软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(3):684–699 [doi: 10.13328/j.cnki.jos.005694]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

分布式异构数据库数据同步工具*

徐梓荐[1,2], 叶 盛[1,2], 张 孝[1,2]

[1](教育部数据工程与知识工程重点实验室(中国人民大学),北京 100872)

[2](中国人民大学 信息学院,北京 100872)

通讯作者: 张孝, E-mail: zhangxiao@ruc.edu.cn

摘 要: 一般而言,读写分离技术可以解决当前大数据环境下的读写速度失配的部分问题,但是现有的读写分离技术主要是针对同构数据库的解决方案.由于存储结构的不一致,由行式存储数据库和列式存储数据库构成的异构分布式数据库系统相较于同构分布式数据库系统在数据同步的过程中就会面临格式转换、同步速度不匹配等诸多难题.提出了基于 MySQL 二进制日志(Binlog)进行 SQL 还原的方法 TD-Reduction,设计并实现了 Binlog 解析器 BinParser 和 Binlog 还原器 BinReducer,它们可以基于 Mixed 格式的 Binlog,针对不同的事件(event)进行日志的解析,并依据相应的规则进行还原,生成可执行的 SQL 语句.综合以上技术,实现了一款分布式数据库数据同步工具 Cynomys.在实验环境中,Cynomys 表现出较好的性能.该方法适用于所有具有类似 Binlog 机制的其他异构数据库之间进行数据同步.

关键词: 数据同步;读写分离;SQL 还原

中图法分类号: TP311

中文引用格式: 徐梓荐,叶盛,张孝.分布式异构数据库数据同步工具.软件学报,2019,30(3):684–699. http://www.jos.org.cn/1000-9825/5694.htm

英文引用格式: Xu ZJ, Ye S, Zhang X. Data synchronization tool for distributed heterogeneous database. Ruan Jian Xue Bao/Journal of Software, 2019,30(3):684–699 (in Chinese). http://www.jos.org.cn/1000-9825/5694.htm

Data Synchronization Tool for Distributed Heterogeneous Database

XU Zi-Jian[1,2], YE Sheng[1,2], ZHANG Xiao[1,2]

[1](Key Laboratory of Data Engineering and Knowledge Engineering of the Ministry of Education (Renmin University of China), Beijing 100872, China)

[2](School of Information, Renmin University of China, Beijing 100872, China)

Abstract: In general, the read-write separation technology can solve some of the problems on mismatch between read and write in the current big data environment, but most of the current read-write separation technology are prepared for homogeneous database. Due to the inconsistent storage structure, heterogeneous distributed database systems composed of a row storage database and a columnar storage database will face many difficulties like format conversion and mismatch of synchronization speed in data synchronization compared to a homogeneous distributed database system. This study proposes the use of MySQL binary log to perform the TD-Reduction of SQL. It designs and implements Binlog parser BinParser and Binlog restorer BinReducer, which based on the mixed format. Different events perform log parsing and restoring according to the corresponding rules to generate executable SQL statements. Based on the above techniques, this study has implemented Cynomys, a distributed database data synchronization tool. In the experimental environment,

* 基金项目: 国家重点研发计划(2018YFB1004401); 国家自然科学基金(61732014); 北京市科技计划(Z171100005117002)

Foundation item: National Key Research and Development Program of China (2018YFB1004401); National Natural Science Foundation of China (61732014); Beijing Municipal Science and Technology Project (Z171100005117002)

本文由智能数据管理与分析技术专刊特约编辑樊文飞教授、王国仁教授、王朝坤副教授推荐.

收稿时间: 2018-07-20; 修改时间: 2018-09-20; 采用时间: 2018-11-01

Cynomys has shown sound performance. The method is suitable for data synchronization between all other heterogeneous databases with a similar mechanism like Binlog.

Key words: data synchronization; read/write separation; SQL reduction

关系型数据库的数据存储方式一般有行式存储和列式存储两种.对于行存储数据库,支持海量数据的高效更新,然而,当需要分析数据库中隐含的信息时,就可能会涉及一些针对数据库表中某一或某些属性上的聚合,而这是行式存储数据库的不足.列式存储数据库对于海量数据更新没有很高的处理效率,而对数据分析,特别是在某些属性上进行分析,往往有很好的查询优势.如果能够构建由行列数据库共同组成的分布式数据库系统,用行存储数据库接受来自业务线的数据,再通过数据同步将这些数据同步到列存储数据库中进行数据分析,就可以弥补数据库的上述不足,为真实的业务场景提供一个优化的解决方案.

本文以 MySQL 为对象,开展基于该数据库的数据同步技术研究工作,主要实现了一款基于数据库 Binlog 的分布式数据库中的数据同步工具 Cynomys.鉴于当前并没有一种针对行存储数据库到列存储数据库的实时同步工具,本文提出了一种基于 SQL 还原的实现方法,利用标准的 SQL 语句避免底层数据库引擎差异可能导致的无法进行数据库同步的问题.

本文的主要贡献包括:(1) 设计并实现了 Binlog 解析器 BinParser 以及还原器 BinReducer,解析器解析并获取 Binlog 的日志信息,还原器将 BinParser 解析的日志内容还原为可执行的 SQL 语句;(2) 提出了基于 BinParser 和 BinReducer 的异构分布式数据同步方法 TD-Reduction,并开发了基于 TD-Reduction 的数据同步工具 Cynomys;(3) 针对 ColumnStore 的存储结构提出了延迟提交(delay commit)的优化技术;(4) 本文针对 Cynomys 的性能、功能等方面做了大量实验,结果表明该工具有效且能够正常运行.

1 相关工作

对于分布式架构的数据库及其数据同步解决方案,研究人员提出了若干种体系架构以及基于此的各类问题的解决方案,如:Stonebraker 提出了一种广域的分布式数据库系统[1],Chen 等人提出了一种基于分布式数据库的在线数据分割方法[2],Google 开发了可扩展、多版本、全球分布式和同步复制的数据库 Spanner[3],Wang 等人提出了基于中间件的分布式数据同步技术[4].

在商用数据库中,Oracle 和 SAP HANA 都提出了解决异构数据存储的方案.其中,Oracle 在其 12c 版本中加入了解决异构数据存储问题的新特性 In-memory Option[5].在 12c 版本以前,Oracle 所有的数据都是以行形式进行存储的.该特性引入后,Oracle 允许用户将指定的表空间内所有的表以列形式存储在单独开辟的一块内存空间 In-memory Area 中[6],从而在查询分析数据时,对涉及某些属性上聚合的查询操作速度加快,并且这种双格式架构多占用的内存开销被控制在单一格式架构内存开销的 20%以内[7].SAP HANA 是混合型的内存数据库,它同时支持行存储以及列存储[8].与 Oracle 不同的是,它需要在创建新表时指定行式或列式存储,两种不同的存储架构使用不同的存储引擎.这两种商用数据库的方案是针对行列存储数据库有不同的优势所提出的数据存储方案,而 Cynomys 则是将行存储数据库中的数据实时同步或者迁移到列存储数据库中,与以上两种异构存储的方案定位不同.

当前,常规意义上的数据迁移已经发展得较为成熟,解决方案也已经较为明确,例如基于负载均衡的数据迁移方法[9].而数据同步可以看做是实时的数据迁移过程.一般的数据同步可以基于数据库日志[10,11]、触发器[12]等,其中,以日志法最具代表性和可行性.然而由于不同数据库的日志格式差别很大,可供日志分析的接口也有所不同,所以基于日志法的数据同步也局限于同种数据库,甚至要求版本相同或相近,这限制了数据同步技术的发展.有人曾提出使用 SQL 还原的方法[13],但该方法是基于 SQL Server 触发器实现的,人工干预程度大、维护困难.Cynomys 的基于标准 SQL 语句进行数据同步的方法受到该工作的启发.

在分布式环境下,许多数据库或分布式系统如 Google 的 Chubby,MegaStore,Spanner,Hadoop 中的 Zookeeper 等,都依赖 Paxos 算法[14]来保证不同结点之间事务的一致性.Cynomys 是一款数据同步工具,是从主数据库(行存

储数据库)到从数据库(列存储数据库)的数据同步,需要保证的是同步操作前后数据的一致性,而非同步过程中事务同步执行的一致性.

MySQL 提供了 Group Commit 机制(https://dev.mysql/com/doc/refman/8.0/en/commit.html),在 MySQL 的 InnoDB 引擎中,为了保证数据库的事务持久性不被破坏,在每个事务被提交之前都会先刷 redo 日志,而 redo 日志是要刷到磁盘上的,在多事务并发的情况下会造成性能的瓶颈.总的来说,Group Commit 是将多个并发的事务一次性写入日志来减少 I/O,而 Cynomys 的延迟提交机制是将一段时间内多个实现类似功能的事务合并在一起,以减少列存储数据库数据文件的访问,从而提升同步性能.

2 Cynomys 的关键技术

2.1 二进制日志Binlog

Binlog 的全称是 Binary Log(https://dev.mysql.com.doc/internals/en/binary-log-overview),即二进制日志文件,是备份的基本要素之一,对于基于时间点的恢复更是必须的.一般而言,日志会比数据小很多,更适合于需要频繁备份的场景.MySQL 的 Binlog 从记录格式上可以分成 3 种类型:Statement,Row,Mixed.其中,Mixed 格式默认以 Statement 格式作为记录格式,针对 Statement 格式不能准确描述的数据变化再以 Row 格式进行记录,综合了两者的优点,适于本文的场景,即:在不依赖 Master 节点具体环境的情况下使用 Statement 格式,在依赖 Master 节点具体环境的情况下使用 Row 格式,保证基于 Binlog 的业务可以快速准确地开展.

2.2 Binlog解析器BinParser

Binlog 提供的事件(event)类型复杂多样,对各种事件归纳出若干种有重要特征的基本事件,针对这些事件再进行下一步的处理.本文将 28 种事件类型归纳为 10 种基本事件类型,并为每种基本事件类型设置相应的属性,使得每种事件类型可以完整准确地表述自身的信息,10 种基本的事件类型及其描述见表 1.

Table 1 10 types of basic event

表 1 10 种基本的事件类型

Event 类型	描述
FormatDescriptionEvent	对 Binlog 信息作出总体的描述
RotateEvent	出现在单个 Binlog 文件过大,需要切换 Binlog 文件时
QueryEvent	在 Mixed 格式下,所有通过 Statement 记录的信息都会以该形式存在
XidEvent	意味着整个事务的提交
TableMapEvent	包含的字段信息的目的是准确匹配源表和目标表的信息
WriteRowsEvent	在源表和目标表的信息匹配后,记录详细的数据变化
UpdateRowsEvent	与 WriteRowsEvent 类似,只不过涉及的是 Rows 格式
DeleteRowsEvent	主要记录非 SQL 语句形式删除的数据信息
StopEvent	表示复制过程的结束
NopEvent	解析出现了异常,不能解析的 Event 都被归类为 NopEvent

基于上述对 Binlog 事件类型的综合归类,形成了本文的解析规则算法.由于事件种类繁多,处理过程中难免会出现过多的分支.为了避免这种情况,在设计算法时本文采用了数据驱动[15]的编程方式.BinParser 的算法伪代码见算法 1.

算法 1. 解析器(BinParser)算法.

输入:Binlog;

输出:*String log_info*.

begin

1. *event array*[][]={Four types of event};
2. *parseEvent*(*String event*);
3. *int type_num=event_array.length*

4. **for each** *type_num*
5. **if** *event.equals*(*event_array*)
6. **return** *event_array*[*i*].*parseFunc*;
7. **else**
8. **return** *null*;

end

2.3 Binlog还原器BinReducer

Cynomys 对 Binlog 的还原遵循最简原则.所谓最简原则,即对于 Mixed 格式中的 Statement 日志直接还原为原 SQL 语句,对于 Row 日志还原为能表述日志内容的最简单的 SQL 语句(如图 1 所示).

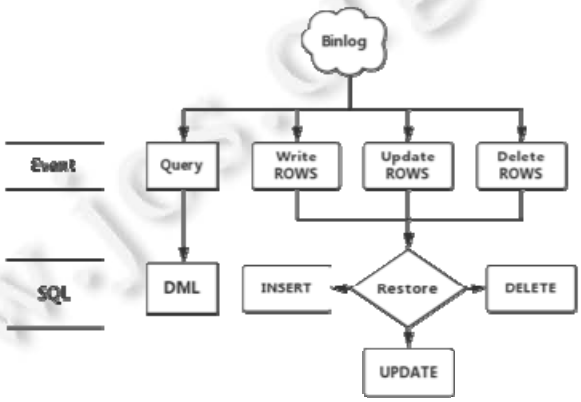

Fig.1 Flow chart of Binlog reduce

图 1 Binlog 还原示意图

日志还原器 BinReducer 的算法伪代码见算法 2.

算法 2. 还原器(BinReducer)算法.

输入:*String event*;

输出:*String Sql*.

begin

1. if *event_head=QueryEvent* then
2. Reduce Directly;
   /*如果包含 SQL 的字符串,直接还原 DML*/
3. else
4. get data with regular expression;
   /*如果不包含 SQL,则通过正则的方法提取出数据*/
5. switch *event_head*
6. Reduce INSERT UPDATE DELETE;

end

具体还原步骤如下.

1) Cynomys 监听 Master 节点的数据变化,并依据第 2.2 节的解析规则对 Binlog 信息进行实时解析;
2) 针对不同的事件类型分别进行还原;
3) 对于 QueryEvent,其中涵盖了具体执行的 SQL 语句,实际上是以 Statement 格式记录的日志信息,对此可以进行直接还原;

4) WriteRowsEvent,UpdateRowsEvent,DeleteRowsEvent 等几类事件,需要经过一定的处理,将其还原为 SQL 语句.

总的来说,根据最简原则来进行 SQL 还原的时候,如果事件信息中包含具体的 SQL 语句,那么直接提取出 SQL 语句即可,如下面这个例子.

日志示例 1

```
1 # at 277
2 # 071030 10::47:21 server id 3 end_log_pos 369 Query thread_id=13 exec_time=0 error_code=0
3 SET TIMESTAMP=1193755641/*!*/
4 insert into test(a) values(2)/*!
```

该事件中包含 SQL 语句,将该行 insert into test(a) values(2)直接提取出来即可(对应算法 2 的第 1 行、第 2 行).如果没有具体的 SQL 语句,那么根据事件的类型信息以及包含的具体数据,将 SQL 语句还原成 INSERT, UPDATE 或者 DELETE 等 DML 语句即可(对应算法 2 的第 4 行~第 6 行),举例如下.

日志示例 2

```
1 # at 107
2 # 071030 10::47:21 server id 3 end_log_pos 369 write_row thread_id=13 exec_time=0 error_code=0
3 SET TIMESTAMP=1193755641 TABLE_ID=43/*!*/
4 rows=[Rowcolumns=[1, c], Rowcolumns=[2, java], …]/*!*/;
```

该事件是 WRITE_ROWS_EVENT,但该日志内没有能够直接提取出来的 SQL 语句,这种情况则需要通过正则表达式提取出事件信息中数据的变化,还原成原始或等价的 SQL 语句.该事件能够还原出以下 SQL 语句:

```
insert into test values(1,‘c’), (2,’c’)
```

2.4 延迟提交

2.4.1 ColumnStore 存储模式

不同的数据库引擎有不同的存储模式,而根据存储模式的不同,对目标数据库的存储结构可以优化数据加载的策略.行存储主要应用在一些事务型数据库,如 Oracle,MySQL,PostgreSQL 等;而列存储主要应用在分析型数据库上,如 Mariadb ColumnStore,MonetDB[16],Infobright 等.本文要解决的主要问题是从 InnoDB 到 ColumnStore 引擎之间的数据同步,ColumnStore 的存储结构如图 2 所示(https://mariadb.com/kb/en/library/columnstore-storage-architecture/):

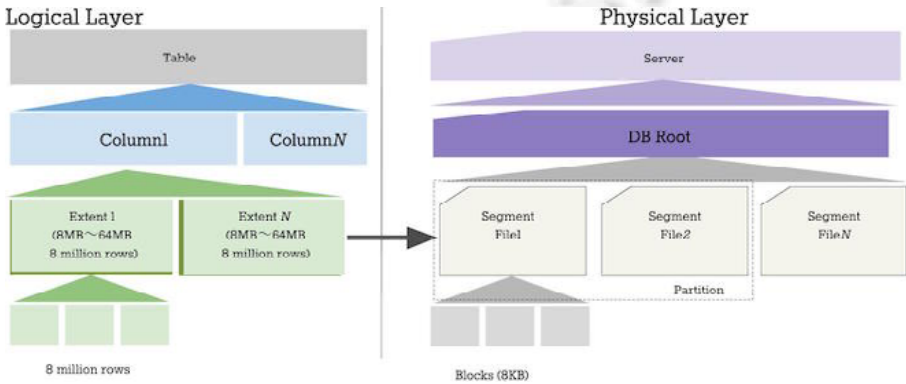

Fig.2 Structure of ColumnStore

图 2 ColumnStore 存储结构

2.4.2 Delay 思想

由于ColumnStore等列存储引擎的特殊存储架构设计,对于行存储架构支持完善的SQL语句就会带来同步

速度不匹配造成的性能瓶颈等问题.

假设存在表 Student(见表 2),并且需要执行以下插入语句:

INSERT INTO Student (StuId, Lastname, Firstname, Score) VALUES (4, Lucy, Alex, 100);

Table 2 Example of Student table

表 2 示例 Student 表

Student	Lastname	FirstName	Score
1	Tom	Edison	70
2	Jack	Lee	80
3	John	Cat	90

在行存储数据库中可以发现:插入一行新的数据十分简单,只要再添加一条完整的记录就可以,对于之前的记录毫无影响.Student 表信息在行存储数据库中存储,那么其逻辑结构就如图 3 所示的结构.而对于同样的 Student 表,如果存在列存储引擎中,那么其逻辑结构如图 4 所示.在执行例子中的 INSERT 语句时,将会涉及到表中的每一列数据,首先取出 StuId 所在的数据块,插入新数据“4”,然后再取出 Lastname 所在列,插入新数据“Lucy”,以此类推.而列存储引擎在通常情况下都会对数据进行压缩,每更新一次数据都会导致数据的解压缩与再压缩操作;同时,列存储数据库作为分析型数据库,通常每张表都有较多的字段.可想而知,通过常规插入手段的性能是极差的.而当用户基数大,用户操作零散,在一些峰值情况下会产生大量用户同时访问一张表的情况,如果以先来先服务的形式逐一满足用户的请求,尽管主库的数据一定会同步到备库中,备库的同步效率将会由于主库的零散插入行为而大幅降低.

StuId	Lastname	Firstname	Score
1	Tom	Edison	70
2	Jack	Lee	80
3	John	Cat	90

Fig.3 Example of row-based database

图 3 行存储数据库数据组织示例

StuId	Lastname	Firstname	Score
1	Tom	Edison	70
2	Jack	Lee	80
3	John	Cat	90

Fig.4 Example of column-based database

图 4 列存储数据库数据组织示例

图 5 是一般的同步处理过程,基本是一语句一提交.以这种方式进行事务提交,直接导致区间(extent)处在不断的解压缩与压缩的状态,降低了同步效率,在主库 InnoDB 达到查询峰值的时候,备库 ColumnStore 基本属于瘫痪状态.基于行列存储的差异,必须提出一种新的思路,以解决 SQL 语句在处理列存储数据性能不足的问题.

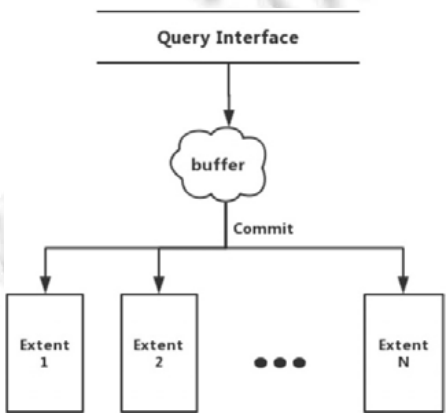

Fig.5 Flow chart of query processing

图 5 一般查询处理流程图

MySQL 的 Group Commit 目的是为了在并发的情况下通过将待 commit 的事务分组刷到 redo 日志中来减少 I/O,而延迟提交则是为了缓解行存引擎和列存引擎在数据插入性能上的差异.通过 binlog 来进行行列同步的过程中,如果不引入延迟提交机制,那么行存作为数据源产生 binlog 的速度会远远高于列存能读取 binlog 的速度.

假如将行存看做生产者,列存看做消费者,那么正常情况下,消费者是没有办法去完成消费生产者的内容的,这就导致消费者始终处于忙的状态,且与 binlog 的差距越来越远.延迟提交机制的实质是通过将大量的待提交的插入事务进行缓存,待到缓存区域达到一定的数据规模的时候再对这一批数据进行统一同步,通过一次解压操作,将各列待插入的数据追加到相应的 Extent 中去.

举例来说,假如行存有 3 个事务,都是往同一个表中插入数据.

```
INSERT INTO test VALUES(1,‘test1’);      //事务 T1
INSERT INTO test VALUES(2,‘test2’);      //事务 T2
INSERT INTO test VALUES(3,‘test3’);      //事务 T3
```

对于行存储数据库来说,进行这 3 个事务的提交很快就能完成.但是对于列存储数据库来说,提交 3 个事务远比提交 1 个事务的代价高得多,因为列存储数据库的存储结构与行存储数据库有本质区别.为了避免类似事务多次提交,延迟提交机制就是将一段时间内对同一表进行的操作进行归并.比如,例子中的 3 个 INSERT 语句等同于以下语句:

```
INSERT INTO test VALUES(1,‘test1’),(2,‘test2’),(3,‘test3’);
```

这样就可以减少对列存储数据文件的访问次数和解缩/压缩次数,提高列存吞吐量.

如图 6 所示,Query Interface 负责接收外界待插入的数据,在当前事务提交之前,Column Buffer 会将数据进行缓存,缓存的大小由用户根据实际需求决定.

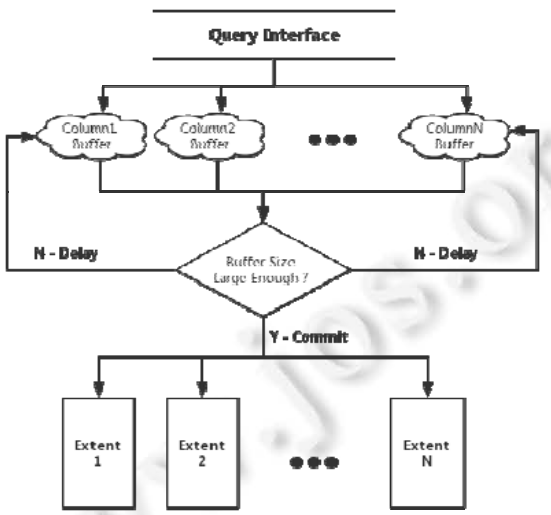

Fig.6 Flow chart of delay commit

图 6 延迟提交流程图

相较于图 4 所示的单语句提交的方式,延迟提交解决了列存储数据库由于存储模式限制导致插入性能较差、在交易高峰可能导致瘫痪的风险.以批量的形式进行语句提交,减少了存储引擎与磁盘的 I/O,也降低了 ColumnStore 本身的压缩与解压缩频率,满足了现实的交易场景.

2.5 数据同步正确性分析

对于数据同步工具来说,保证数据同步操作完成后目标数据库(列存储数据库)表数据与源数据表中数据的

一致性是最为重要的指标.下面简单分析数据同步的正确性.

数据同步分为两种情况.

- 第 1 种情况是只有一个用户(进程).从实现的角度看,Cynomys 实际上是设置了一个监听器(listener),实时地监控日志文件的变化,一旦日志文件新增了事件,监听器就会将这个事件捕捉并发送给解析程序 BinParser,然后执行后续的 BinReducer 等操作.如果用户的请求中包含对同一表的 INSERT 操作过多,Cynomys 会采用延迟提交机制,在不超过 Buffer Size 的情况下,将这些 INSERT 语句保存到缓冲区中一起执行.单用户情况下,数据同步的正确性实际依赖的是 BinParser 的解析以及 BinReducer 还原的 SQL 语句的正确性,只要上述步骤中的各个环节没有错误,最后还原的 SQL 是正确的语句,同步到从数据库中的数据就是正确的;
- 第 2 种情况是多用户(进程)并发的情况.多用户并发的情况下,不同的进程会同时向日志写入事件.Cynomys 对日志监听并还原 SQL 是一个串行的操作,即使多用户同时并发操作,解析事件的顺序与 Cynomys 监听到日志的顺序相同.即使并发的情况下,也不会影响数据同步的准确性.

算法 3. 延迟提交算法.

输入:*Buffer_Size*,*Query*;

输出:*null*.

begin

```
1.  set Buffer_Size
2.  while Pool_Size<Buffer_Size      /*当前缓冲区小于设定的缓冲区,缓存 Query*/
3.      Delay Query;
4.  Pool_Size++;
5.  commit Query;                    /*当前缓冲区满,批量提交 Query*/
6.  Pool_Size=0;                     /*重置 Pool_Size*/
```

end

3 Cynomys 整体架构及实现

3.1 总体架构

系统的架构总体分为日志解析、日志序列化、消息传递、SQL 执行等几个主要模块.

Cynomys 是一个典型的“生产者-消费者”模型,日志解析模块就是整个工具的生产者,消息传递模块充当缓冲区的作用,而最终的 SQL 执行模块则是最后的消费者,如图 7 所示.

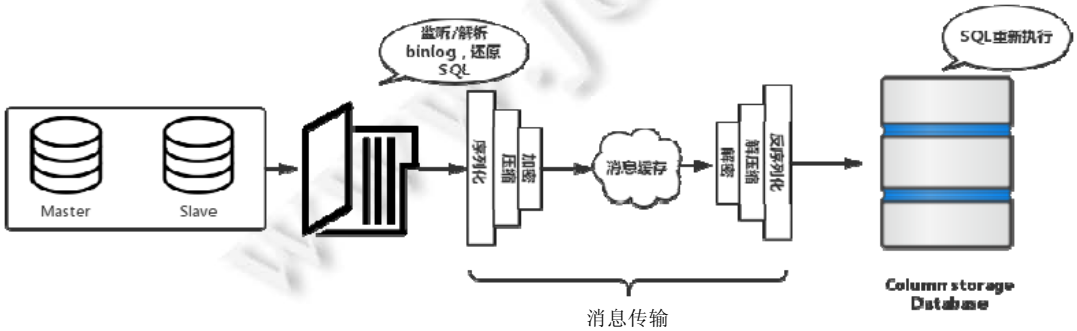

Fig.7 Whole architecture of Cynomys

图 7 Cynomys 整体架构图

日志解析模块的作用是读取并解析日志,将日志内容还原为 SQL 语句;序列化模块的作用是为还原出来的

SQL 语句提供序列化支持;数据压缩加密模块的目的是应对可能出现的带宽不足及网络攻击等问题;消息传递模块主要是为了使消息能够按照既定的顺序或者规则进行传递,保证下游的应用不会因为消息传递不合理导致的错误;SQL 执行模块的作用是将先前模块中得到的信息进行解压缩、反序列化等操作,将还原出来的 SQL 语句在目标数据库中执行,达到数据同步的效果.

3.2 日志解析模块

在第 2.1 节中已经提到 MySQL 的 Binlog 属于二进制文件,本身并不具有可读性,因此对基于 Binglog 的异构数据库同步,需先将 Binlog 中的内容解析成目标数据库可执行的相关内容.

首先从日志解析模块就进行事件的分类处理,Binlog 解析器的一级子模块可以分为 RotateEventParser, QueryEventParser 和 WriteEventParser 这 3 个部分,分别解析更换 binlog 文件事件、解析 statement 的事件以及数据更新相关的事件.根据解析出来的事件类型,EventFilter 会将所有的事件分成更详细具体的事件类别, EventListener 监听器会实时监听 MySQL-Master 节点的日志内容变化并通知业务线.具体解析流程如图 8 所示.

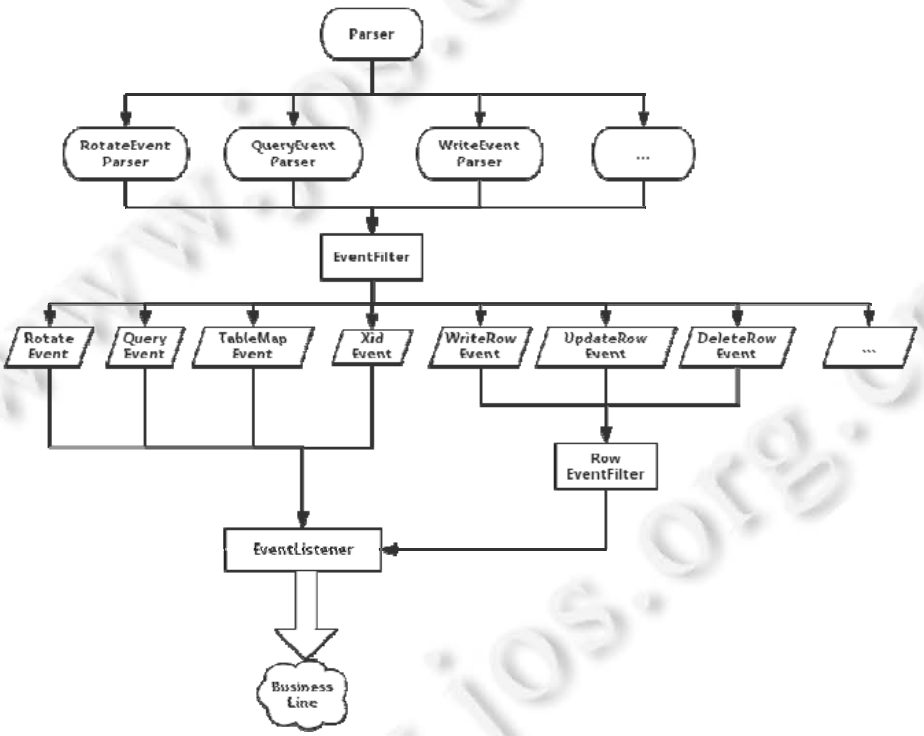

Fig.8 Flow chart of Binlog parser

图 8 Binlog 解析器解析流程图

3.3 序列化模块

数据或者对象状态在分布式系统中的传输不可能直接以字符串的形式存在,必须根据一定的模式对数据进行序列化[17].

支持序列化的工具或方法有很多,例如 Avro,Protocol Buffer,JSON 等.Cynomys 中选用 Apache Avro 作为序列化工具,主要是由于 Avro 具有支持多样数据结构、可持久化、支持 RPC(远程过程调用)、能够与动态语言简单结合等特点.JSON 在 Avro 中用来定义序列化模式文件,使用 Avro 的关键在于定义消息的模式,只要定义好消息模式并编译保存,Avro 就会根据定义的模式进行相应的序列化和反序列化.

在 Cynomys 中,定义一个 JSON 格式的模式如下所示.

Cynomys 的 JSON 定义格式

```
{
  “namespace”: “com.cynomys.code.serialization”,
  “type”: “record”,
  “name”: “Message”,
  “fields”: [
    {
      “name”: “Ti”,
      “type”: “string”
    },
    {
      “name”: “Database_name”,
      “type”: “int”
    },
    {
      “name”: “Table_name”,
      “type”: “int”
    },
    {
      “name”: “SQL”,
      “type”: “string”
    }
  ]
}
```

在 JSON 定义的模式中,分别定义了一条消息的几个域,为简洁明了表述消息内容,选择事务 ID(tid)、涉及的数据库名(database_name)、涉及的表名(table_name)以及还原出来的 SQL 语句(SQL).经过序列化模块产生的序列化文件会被传送到下一个处理模块进行进一步处理,序列化具体的流程如图 9 所示.

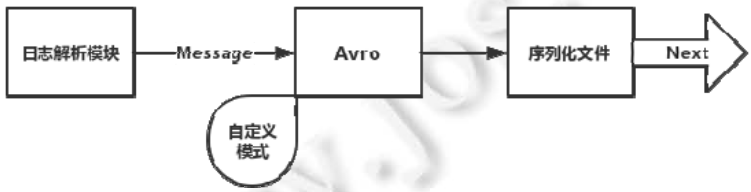

Fig.9 Flow chart of serialization based on Avro

图 9 基于 Avro 的序列化操作流程图

3.4 其他模块

除前面提到的日志解析模块和序列化模块外,Cynomys 还包含数据压缩加密、消息传递、SQL 执行等模块.在数据压缩加密模块中,本文所采用的 GZip 数据压缩算法[18],用 AES 算法对压缩包进行加密,RSA 算法对 AES 算法的密钥进行加密,与被加密的压缩包一起发送到消息传递模块.数据压缩加密模块的流程如图 10 所示.在消息传递模块中,本文选择 Redis 作为消息队列,基于 Redis 实现消息的发布/订阅(简称 Pub/Sub)模式[19].利用发布/订阅模式,解决了被压缩加密的消息包可能在网络传输过程中丢失或者消息发送方和接收方处理效率不一致带来的延迟等问题.

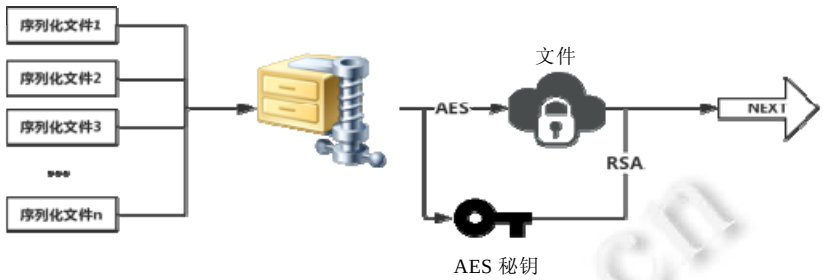

Fig.10 Flow chart of compression and encryption

图 10 数据压缩加密模块流程图

来自消息队列中的消息,需由消费模块对消息进行解密、解压缩并反序列化后使用消息.Cynomys 不会直接执行消息中的 SQL 语句,而是根据情况决定是否启用延迟提交机制.

4 实验评估与分析

本节内容旨在检验 Cynomys 的可用性、高效性、兼容性和正确性.在可用性实验中,主要验证 Cynomys 对数据库基本操作的支持度以及 Cynomys 本身的稳定性;在高效性实验中,主要验证了延迟提交机制下 Cynomys 的 INSERT 性能;在兼容性实验中,主要验证了 Cynomys 的数据库兼容性和运行平台的兼容性;在正确性实验中,主要验证了多个事务并发情况下 Cynomys 同步数据的正确性.

4.1 环境配置

Cynomys 的功能实验、性能实验在两台服务器上运行,其中,主服务器运行 MySQL 等行存储数据库,备份服务器运行 MariaDB ColumnStore 等列存储数据库.所有实验的测试数据均采用 TPC-C 或 TPC-H 基准测试数据集.具体的实验环境参数见表 3.

Table 3 Configurition of experiment

表 3 实验环境配置

参数设置	具体信息
主机 CPU	Intel(R) Xeon(R) CPU E5-4607
主机内存	8GB
操作系统	Linux CentOS 7.4
网络	千兆以太网
MySQL	V5.7
ColumnStore	V1.1.2
JDK	1.8.0_101

4.2 功能实验

4.2.1 数据类型支持度

数据库基本数据类型是支撑表元素的基础,Cynomys 对数据类型的解析是根据 MySQL 写日志的规则,不同的数据类型对应不同的 ID,对字符串类型和非字符串类型分别解析.本文对 MySQL 的各个基本数据类型都进行了相应的测试.实验结果表明:Cynomys 能够完全支持 MySQL 的基本数据类型(MySQL 的基本数据类型:https://dev.mysql.com/doc/refman/8.0/en/data-type-overview.html),MariaDB,MonetDB 的基本数据类型和 MySQL 一致.

4.2.2 操作类型支持度

本文用例中涉及的 SQL 语句大都与具体的数据直接相关,所以在实验论证中只对 DDL 语句和 DML 语句进行实验验证证,而对 DCL 语句不做支持.实验结果见表 4,可以看出,Cynomys 能够支持所有 DDL 语句和 DML 语句.

Table 4 SQL statement support in Cynomys

表 4 Cynomys 对 SQL 语句类型支持情况

语句类型		支持情况
DDL	CREATE	支持
	ALTER	支持
	DROP	支持
	TRUNCATE	支持
	COMMENT	支持
DML	SELECT	支持
	INSERT	支持
	UPDATE	支持
	DELETE	支持

4.2.3 系统稳定性

作为一个实时同步工具,对其稳定性要求一般较高,必须保证 7×24 小时无间断的稳定运行和快速响应.本实验是将 Cynomys 部署在服务器上连续运行一周,在每天的 4 个时间点各发起一次同步请求,观察 Cynomys 的表现并进行采样,得出结论,Cynomys 的稳定性达到了 7×24 小时及时响应,符合 MySQL 的运行状态和同步要求,实验结果见表 5.

Table 5 Test table of operational stability

表 5 Cynomys 运行稳定性测试表

	00:00	6:00	12:00	18:00
第 1 天	正常	正常	正常	正常
第 2 天	正常	正常	正常	正常
第 3 天	正常	正常	正常	正常
第 4 天	正常	正常	正常	正常
第 5 天	正常	正常	正常	正常
第 6 天	正常	正常	正常	正常
第 7 天	正常	正常	正常	正常

4.3 性能实验

性能实验中,主要测试从行存储到列存储的同步效率,其中,列存储数据库选择前面提到的 MariaDB ColumnStore.测试集选用的是 TPC-C 产生的模拟数据.

MariaDB ColumnStore 主要用于分析型数据库使用,所以 Cynomys 的同步只涉及 INSERT,对 UPDATE 和 DELETE 性能不作要求,在性能实验中,主要是要检验 Cynomys 执行 INSERT 的性能.

在测试 Cynomys 的 INSERT 性能时将启用延迟提交功能,通过调整延迟提交的缓冲区大小,得到实验数据如图 11 所示.

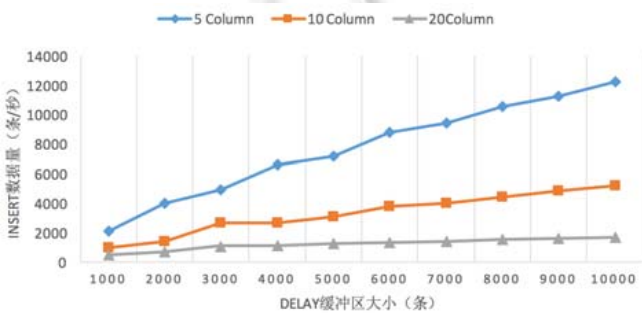

Fig.11 Figure of DELAY buffer and INSERT effiency

图 11 DELAY 缓冲区大小与 INSERT 效率关系表

依据上述实验结论,延迟提交确实可以提高 INSERT 操作的同步性能.在内存容量充足的情况下,缓存的待插入数据越大,相应的同步效率越高.

根据 MySQL 的现实场景,一般峰值的 INSERT 量在 1 000 条/s 左右.由于表的列数对插入性能影响较大,所以对于延迟提交缓冲区大小的需要可依据表结构来确定.以一个拥有 20 个字段的表为例,满足峰值 INSERT 的缓冲区大小设置为 3 000 左右即可.

从数据中还能看出另一个问题:随着表的列数增加,INSERT 操作效率的提升降低了.这是由于受测列式数据库的特殊存储结构,列式数据库中每一列都对应一个文件,这样的文件组织结构导致当表中的列增多时,会对性能有较大的影响.

4.4 兼容性实验

兼容性实验主要测试 Cynomys 在不同操作系统上是否能正常运行以及不同行列数据库能否正常同步.

Cynomys 采用 JAVA 语言开发完成,目的就是为了能兼容各个操作平台.本节选取了当前主流的操作系统包括 Windows 7,Windows 8,Ubuntu,CentOS 等不同版本测试 Cynomys 在各个操作系统平台的运行情况,实验结果见表 6.

Table 6 Experimental result of compatibility of Cynomys

表 6 Cynomys 平台兼容性实验结果

操作系统	运行情况
Windows 7 32bit	正常
Windows 7 64bit	正常
Windows 8 64bit	正常
Ubuntu 14.04 LTS 64bit	正常
Ubuntu 16.04 LTS 64bit	正常
CentOS 6.5 64bit	正常
CentOS 7.2 64bit	正常

从表 6 可以看出,Cynomys 在当前主流操作系统上都能正常运行.此外,还对不同版本的行列数据库进行了软件兼容性实验.Cynomys 是基于 Binlog 实现的,因此所有基于 Binlog 且能执行标准 SQL 语句的行数据库理论上都能运行,本节测试了 MySQL 与 MariaDB(ColumnStore)的不同版本进行软件兼容性实验,实验结果见表 7.

Table 7 Experimental result of compatibility of database in Cynomys

表 7 Cynomys 数据库兼容性实验结果

MySQL 版本(InnoDB)	MariaDB 版本(ColumnStore)	同步情况
MySQL 5.5	ColumnStore 1.1.0	正常
MySQL 5.5	ColumnStore 1.1.1	正常
MySQL 5.5	ColumnStore 1.1.2	正常
MySQL 5.6	ColumnStore 1.1.0	正常
MySQL 5.6	ColumnStore 1.1.1	正常
MySQL 5.6	ColumnStore 1.1.2	正常
MySQL 5.7	ColumnStore 1.1.0	正常
MySQL 5.7	ColumnStore 1.1.1	正常
MySQL 5.7	ColumnStore 1.1.2	正常

除表 7 给出的 MySQL 和 MariaDB 外,还对其他列存储数据库如 MonetDB 进行了测试,在这些行存储数据库之间,Cynomys 都能正常同步数据,不同列存储数据库对数据查询时间的影响如图 12(以 MariaDB 和 MonetDB 进行对比)所示.

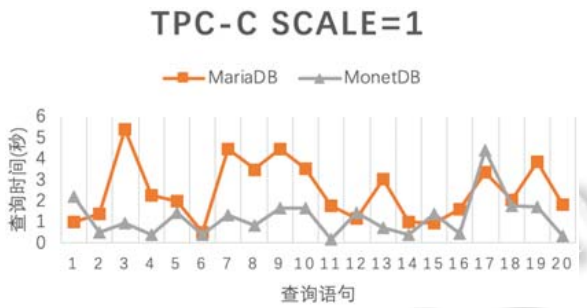

Fig.12 Influence of query time for different column-based database

图 12 不同列存储数据库对查询时间的影响

4.5 正确性实验

正确性实验主要测试在多用户(进程)并发的情况下,Cynomys 同步数据的准确性与源数据是否一致.本节实验所选取的主数据库为 MySQL5.6,列数据库为 MonetDB11.27,数据集为 TPCH-DBGEN 生成的 TPC-H 数据中的 LINEITEM 表的前 100 万条数据.

实验假设有 4 个用户同时对 MySQL 写入共 100 万条数据,每个用户写入 25 万条.从 MySQL 写入数据一共用时 5.68s,由于 Cynomys 对列存储数据库 SQL 语句提交的优化,数据写入 MonetDB 的速度与 MySQL 写入的速度几乎是同步的,仅用 6.12s.实验为采样实验,测试采样为 30%,50%,100%的情况下,数据响应速度与数据同步的准确率.对比是否与原数据相同采用将所有字段进行拼接,比较与原数据的字符串是否相同的方法.同步实验准确性结果如图 13 所示.

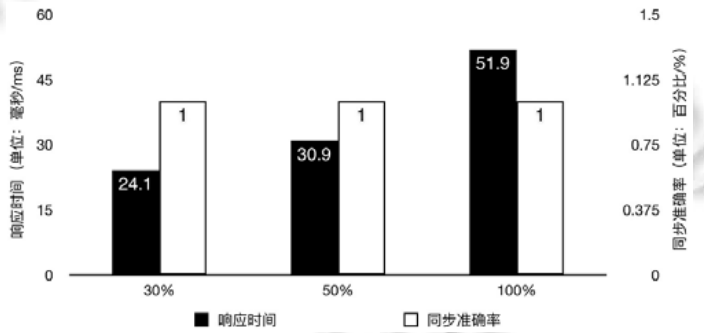

Fig.13 Accuracy of synchronization

图 13 同步准确性实验

实验结果显示,在不同采样大小的情况下,Cynomys 同步到从数据库的数据都能与源数据保证完全一致的准确性,验证了第 2.5 节中对数据同步正确性的说明分析.

5 总 结

在当前分布式系统得到广泛应用的背景下,本文提出了一种利用数据库所提供的 Binlog 机制,设计了日志解析器 BinParser 和日志还原器 BinReducer.为了应对分布式场景,也为工具实现了包括序列化、压缩加密、消息分发等一系列的工作.同时,针对列存储数据库本身在 INSERT 性能上存在的不足,提出了延迟提交的思想.基于以上各项技术实现了同步工具 Cynomys,用于同步行存储数据库和列存储数据库之间的数据.该方法弥补了不同存储模式的异构数据库之间无法进行实时数据同步的空白,解决了从分布式行存储数据库到列存储数据库的同步问题,且在实验中表现出了良好的性能.

此外,目前 Cynomys 只支持部分版本的 MySQL 数据库,在 MySQL5.6 之后,由于 Binlog 引入了 CRC 校验,Cynomys 对于如何适应 CRC 校验并没有给出解决方案,所以对于最新版本的 MySQL 数据库使用场景还不能做到同步,需要人为地对高版本 MySQL 进行配置才可以.这也是未来需要改进的方面.

References:

[1] Stonebraker M, Aoki PM, Litwin W, Pfeffer A, Sah A, Sidell J, *et al*. Mariposa: A wide-area distributed database system. VLDB Journal, 1996,5(1):48−63.

[2] Chen K, Zhou Y, Cao Y. Online data partitioning in distributed database systems. In: Proc. of the Int'l Conf. on Extending Database Technology (EDBT 2015). 2015. 1−12.

[3] Corbett JC, Dean J, Epstein M, *et al*. Spanner: Google's globally-distributed database. In: Proc. of the Usenix Conf. on Operating Systems Design and Implementation, Vol.31. 2012. 251−264.

[4] Wang J, Zhang DS. Research and design of distributed database synchronization system based on middleware. In: Proc. of the Modern Electronics Technique. 2016. 685−688.

[5] Lahiri T, Chavan S, Colgan M, *et al*. Oracle database in-memory: A dual format in-memory database. In: Proc. of the IEEE, Int'l Conf. on Data Engineering. 2016. 1253−1258.

[6] Mukherjee N, Chavan S, Colgan M, *et al*. Distributed architecture of oracle database in-memory. Proc. of the VLDB Endowment, 2015,8(12):1630−1641.

[7] Mukherjee N, Kulkarni K, Jin H, *et al*. How does oracle database in-memory scale out? In: Proc. of the Int'l Joint Conf. on Software Technologies, Vol.1. 2015. 1−6.

[8] Färber F, May N, Lehner W, *et al*. The sap hana database—An architecture overview. Bulletin of the Technical Committee on Data Engineering, 2012,35(1):28−33.

[9] Wang Z. Research and implementation of load balancing algorithm for offline data migration [MS. Thesis]. Shenyang: Northeastrn University, 2015 (in Chinese with English abstract).

[10] Li GX, Liu S, Liu JC, *et al*. Research and application of data synchronization service platform based on achived logs. Electric Power Information and Communication Technology, 2010,8(2):31−35 (in Chinese with English abstract).

[11] Song FL. The research and implementation of massive data synchronization system for database based on log parser [MS. Thesis]. Guangzhou: South China University of Technology, 2016 (in Chinese with English abstract).

[12] Lin Y, Chen ZB. Implementation of synchronization system for distributed database. Computer Engineering and Design, 2010, 31(24):5278−5281 (in Chinese with English abstract).

[13] Zheng HM. Research and implementation of heterogeneous database synchronization technology based on SQL restore method. Computer Era, 2008(10):15−18 (in Chinese with English abstract).

[14] Prisco RD, Lampson B, Lynch N. Revisiting the Paxos algorithm. In: Proc. of the Int'l Workshop on Distributed Algorithms, Vol.243. 1997. 111−125.

[15] Xu JX, Hou ZS. Notes on data-driven system approaches. Acta Automatica Sinica, 2009,35(6):668−675.

[16] Boncz PA, Zukowski M, *et al*. MonetDB/X100: Hyper-pipelining query execution. In: Proc. of the Int'l Conf. on Innovation Database Research (CIDR), Vol.5. 2005. 225−237.

[17] Bouchenak S, Hagimont D, Palma ND. Techniques for implementing efficient Java thread serialization. In: Proc. of the ACS/IEEE Int'l Conf. on Computer Systems and Applications, Vol.34. 2003. 355−393.

[18] Zeng CH, Zhang JJ, Xiong SF. Design and implementation of P2P remote assistance system based on JXTA. Journal of Jiangxi University of Science and Technology, 2009,30(3):36−40 (in Chinese with English abstract).

[19] Gutierrez F. Messaging with Redis. Berkeley, Apress, 2017. 120−155.

附中文参考文献:

[9] 王智.负载均衡的离线数据迁移算法的研究与实现[硕士学位论文].沈阳:东北大学,2015.

[10] 李功新,刘升,刘金长,等.基于归档日志的数据同步服务平台研究与应用.电力信息与通信技术,2010,8(2):31−35.

[11] 宋芳利.基于日志解析的数据库海量数据同步系统的研究与实现[硕士学位论文].广州:华南理工大学,2016.

[12] 林源,陈志泊.分布式异构数据库同步系统的研究与应用.计算机工程与设计,2010,31(24):5278−5281.

[13] 郑海明.基于 SQL 还原法的异构数据库同步技术的研究与实现.计算机时代,2008(10):15−18.

[18] 曾传璜,张晶晶,熊圣芬.基于 JXTA 的 P2P 远程协助系统的设计与实现.江西理工大学学报,2009,30(3):36−40.

徐梓荐(1994－),男,硕士,主要研究领域为关系型数据库,数据同步.

叶盛(1993－),男,硕士,主要研究领域为数据迁移,数据同步.

张孝(1972－),男,博士,副教授,CCF 高级会员,主要研究领域为数据库,大数据.