计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(6):60−68 [doi: 10.15888/j.cnki.csa.006374]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

MySQL 集群到 Oracle 数据库的数据同步方法①

杨明珉, 陈　勇

(中国移动 (深圳) 有限公司, 深圳 518048)
通讯作者: 杨明珉, E-mail: alexanderrein@126.com

摘　要: 随着开源技术的蓬勃发展, 开源数据库在很多业务场景中已经完成对商用数据库的替换, 其中 MySQL 数据库对于 Oracle 数据库的替换最为常见. 在分阶段替换的过程中往往存在 MySQL 数据库和 Oracle 数据库在同一套业务系统中并存, MySQL 数据库需要向 Oracle 数据库进行数据同步的情况. 同时, MySQL 数据库多节点高可用集群的部署方式进一步增加了数据同步的难度. 通过研究运用 Oracle GoldenGate 软件对 MySQL 集群进行数据抽取的方法, 设计出一套行之有效的方案, 实际解决从 MySQL 集群向 Oracle 数据库同步数据的难题, 同时保证了同步过程中数据库的一致性和完整性.

关键词: Oracle; MySQL 集群; GoldenGate; 异构数据库同步

引用格式: 杨明珉,陈勇.MySQL 集群到 Oracle 数据库的数据同步方法.计算机系统应用,2018,27(6):60–68. http://www.c-s-a.org.cn/1003-3254/6374.html

Data Synchronization Method for MySQL Cluster to Oracle Database

YANG Ming-Min, CHEN Yong

(China Mobile (ShenZhen) Limited, Shenzhen 518048, China)

Abstract: With the rapid development of open source technology, open source database has completed the replacement of commercial database in many business scenarios. The substitution of MySQL database for Oracle database is the most common. In the process of staged substitution, MySQL database and Oracle database often exist in the same set of business system, and MySQL database needs to synchronize data with Oracle database. At the same time, the deployment mode of multi node and high availability cluster in MySQL database further increases the difficulty of data synchronization. By researching the method of data extraction of MySQL cluster by using Oracle GoldenGate software, we design a set of effective schemes to solve the problem of synchronizing data from MySQL cluster to Oracle database, and ensure the consistency and integrity of database in the process of synchronization.

Key words: Oracle; MySQL cluster; GoldenGate; heterogeneous database synchronization

引言

从新兴的互联网行业到其他的传统行业, 使用开源的数据库替代传统商业数据库一直是 IT 系统建设的一个趋势. 以 MySQL 数据库为代表的开源数据库, 在多种业务场景中已经完成或者正在进行对以 Oracle 数据库为代表的大型商用数据库的替换. 然而, 在很多 IT 系统中, 由于应用程序的开发与数据库的紧密关联, 数据库的更替涉及到大量的应用程序改造和数据库迁移, 不可能做到一蹴而就. 这将存在开源数据库和原有商用数据库在同一套 IT 系统中长期并存, 数据相互耦合的情况. 其中, MySQL 数据库向 Oracle 数据库做数据同步是最为常见的需求之一. 异构数据库

① 收稿时间: 2017-09-19; 修改时间: 2017-10-25; 采用时间: 2017-10-31; csa 在线出版时间: 2018-05-28

之间的数据同步向来是 IT 系统工程实施的难点, 而在核心 IT 系统中, MySQL 数据库往往以多节点高可用集群的方式部署, 这又为其向 Oracle 数据库做数据同步进一步增加了难度.

本文采用在异构数据库同步中表现良好的 Oracle GoldenGate 作为 MySQL 数据库到 Oracle 数据库的数据同步工具, 意在研究设计出一套行之有效的方案, 实际解决从 MySQL 集群往以 Oracle 为代表的异构数据库同步数据的难题.

1 MySQL 集群与 Oracle GoldenGate 介绍

1.1 基于主从复制的 MySQL 集群简介

MySQL 集群架构有多种实现方式, 最为常见的包括基于主从复制、基于磁盘复制和基于网络存储引擎 (ndb) 的集群架构, 其中目前使用最为广泛的是基于主从复制的集群架构, 也是本文论述的 MySQL 集群架构.

主从复制技术在 MySQL 5.5 版本被引入, 可用于构建 MySQL 高可用架构方案, 其原理是通过把主数据库 (以下简称主库) 记录事务信息的二进制日志传递和应用到从数据库 (以下简称从库), 从而保证主库和从库的数据一致性. 主从复制的实现总体上分为 3 步:

1) 主库在完成每个事务的数据更新之前, 首先将相应的事务变更信息记录到二进制日志中, 所有的事务变更信息是以串行的方式写入二进制日志. 只有当每个事务变更信息被成功写入二进制日志后, 该事务的数据更新才能提交完成.

2) 从库会开启一个本地工作线程——I/O 线程. I/O 线程与主库建立一个客户端连接, 主库随即启动一个二进制抽取线程, 该线程从主库的二进制日志中依次读取事务变更信息并发送给从库的 I/O 线程, I/O 线程将收到的事务变更信息存储进本地的中继日志中.

3) 从库启动 SQL 线程, 该线程依次读取中继日志中的事务变更信息并解析, 然后在从库中重做执行这些事务, 完成与主库相同的数据更新[1–4].

主从复制的架构可以很灵活地实施, 根据不同的需求场景, 可以采用一主一从, 一主多从, 互为主从等架构. 当主库发生故障时, 从库可以代替主库接管业务, 成为新的主库, 并继续向其他从库进行数据同步, 从而最大程度保证数据一致性和服务高可用性. 在主从复制的基础之上部署一套主从复制的自动化管理程序, 即可组建一套 MySQL 高可用集群[5,6].

本文介绍的 MySQL 集群采用的是一主多从架构, 即集群中一个节点用作主库, 其他多个从节点用作从库. 集群管理程序是基于开源的 MHA(Master High Availability) 集群软件改进而成, 可以实现主从复制状态的实时监控, 秒级故障自动切换和自动主从提升等功能.

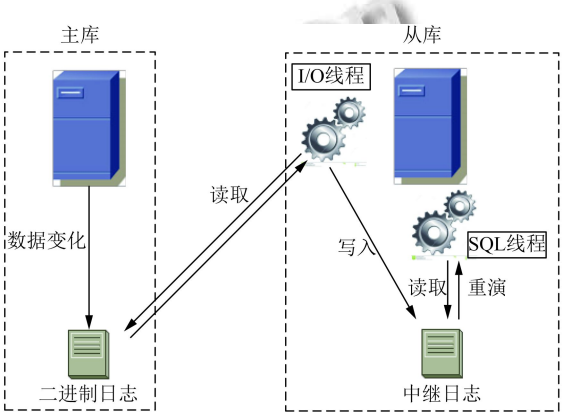

图 1 MySQL 主从复制架构示意图

1.2 Oracle GoldenGate 简介

GoldenGate, 即 Oracle GoldenGate Transactional Data Management, 是一种基于日志的数据同步软件. GoldenGate 从源数据库读取并解析在线日志或归档日志, 捕捉到数据变化, 再将这些数据变化投递到目标端, 在目标端将数据变化重新解析为 SQL 语句并应用到数据库, 从而完成源数据库到目标数据库的数据同步.

GoldenGate 的核心进程包括: 抽取进程、复制进程和管理进程.

1) 抽取进程 (Extract)

抽取进程运行在源端, 负责从源数据库的源数据表和日志中抽取数据和数据变化, 用于目标数据库的初始数据装载和变化数据的同步.

需要同步数据的对象写在配置文件中, 抽取进程读取配置文件获取同步的对象, 对这些对象进行的 DML (Data Manipulation Language) 和 DDL (Data Define Language) 操作, 都被抽取进程从日志中捕获, 并将其中完成提交的事务输出到 Trail 文件中以传送到目标端. 抽取进程周期性地完成检查点 (checkpoint), 记录其完成捕获的日志位置, 可以进行断点续传, 保证数据同步的完整性.

2) 复制进程 (Replicat)

复制进程在目标端运行, 该进程读取目标端接收到的 trail 文件, 并将文件内容解析为可以执行的 SQL 语句, 然后在目标数据库中执行这些 SQL 语句. 与 Extract 进程一样, 复制进程也是使用内部的检查点 (checkpoint) 机制来保证进程意外中断后, 可以从中断的位置重新启动恢复, 从而避免数据丢失的风险.

3) 管理进程 (MGR)

在 GoldenGate 的源端和目标端分别有一个控制进程, 称为管理进程. 该进程主要负责监控 GoldenGate 的其他进程状态, 发现进程异常, 发送超时或阀值告警, 报告进程错误, 为其他进程分配存储空间和重启异常进程等.

数据的变化信息存储在 trail 文件中, 目标端的复制进程通过读取该文件来获取需要应用到目标数据库的 SQL 语句. 从源端到目标端的数据传输, 大多情况下建议采用数据泵进程. 该进程在数据库源端启动, 负责把源端的 trail 文件以数据块级别通过网络传输到目标端. 运行在目标端的收集进程 (Collector) 把从远端传输过来的数据进行过滤, 映射和转换, 重新生成目标端的 trail 文件[7].

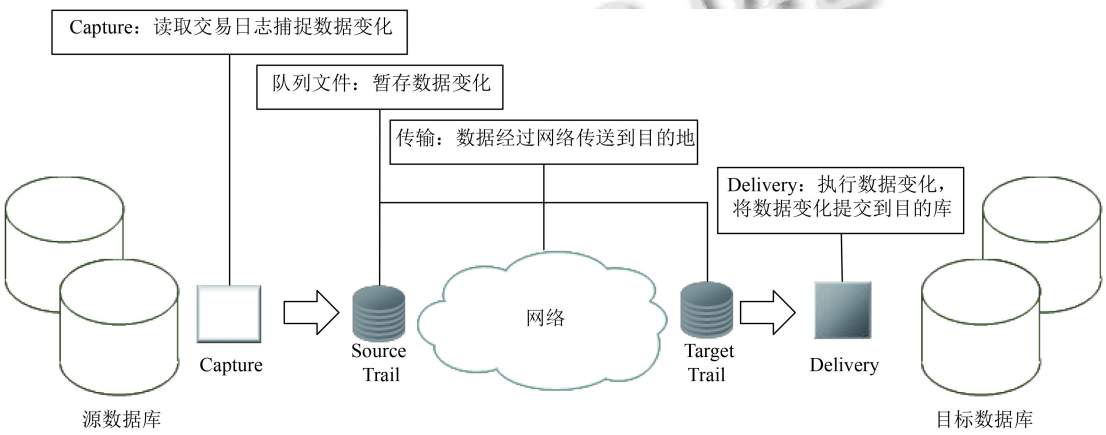

图 2 GoldenGate 工作原理示意图

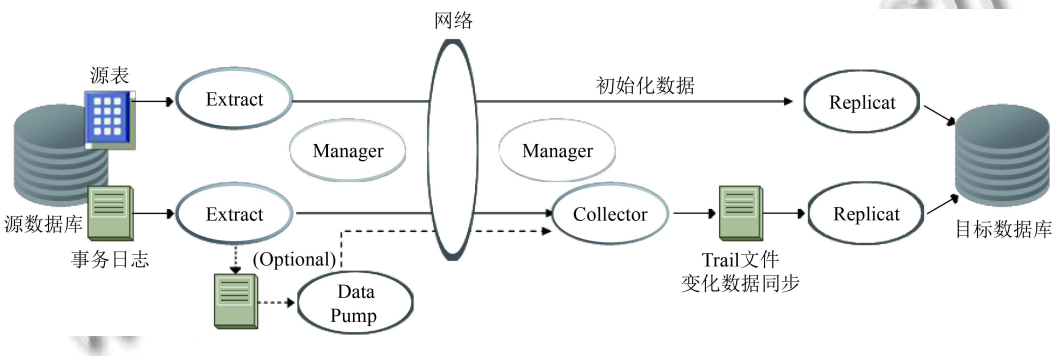

图 3 GoldenGate 关键进程示意图

GoldenGate 支持几乎所有主流操作系统和数据库的异构平台之间的实时复制. 此外, GoldenGate 复制的拓扑结构非常灵活, 除了支持一对一单向复制以外, 还支持双向复制、一对多复制、多对一复制、多级复制等. 因此, GoldenGate 可以应用于系统的应急容灾、业务双中心、数据库备份、数据库迁移、数据搬迁、异构数据库数据同步等多种应用场景[8,9].

2 MySQL 集群的 GoldenGate 复制架构设计

2.1 架构设计难点分析

在使用 GoldenGate 从 MySQL 到 Oracle 的异构数据同步架构中, MySQL 数据库作为复制的源端, Oracle 数据库作为复制的目标端, 整个复制过程涉及 3 个环节:

1) GoldenGate 的抽取进程从源端 MySQL 数据库

的二进制日志中抽取事务日志.

2) 抽取的事务日志通过网络传输到目标端 Oracle 数据库.

3) GoldenGate 的复制进程对接收到的事务日志进行解析并在目标端 Oracle 数据库内应用.

在这三个环节中, 环节 2) 事务日志的传输稳定性主要依赖于网络方案保障, 本文不做论述. 环节 3) 中由于 GoldenGate 与 Oracle 数据库具有原生兼容性, 搭配使用的方案非常成熟, 不是架构的难点, 本文也不做论述.

此架构的重点在于 GoldenGate 的抽取进程如何从 MySQL 数据库源端抽取事务日志. 当复制的源端 MySQL 数据库为单节点时, GoldenGate 的抽取进程直接从二进制日志中抽取事务日志即可. 如果在源端的抽取进程发生异常中断, GoldenGate 会记录下事务中断位置, 事务中断位置是一个二进制日志文件号对应的文件中所记录的标记该事务开始的位置号. 当源端的抽取进程恢复时, GoldenGate 会找到中断时记录的二进制日志文件及该文件中的事务位置号, 从这个位置继续开始事务日志的抽取, 从而保证不会有事务遗漏和事务被重复抽取的情况, 这样就能保证数据复制在事务日志抽取环节的完整性和一致性. 但是复制源端的 MySQL 数据库为多节点集群架构时, MySQL 集群的主库是复制的源端, GoldenGate 需要在主库所在服务器上启动, 并从主库的二进制日志中进行事务日志的抽取. 当集群发生切换的时候, 原有的从库之一接管主库的业务, 成为新的主库. 此时, 要保证业务数据继续往 Oracle 数据库做同步, 就需要 GoldenGate 在新的主库上继续抽取二进制日志, 即 GoldenGate 的复制源端需要从 MySQL 集群的原主库变为集群切换后的新主库, 且 GoldenGate 可以在新主库上完成断点续传. 这就需要在新主库的二进制日志中能够准确的找到抽取中断的位置, 确保从中断的准确位置开始抽取, 而不会抽取的位置靠后造成事务遗漏或者抽取的位置靠前造成事务被重复抽取, 因为无论事务遗漏还是事务重复都会造成最终的不一致. 要确保做到这一点就会面临以下两个难点:

1) 需要 GoldenGate 能够感知到 MySQL 集群发生的主从切换行为, 且能够在 MySQL 集群发生主从切换后即刻在原主库上停止抽取进程, 然后在新主库上启动抽取进程.

2) 在从新主库的二进制日志开始抽取事务日志之前, GoldenGate 如何找到准确的抽取起始位置. 因为 MySQL 原主库和新主库的二进制日志是两套完全独立的日志, 没有继承性, 在原主库和新主库中, 记录同一个事务的二进制日志文件号和文件内的事务 position 号完全不一样. 这会让 GoldenGate 通过原主库抽取中断时记录的二进制日志文件号和事务 position 号, 在新主库的二进制日志中确定准确的抽取起始位置变得非常困难.

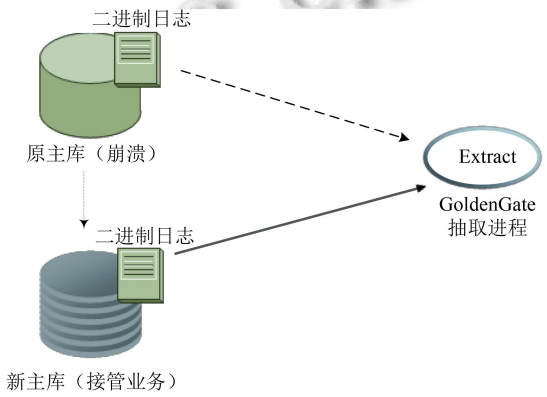

图 4 GoldenGate 日志抽取源端切换示意图

2.2 总体方案设计

根据 GoldenGate 要能完成集群切换后即刻在新的主库上启动需求, 本文设计了一套针对 GoldenGate 部署的共享存储方案:

首先, 在 MySQL 集群的各节点间部署一套共享存储, 该存储与集群各节点服务器均相连, 但该存储上的文件系统同一时刻只能挂载在一台服务器上, 没有挂载共享存储的服务器不能对该存储上的文件系统进行读写.

然后, 将 GoldenGate 的应用程序安装部署在共享存储的文件系统上, GoldenGate 的启动文件、配置文件和日志文件均放置在该文件系统上. 这样, 只要挂载了该共享文件系统的服务器, 就可以在其上启动 GoldenGate.

最后, 要保证 MySQL 集群发生切换的时候, 共享文件系统能够从切换前的主库服务器卸载下来, 重新挂载到切换后的新主库服务器上, 这一系列共享文件系统的操作均可以通过开发切换控制程序来实现.

这样就实现了 MySQL 集群原主库崩溃, 新主库接管业务后, GoldenGate 能够在新主库上启动并从新主库捕捉数据变化继续进行同步.

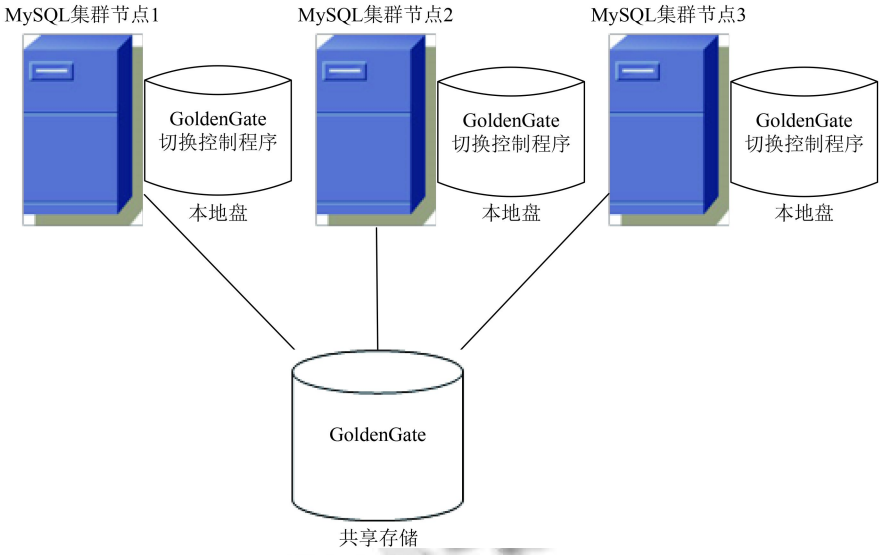

图 5 GoldenGate 共享存储部署方案

本文在方案中引用了 MySQL5.6 的新特性—GTID (Global Transaction ID), 来解决 GoldenGate 在新主库服务器上启动后, 如何找到准确抽取起始位置的难题. GTID 是在 MySQL 集群中的一个全局编号, 该编号对应于一个已提交事务, 同一个事务在 MySQL 集群中所有节点对应的 GTID 是完全一致的. 这样只要选择 MySQL 5.6 以上的版本, 在主从复制配置时开启 GTID 功能, GTID 就会显式输出到集群每个节点的二进制日志中. 那么在 MySQL 集群发生切换的时候, 根据原主库二进制日志抽取停止点的 GTID 找到新主库二进制日志上对应的 GTID, 就可以唯一定位到日志续传的起始点. 根据这个思路, 只要对 MySQL 集群管理程序进行改造, 完成如下功能即可:

1) 根据 GoldenGate 输出的原主库的二进制日志名称和日志位置号找到对应的 GTID.

2) 根据 GTID 定位出新主库对应的二进制日志名称和日志位置号, 并将定位结果输出.

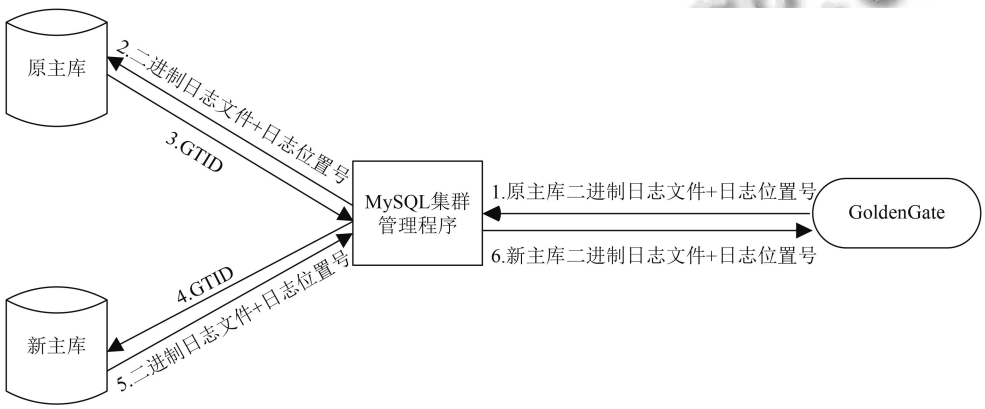

图 6 二进制日志断点位置定位流程

综合以上设计思路, 总体方案设计的关键点归纳如下:

1) 在集群各节点间部署共享存储, 将 GoldenGate 部署在共享存储的文件系统上.

2) 开发 GoldenGate 切换控制程序, 负责与 GoldenGate 的交互和共享存储在集群各节点间的切换.

3) 开启 MySQL 集群主从复制的 GTID 功能.

4) 改造 MySQL 集群管理程序, 新增与 GoldenGate

切换控制程序的交互接口, 能够发送切换通知, 并在接收到 GoldenGate 切换控制程序反馈的日志抽取停止点后, 将切换后日志续传起始点告知 GoldenGate 切换控制程序.

方案的总体工作流程设计如图 7.

当由于某个触发条件触发 MySQL 集群开始切换时, MySQL 集群管理程序同时出发 GoldenGate 日志抽取的切换流程:

1) MySQL 集群管理程序向 GoldenGate 切换控制程序发出切换通知, 并将切换目的地, 即新主库所在服务器的 IP 地址告诉 GoldenGate 切换控制程序.

2) GoldenGate 切换控制程序将部署着 GoldenGate 的共享存储切换到新主库所在服务器.

3) GoldenGate 切换控制程序获取 GoldenGate 抽取进程在 MySQL 集群切换前所抽取到原主库的最新的二进制文件名和日志位置号.

4) GoldenGate 切换控制程序把原主库的二进制文件名和日志位置号发送给 MySQL 集群管理程序, 请求返回新主库对应的二进制文件名和日志位置.

5) MySQL 集群管理程序根据原主库的二进制文件名和日志位置号定位出新主库对应的二进制文件名和日志位置, 将定位结果发送给 GoldenGate 切换控制程序.

6) GoldenGate 切换控制程序在新主库所在服务器启动 GoldenGate, 并指定 GoldenGate 切换控制程序从新主库对应的二进制文件名和日志位置开始抽取日志.

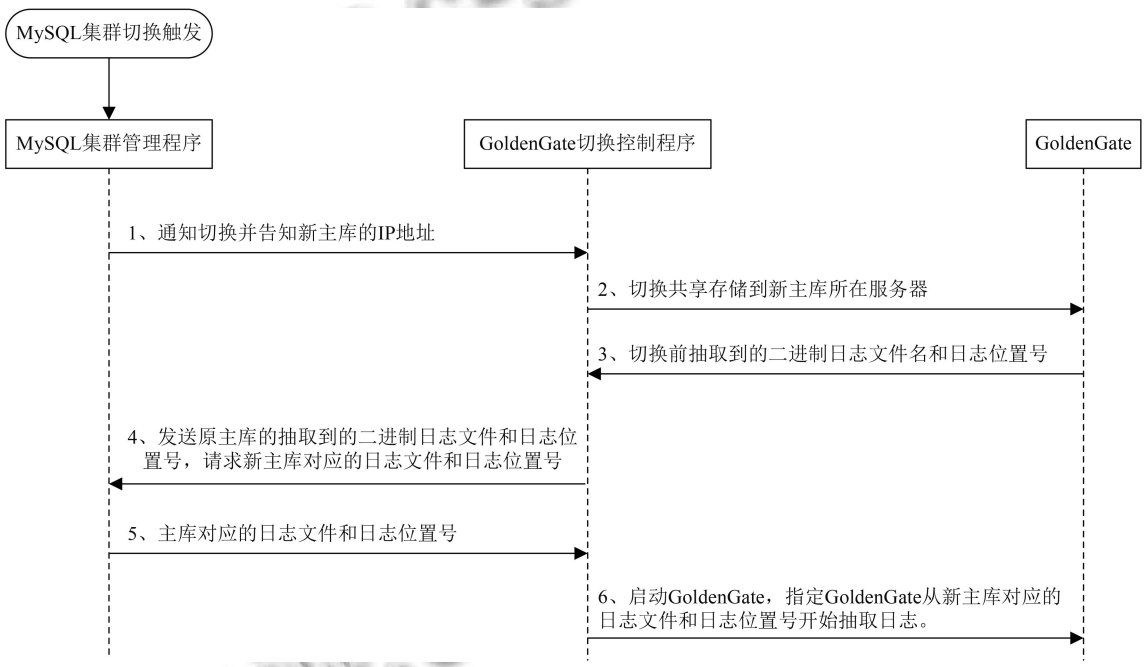

图 7 方案总体流程设计

3 方案实现

以上设计的同步方案的关键实现主要包括:

1) 对 MySQL 集群管理程序进行改造, 实现日志定位, 即二进制文件名、日志位置号与 GTID 的转换功能.

2) 实现 GoldenGate 切换控制程序的切换控制和断点续传功能.

3.1 MySQL 集群管理程序日志定位的功能实现

日志定位功能的核心实现思路是:

Step 1. 通过传入的二进制日志文件路径和位置号, 查找出该二进制日志文件中包含该位置号的事务, 再输出在该事物之前完成复制的最新事务的 GTID.

Step 2. 根据 Step 1 获取的 GTID, 遍历新主库的二进制日志文件, 查找出 GTID 所在的二进制日志文件和对应的已完成复制的事务, 再输出该事务之后第一个未完成复制的事务对应的位置号.

其中, Step 1 代码实现逻辑如下:

1) 读取二进制日志文件.

2) 读取文件头信息, 根据输入的位置号获取前一事务的事件位置.

3) 解析前一事务的事件内容, 获取该事务 GTID 的位置信息.

4) 从获取的 GTID 位置信息获取 GTID 字符串并输出.

关键步骤代码实现如下:

```
func getPreviousGtids(binlogPath string) (gtidDesc string, err error) {
    file, err := newBinlogReader(binlogPath)
    p := int64(4)
    headerBs := make([]byte, EVENT_HEADER_SIZE)
    payloadBs := make([]byte, 1024)
    length := binary.LittleEndian.Uint32(headerBs[9:13])
    eventType := int(headerBs[4])
    payloadLength := length – EVENT_HEADER_SIZE
    if payloadLength > uint32(len(payloadBs)) {
      payloadBs = make([]byte, payloadLength)
    }
    ret := ""
    sidNumberCount := bytesToUint(payloadBs[0:8])
    pos := 8
    for i := uint(0); i < sidNumberCount; i++ {
      if "" != ret {
        ret = ret + ", "
      }
      uuid := bytesToUuid(payloadBs[pos : pos+16])
      ret = ret + uuid
      internalCount := bytesToUint(payloadBs[pos+16 : pos+16+8])
      pos = pos + 16 + 8
      for i := uint(0); i < internalCount; i++ {
        from := bytesToUint64(payloadBs[pos:pos+8])
        to := bytesToUint64(payloadBs[pos+8:pos+16]) – 1
        pos = pos + 16
        ret = ret + ":" + strconv.FormatUint(from, 10) + "-" + strconv.FormatUint(to, 10)
      }
    }
    file.close()
    return gtidDesc, nil
}
```

Step 2 的代码逻辑与实现与 Step 1 类似, 在此不再做详述.

3.2 GoldenGate 切换控制程序的实现

GoldenGate 切换控制程序主要实现的核心功能是, 在接到 MySQL 集群的切换通知后, 完成 GoldenGate 的切换, 关键步骤如下:

1) 判断原主库服务器是否宕机. 如果没有宕机, 则在原主库服务器上停止 GoldenGate 复制, 然后触发 GoldenGate 切换, 如果已经宕机, 则直接触发 GoldenGate 切换.

2) 将部署着 GoldenGate 的共享存储切换到新主库所在服务器.

3) 通过 MySQL 集群将原主库二进制日志的位置转换为新主库二进制日志的位置, 并启动 GoldenGate 从新主库二进制日志的位置开始抓取.

这两个步骤实现的部分代码如下:

1) 将原主库的 IP 地址记录到配置文件 old_master_ip.conf 中, 在切换时需要先停止原主库服务器上的 GoldenGate 的情况下需要使用.

```
export CNT='df -h |grep"/ogg"| wc –l'
export HOSTIP=192.168.56.13
export SLAVE_IP1=192.168.56.15
export SLAVE_IP2=192.168.56.16
if [$CNT –eq 1] ;then
    sleep 1
    echo $HOSTIP > old_master_ip.conf
    sleep 1
    ssh $SLAVE_IP1 "cd /opt/ogg_switch_script; echo $HOSTIP > old_master_ip.conf"
    sleep 1
    ssh $SLAVE_IP2 "cd /opt/ogg_switch_script;echo $HOSTIP > old_master_ip.conf"
else
    exit
fi
```

2) 切换时判断原主库所在服务器是否已经宕机, 如果已经宕机不进行停止 GoldenGate 操作, 如果没有宕机, 则进行停止 GoldenGate 操作.

```
#** GG of source: stop process and umount /ogg **#
echo 'date "+%Y-%m-%d %H:%M:%S"" : Begin stop process and umount /ogg on "$OLD_MASTER_IP" ....." >> /opt/ogg_switch_script/switch_ogg.log
export CNO='ping $OLD_MASTER_IP –c4 | awk '/^4/{print $4}"
if [$CON-ge 1]; then
  ssh $OLD_MASTER_IP "cd $SWITCH_SCRIPT; sh/opt/ogg_switch_script/source_stop_process.sh > /opt/ogg_switch_script/source_stop_process. sh.out"
fi
sleep 5
```

3) 把共享盘从原主库服务器切换到新主库服务器.

如果原主库服务器没有宕机, 需要先卸载共享盘:

```
cd $SWITCH_SCRIPT
fuser –k/ogg
sleep 3
umount/ogg
```

在新主库服务器上挂载共享盘:

```
cd $SWITCH_SCRIPT
#vgchange-a y vgogg
mount/dev/mapper/vgogg–lvogg /ogg
sleep 2
```

4) 在新主库上获取原主库停止同步时的二进制日志位置, 然后通过 MySQL 集群接口转换, 找到新主库对应的二进制日志位置, GoldenGate 从新的二进制日志位置上启动进程进行同步.

```
export CNT='grep Position $LOG_INFO_EXTRACT | wc –l'
if [$CNT-ge 1]; then
    export EXT1_NAME='cat $LOG_INFO_EXTRACT| grep –i –E "Last Started|Log Number:" |sed –n 1p|awk '{print $2}"
    export EXT1_FILE='cat $LOG_INFO_EXTRACT| grep –i –E "Last Started|Log Number:" |sed –n 2p|awk '{print $3}'|sed 's/, //g"
    export EXT1_POS='cat $LOG_INFO_EXTRACT| grep –i –E "Last Started|Log Number:"|sed –n 2p|awk '{print $5}"
else
    export EXT1_NAME='cat $LOG_INFO_EXTRACT| grep –i –E "Last Started|Log Number:|Record Offset:" |sed –n 1p|awk '{print $2}"
    export EXT1_FILE='cat $LOG_INFO_EXTRACT| grep –i –E "Last Started|Log Number:|Record Offset:" |sed –n 2p|awk '{print $3}"
    export EXT1_POS='cat $LOG_INFO_EXTRACT| grep –i –E "Last Started|Log Number:|Record Offset:" |sed –n 3p|awk '{print $3}"
fi
cd $SWITCH_SCRIPT
export CN='ping $OLD_MASTER_IP –c4 | awk '/^4/{print $4}"
echo $CN > 1.txt
if [$CN-ge 1] ;then
    echo "r" >> 1.txt
    sh convert_binlog_file_pos.sh $OLD_MASTER_IP $EXT1_FILE $EXT1_POS $HOSTIP r |sed 's/:/ /g' > $LOG_EXT1_NEW_POSITION
else
    echo "local" >>1.txt
    sh convert_binlog_file_pos.sh $OLD_MASTER_IP $EXT1_FILE $EXT1_POS $HOSTIP local |sed 's/:/ /g' > $LOG_EXT1_NEW_POSITION
fi
```

3.3 效果验证

通过实际的部署使用, 在 MySQL 集群发生主从切换的情况下, GoldenGate 可以跟随主库的切换而自动进行故障切换, 完成抽取进程与新主库的自动对接, 并能够准确定位断点, 复制的数据无重复无丢失, 完美保障数据同步的连续性和一致性.

4 方案对比

使用 GoldenGate 进行 MySQL 数据库和 Oracle 数据库的异构数据库同步复制, 是行业内最常用和最成熟的方法. 多数情况下主要是针对 MySQL 数据库单实例和 Oracle 数据库进行同步, 当涉及 MySQL 集群环境和 Oracle 数据库进行同步的情况, 主要有两种部署

方案:

(1) GoldenGate 的抽取进程部署在 MySQL 集群的主库上. 缺点是一旦主库发生故障切换, GoldenGate 的抽取进程将无法继续工作, 需要人工介入处理.

(2) GoldenGate 的抽取进程部署在 MySQL 集群的从库上. 缺点是 MySQL 集群主库和从库数据存在一定的同步延迟, 导致 GoldenGate 的抽取进程抽取的数据也将与主库存在一定的延迟.

本文介绍的实现方法是在第一种部署方案的基础上进行了架构强化, 能够完美解决第一种部署方案的缺点, 在主库发生故障切换时, GoldenGate 的抽取进程能够自动的进行故障切换并继续无缝的完成事务抽取, 全自动无需人工干预. 表 1 为 3 种方案的对比结果.

表 1 方案对比矩阵

	第一种方案	第二种方案	本文方案
健壮性	低	低	高
数据丢失风险	高	低	无
复制延迟	低	高	低
人工参与度	高	低	无

5 结语

本文介绍了利用 GoldenGate 构建 MySQL 数据库到 Oracle 数据库的异构数据库同步复制架构, 实现了在 MySQL 集群发生主从切换时, GoldenGate 能够自动的进行故障切换并继续无缝的完成事务抽取, 保障了异构数据库间数据同步复制的连续性和可靠性. 同时, 本文介绍的复制架构不仅仅适用于 MySQL 数据库到 Oracle 数据库的异构数据库同步复制, 对于 MySQL 数据库到其他类型数据库的同步复制同样具有指导借鉴意义, 在整体架构不需要做大的改动的情况下, 只用对 GoldenGate 的复制端稍作调整, 就能满足 MySQL 数据库到其他类型数据库的同步复制需求.

参考文献

1 张伟丽, 江春华, 魏劲超. MySQL 复制技术的研究及应用. 计算机科学, 2012, 39(11A): 168–170.
2 邢志峰. MySQL 主从复制的研究与应用. 电子技术与软件工程, 2017, (15): 188.
3 刘腾. MySQL 复制技术的研究与改进[硕士学位论文]. 杭州: 浙江大学, 2011.
4 田关伟. MySQL 复制技术分析研究. 哈尔滨师范大学自然科学学报, 2015, 31(4): 45–48, 64.
5 韦一鸣. 基于 MySQL 复制技术的数据库集群研究[硕士学位论文]. 杭州: 杭州电子科技大学, 2014.
6 宣振国. 基于 Mysql 的数据库集群设计与实现[硕士学位论文]. 北京: 北京邮电大学, 2013.
7 曲波, 邓旭东, 姜锋. Oracle GoldenGate 数据同步机制研究与应用. 微型电脑应用, 2014, 30(6): 55–58.
8 曹建辉. 基于 GoldenGate 高级复制技术实现数据库同步. 甘肃科技, 2012, 28(24): 23–25. [doi: 10.3969/j.issn.1000-0952.2012.24.008]
9 王二暖. Goldengate 实现异构数据库间的数据同步. 电脑开发与应用, 2014, 27(5): 70–72.