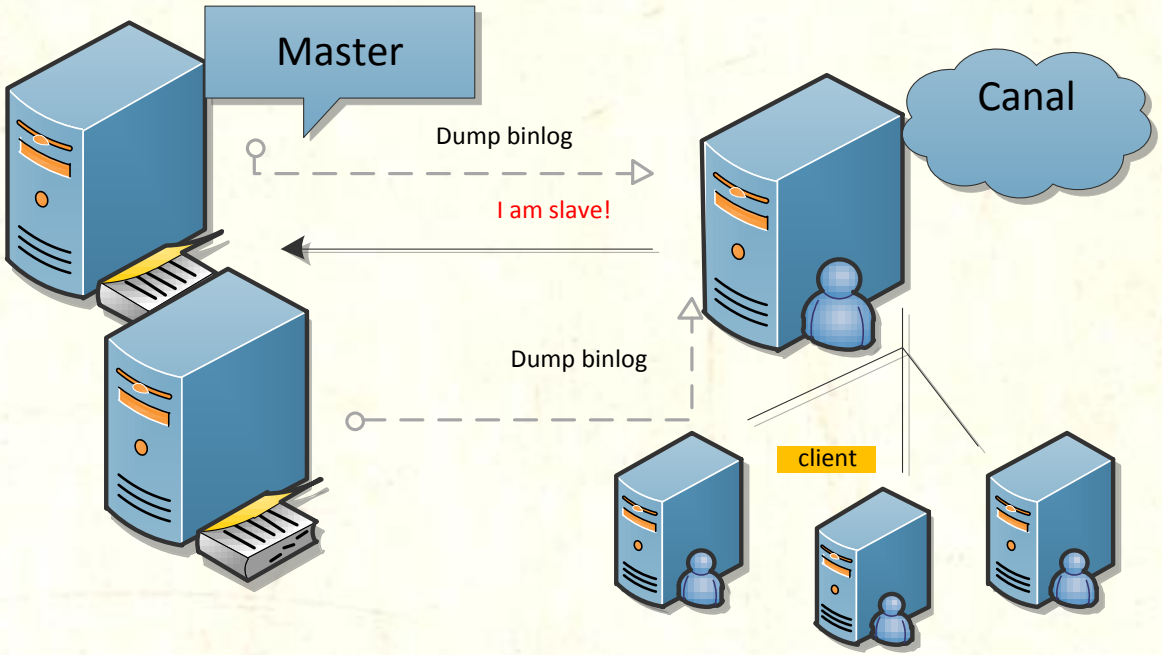
ROMA MySQL ASYNC MESSAGE SYSTEM ANALYSIS

Our MySQL async message system Roma has already online. Here I want to talk about the process and explain the advantage of this system.

Roma is based on Canal system which is used in Alibaba Inc. for different platform data flow translations. Canal acts as a slave of master and send quest to it, then master database will dump binlog event to Canal server.

Canal will miner MySQL binlogs and translate them into messages, clients can subscribe these messages and do some other work later. Roma was marked as a subscriber behind Canal system.

The architecture of Canal



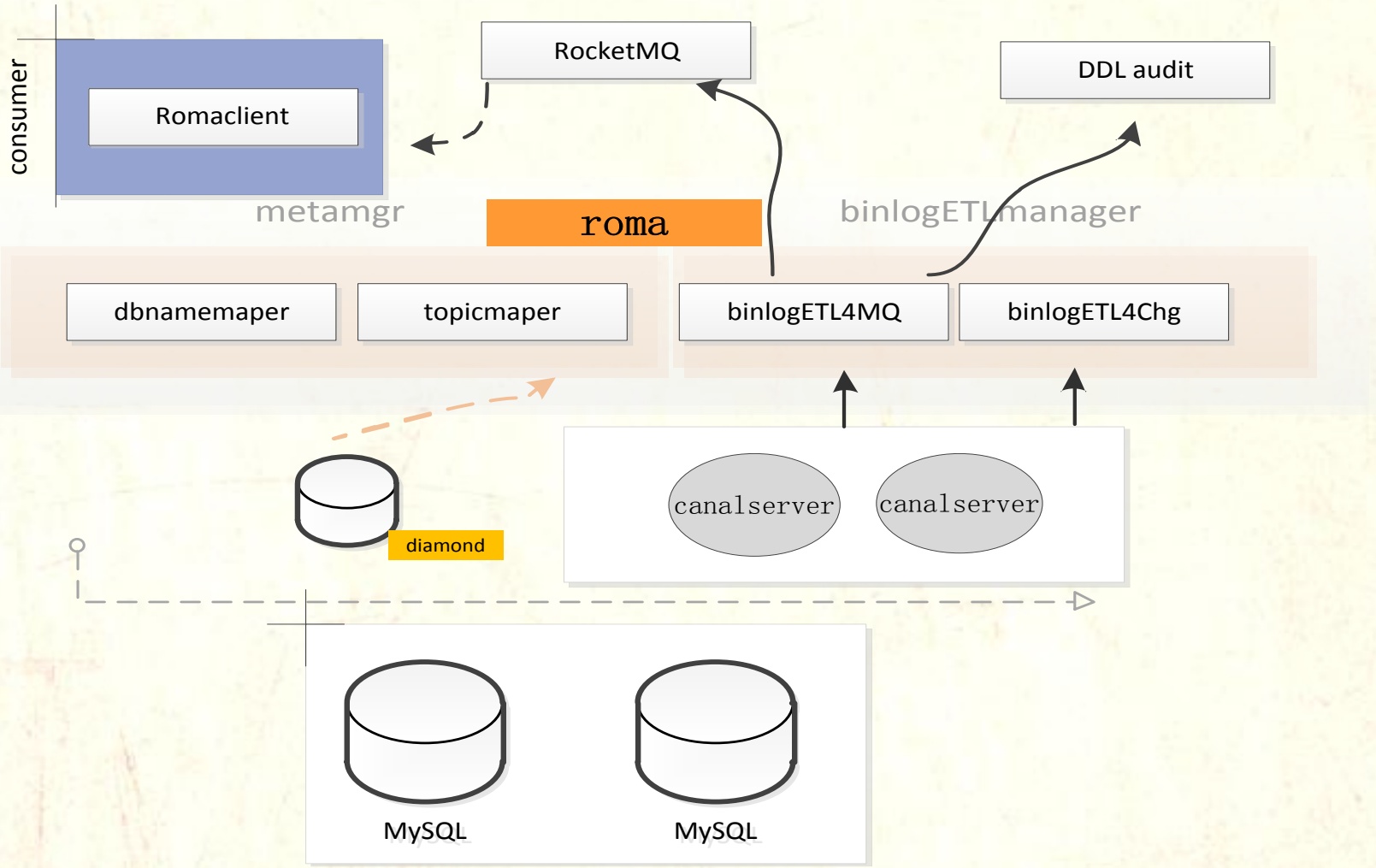
Process of Canal work:

1. Canal tells MySQL Master (hey ,I'm Slave pleases sent binlog event to me)
2. Master gets this message and finds that 'real' Slave (ok! dump binlog event to this Slave ☺)
3. Canal gets the binlog event and calls the event parser to miner event into the formatted message
4. Roma subscribes these messages then stores them in it 's own store
5. Everyone who wants to subscribe these message can ask Roma system

Why not use Canal as the final provider for customers? In other words why add Roma server behind Canal? The most important reason is that Canal can't store messages in a solid state for its memory is used as RingBuffer, a ring circle based on arrays, that when you insert a new record, the old one will be overwritten then some message will be lost. Since we must keep everything in our system, we wrote Roma which uses rocketmq as a data storage (we can keep data on MetaQ for a long time). In that way, Roma acts as a subscriber behind Canal and a provider for customers.

Architecture of Roma:

1. Based on Canal's messages
2. Roma subscribes Canal message
3. Roma puts its message in MetaQ
4. Canal opens ddl isolation, Roma subscribes ddl record


consumer
Romaclient
RocketMQ
DDL audit
metamgr
roma
binlogETLmanager
dbnamemaper
topicmaper
binlogETL4MQ
binlogETL4Chg
diamond
canalserver
canalserver
MySQL
MySQL

Roma server subscribes what Canal server has provided, let us talk about the details

First, MySQL database, we always do high availability for MySQL database, Canal can use these features to provide its own HA algorithm, Canal will do these steps when MySQL failover happens:

Canal server configurations

1. First look at Canal server configuration for MySQL database Master -> standby

 # position info
Canal.instance.Master.address = 10.20.144.25:3306
Canal.instance.Master.journal.name =
Canal.instance.Master.position =
Canal.instance.Master.timestamp =

Canal.instance.standby.address = 10.20.144.29:3306
Canal.instance.standby.journal.name =
Canal.instance.standby.position =
Canal.instance.standby.timestamp =

2. When heartbeatHaEnable = true and also Canal got threshold of retry time
3. When interval time * retry has been reached, Canal will switch to standby
4. Canal finds new binlog position from new MySQL database using (timestamp minus fallbackIntervalInSeconds)

5. Canal will receive new binlog events from MySQL database

What is fallbackIntervalInSeconds：when MySQL failover starts，Canal will find timestamp to identify which is the binlog position in new MySQL database. This timestamp should be earlier than which Canal has already determined (we consider this new timestamp should be (last timestamp – fallbackintervalseconds)). But if these steps have been done, Canal will produce some repeat records that have already been parsed.

Slave database always stay behind master (we usually say there is a lag between them) so we need go back to old position due to the “lag”. For some other reasons we also need to do so -- system time difference or IDC time zone difference etc. It needs to roll back fallbackIntervalInSeconds to guarantee no data loss.

But nothing is absolute. If Slave loses data in replication process, no matter how wonderful Canal is, the data will be lost。

In this process ,Canal finds the final position based on timestamp -- using timestamp minus fallbackIntervalInSeconds to find new binlog position – below is the code ：

```
if (StringUtils.isEmpty(entryPosition.getJournalName())) {
                    // 如果没有指定 binlogName，尝试按照 timestamp 进行查找
                    if (entryPosition.getTimestamp() != null && entryPosition.getTimestamp() > 0L) {
                        return findByStartTimeStamp(MySQLConnection, entryPosition.getTimestamp());
                    } else {
                        return findEndPosition(MySQLConnection); // 默认从当前最后一个位置进行消费
                    }
                } else {
                    if (entryPosition.getPosition() != null && entryPosition.getPosition() > 0L) {
                        // 如果指定 binlogName + offest，直接返回
```

```
                    return entryPosition;
                } else {
                    EntryPosition specificLogFilePosition = null;
                    if (entryPosition.getTimestamp() != null && entryPosition.getTimestamp() > 0L) {
                        // 如果指定 binlogName + timestamp，但没有指定对应的 offest，尝试根据时间找一下 offest
                        EntryPosition endPosition = findEndPosition(MySQLConnection);
                        if (endPosition != null) {
                            specificLogFilePosition = findAsPerTimestampInSpecificLogFile(
                                                                        MySQLConnection,
                                                                        entryPosition.getTimestamp(),
                                                                        endPosition,
                                                                        entryPosition.getJournalName());
                        }
                    }

                    if (specificLogFilePosition == null) {
                        // position 不存在，从文件头开始
                        entryPosition.setPosition(BINLOG_START_OFFEST);
                        return entryPosition;
                    } else {
                        return specificLogFilePosition;
                    }
                }
            }
```

```
        } else {
            if (logPosition.getIdentity().getSourceAddress().equals(MySQLConnection.getConnector().getAddress())) {
                return logPosition.getPostion();
            } else {
                // 针对切换的情况，考虑回退时间
                long newStartTimestamp = logPosition.getPostion().getTimestamp() - fallbackIntervalInSeconds * 1000;
                return findByStartTimeStamp(MySQLConnection, newStartTimestamp);
            }
        }
    }
```

But problem persists, although Canal calculates a new timestamp, it may still could not find the right position (slave lags too much). So image that Canal minuses timestamp with fallbackIntervalInSeconds data and gets a new timestamp 12:00:00, but slave does not reach this timestamp at the moment, then data will be lost. New master's(original slave) binlog position is the key point in this process, Canal will find new binlog position automatically ,then start parsing binlog from the new master database.

This algorithm will produce duplicate messages. Canal can solve it by itself with filtering duplicate message on server & client site (we can also do idempotent protection in the end of services).

Generally, Canal cannot guarantee data is correct whenever MySQL database failover happens (if data loss happens in MySQL replication layer, no one will save these)

```
mysql> show processlist;
+----------+--------------+---------------------+--------------+-------------+----------+-----------------------------------------------------------------------+------------------+
| Id       | User         | Host                | db           | Command     | Time     | State                                                                 | Info             |
+----------+--------------+---------------------+--------------+-------------+----------+-----------------------------------------------------------------------+------------------+
|  1231388 | repl         | 10.128.5.21:23908   | NULL         | Binlog Dump | 13735275 | Master has sent all binlog to slave; waiting for binlog to be updated | NULL             |
|  4839184 | repl         | 10.128.6.28:53210   | NULL         | Binlog Dump | 10182991 | Master has sent all binlog to slave; waiting for binlog to be updated | NULL             |
|  4889714 | dbadmin      | 10.128.3.57:31448   | test         | Sleep       |        2 |                                                                       | NULL             |
|  6770498 | repl         | 10.128.5.21:64357   | NULL         | Binlog Dump |  8313335 | Master has sent all binlog to slave; waiting for binlog to be updated | NULL             |
|  9522480 | dbadmin      | 10.128.6.226:50489  | NULL         | Sleep       |        9 |                                                                       | NULL             |
| 11868579 | binlog       | 10.128.7.68:42497   | NULL         | Sleep       |        2 |                                                                       | NULL             |
| 11868654 | binlog       | 10.128.7.68:42506   | NULL         | Binlog Dump |  3712068 | Master has sent all binlog to slave; waiting for binlog to be updated | NULL             |
| 11968540 | tigase       | 10.128.3.44:39114   | tigase       | Sleep       |        2 |                                                                       | NULL             |
```

In our MHA environment, we still set maser&standby parameters. MHA VIPs are not useful to Canal servers (different from business services)

MySQL serverId
Canal.instance.MySQL.SlaveId9 = 1234

 # position info
Canal.instance.Master.address9 = xxx.xxx.xxx.xxx:3306
Canal.instance.Master.journal.name9 =
Canal.instance.Master.position9 =
Canal.instance.Master.timestamp9 =

Canal.instance.standby.address9 = xxx.xxx.xxx.xxx:3306
Canal.instance.standby.journal.name9 =
Canal.instance.standby.position9 =
Canal.instance.standby.timestamp9 =

```
# username/password
Canal.instance.dbUsername9 = binlog
Canal.instance.dbPassword9 = xxxxxxxxx
Canal.instance.defaultDatabaseName9 =
Canal.instance.connectionCharset9 = UTF-8

# table regex
Canal.instance.filter.regex9 = .*\\..*
# table black regex
Canal.instance.filter.black.regex9 = pajk_monitor\\..*
#################################################
```

Canal server self HA

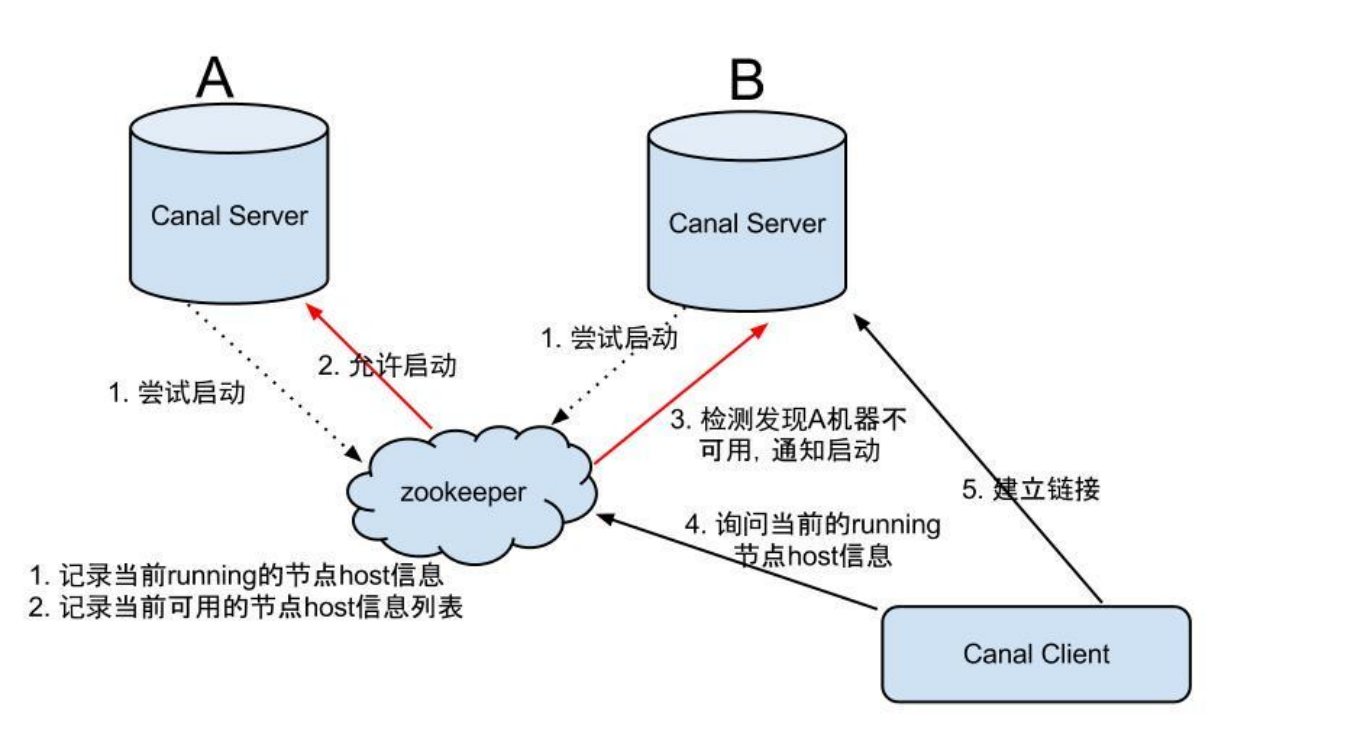
Reference documents:

Two parts of Canal HA, Canal client and Canal server

Canal server: Instances in different servers must keep one running at the same time, others are keeping standby state to reduce requests from MySQL dump.

Canal client: For one instance, just only one Canal client can do get/ack/rollback commands at the same time to keep message in order.

Whole HA depends on zookeeper, watcher and EPHEMERAL nodes and session lifecycle binding.



Canal server

Following are the steps：

1. Canal server sends zookeeper a request to start one instance (create EPHEMERAL node, who successes first will get the privileges)
2. After creating zk node successfully, the Canal server will create instance and other servers will be in standby state.

3. If zookeeper finds Canal server A has disappeared, it will inform other Canal servers to repeat step 1 immediately to elect a new candidate to create instance.
4. Whenever Canal client tries to connect, it will ask zookeeper who has already created instance. Clients will keep trying to reconnect to servers if connection failure happens.

Canal client is similar to Canal server, both of them intent to grab EPHEMERAL node

Tips:

1. If you choose memory mode for Canal event store, no position will be kept. Next time when Canal starts,it will read the last position which has been subscribed successfully(last 'ack' submit point)

```
Canal.id= 1
Canal.ip=
Canal.port= 11111
Canal.zkServers = xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181,xxx.xxx.xxx.xxx:2181
# flush data to zk
Canal.zookeeper.flush.period = 1000
# flush meta cursor/parse position to file
Canal.file.data.dir = ${Canal.conf.dir}
Canal.file.flush.period = 1000
## memory store RingBuffer size, should be Math.pow(2,n)
Canal.instance.memory.buffer.size = 16384
## memory store RingBuffer used memory unit size , default 1kb
```

Canal.instance.memory.buffer.memunit = 1024
meory store gets mode used MEMSIZE or ITEMSIZE
Canal.instance.memory.batch.mode = MEMSIZE

GroupEventParser and GroupEventSink

It is popular among internet companies to split database into many physical instances. In this situation, if you want to combine them into one logical instance, Group parameter will be helpful. Using these two parameters will help you to create new logical instance for subscribing.

If database has been split into four shards, every shard has one instance. If you don't make up a group, you will have to create four clients to subscribe each shard rather than one client to access all of them.

GroupEventParser: multi threads event parser, parallel parse.
GroupEventSink: Corresponding to groupeventparser ,global timestamp sort

Main for global group ,according to this architecture ,if you split database db into db1:db2:db3:db4 split table user into user1:user2:user3:user4：

Db1 -> user1+user2+user3+user4

You can use Roma create logical topic + tag => logical:db + logical:user as table merge

Modify logical db mapping com.pajk.Roma.dbs/ROMA_GROUP(dataId/组名).
[{"logicDb":"logicDb1","realDbs":"realDb11,realDb12"},{"logicDb":"logicDb2","realDbs":"realDb21,realDb22"}]

Modify logical table mapping com.pajk.Roma.tbs/ROMA_GROUP(dataId/组名).
[{"logicTb":"logicTb1","realTbs":"realTb11,realTb12"},{"logicTb":"logicTb2","realTbs":"realTb21,realTb22"}]

Modify logical db mapping to topic com.pajk.Roma.topics/ROMA_GROUP(dataId/组名).
[{"logicDb":"logicDb1","topic":"topic1"},{"logicDb":"logicDb2","topic":"topic2"}]
。

Modidfy logical table mapping to tag com.pajk.Roma.tags/ROMA_GROUP(dataId/组名).
[{"logicTb":"logicTb1,field1","tag":"tag1,f1"},{"logicTb":"logicTb2","tag":"tag2"}]

Client will subscribe topic in logic:db+logic:table format direct.

Advantages of Canal:

1. Parallel parses, especially when we use group event parse. (refers Logical instance parallel parses.)
2. Support different platforms.

Canal cannot apply parallelly (message subscribe should in order) which is the same as MySQL slave recover. Canal and MySQL

can learn from oracle logical standby database.

Advantages of Roma：

1. Storage is permanent which is good for subscribing.
2. Support logical topic which is good for combining operation (similar to partition merge in Kafka)