，即没有延迟。可以通过[监控迁移任务](#_监控迁移任务)进行监控任务具体进度。

当源MySQL没有数据变更时，position字段是不会发生变化，有必要记录当前的binlog文件信息及其位置信息。

curl -X GET http://localhost:8888/shardingscaling/job/progress/1

{"success":true,"errorCode":0,"errorMsg":null,"model":{"id":1,"jobName":"Local Sharding Scaling Job","status":"RUNNING","syncTaskProgress":[{"id":"88.88.16.126-3306-scaling","status":"SYNCHRONIZE\_REALTIME\_DATA","historySyncTaskProgress":[{"id":"history-scaling-t\_order\_item#0","estimatedRows":2059,"syncedRows":2059},{"id":"history-scaling-t\_order#0","estimatedRows":12,"syncedRows":12},{"id":"history-scaling-t\_dt#0","estimatedRows":6,"syncedRows":6}],"realTimeSyncTaskProgress":{"id":"realtime-scaling","delayMillisecond":2496,"logPosition":{"filename":"mysql-bin.000025","position":20880623,"serverId":0}}}]}}

如确保源MySQL无数据操作后，同时关注position没有发生变化，则应用可立即切换至新数据源。

注意：在应用切割过程中，须避免应用在旧新数据源同时写入的情况，尽可能在无数据操作的情况下，完成应用切割。

在完成应用切割后，上述position值理应和切割前一致，如不一致，则需分析是什么操作造成position变更。

可以通过mysqlbinlog分析binlog日志，其记录了发生的DML。

mysqlbinlog --no-defaults --base64-output=decode-rows -v --start-position 20880623 mysql-bin.000025 |more

如果没有停止迁移任务，上述这部分操作已同步至经过分片后的新数据源。当应用完全切割后，停止迁移任务。

curl -X POST \

http://localhost:8888/shardingscaling/job/stop \

-H 'content-type: application/json' \

-d '{

"jobId":1 }'

# 回退

如果在测试过程中，发现异常需要回退，应用直接切回源数据。

如果在升级后，发现有异常，则无法回退。