ODF 2017 开源数据库论坛(北京)
OPEN-SOURCE DATABASE FORUM(BEIJING)

开源数据库正在改变世界

2017年8月24日-25日　北京－京仪大酒店

TDSQL分布式事务原理剖析

赵伟 腾讯TDSQL技术专家

目录

CONTENTS

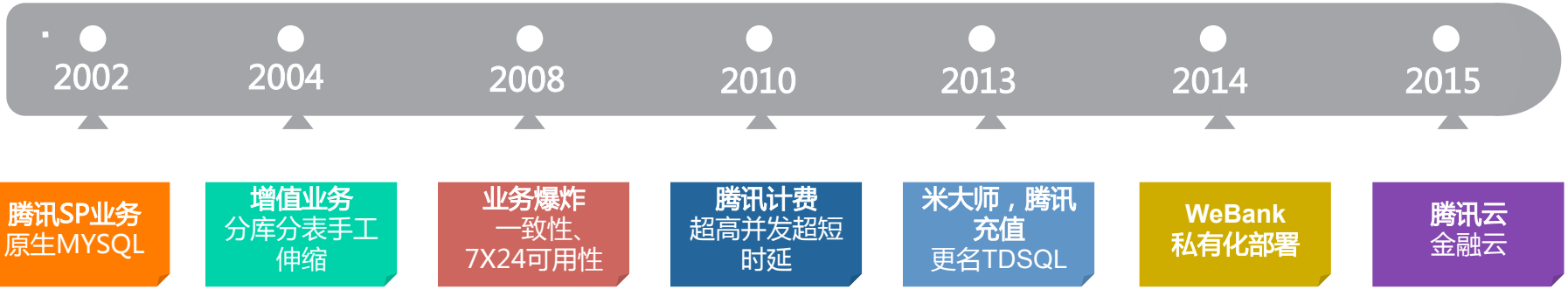
TDSQL简介

腾讯金融级分布式数据库系统

金融级云数据库解决方案（CDB for TDSQL）

面向金融类业务，十年积累，亿级账户验证

腾讯公司内与计费、充值、转账、财务等核心系统90%以上都使用TDSQL！

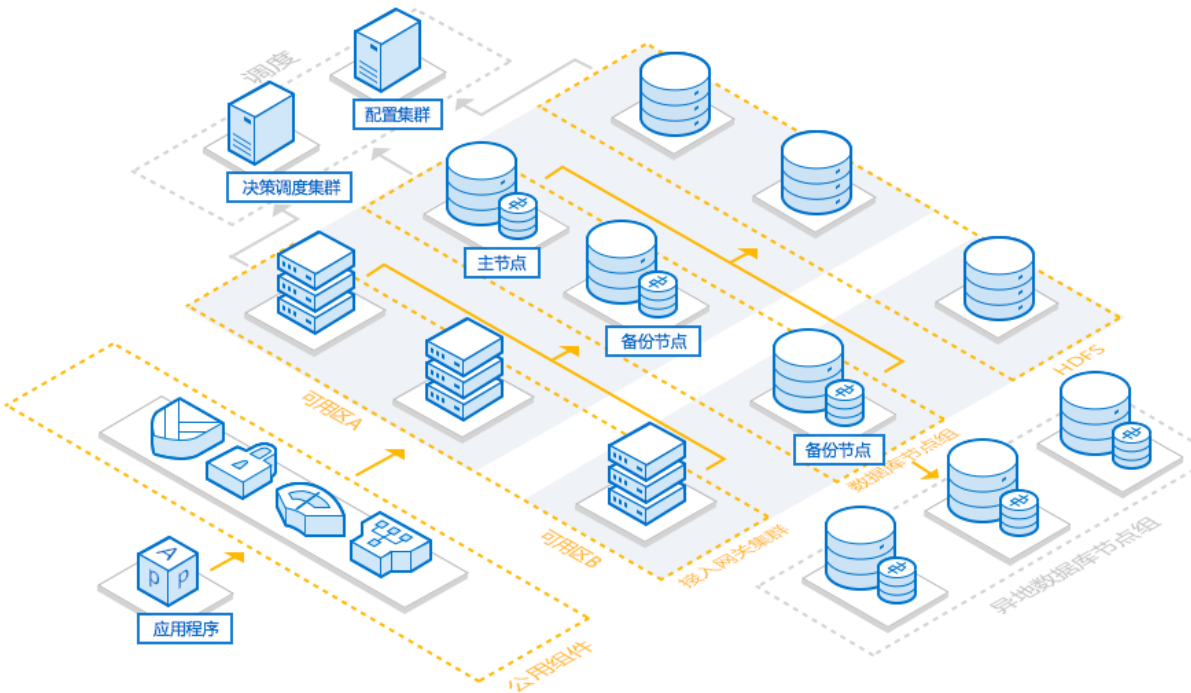


TDSQL 数据库的特点

基于Percona和MariaDB
为金融级业务设计的
与MySQL兼容

基于OLTP场景
永不停机、高一致性
数据库集群

数据库部署架构



数据库节点组(SET)由MySQL数据库、监控和信息采集模块组成一主二从数据库节点。

调度集群作为集群的管理调度中心，主要管理数据库节点组、接入网关集群的正常运行

接入网关集群账号鉴权、管理连接、SQL解析、分配路由

Hadoop分布式文件系统(HDFS)提供数据灾备服务，提供至少3份备份

异地容灾数据库节点组部署在主节点以外的异地机房。

稳定安全可靠的架构

跨机房部署
网络故障不影响业务

三重保障
集群内保障3套节点，单点故障整体稳定

数据强同步
主备数据完全一致

金融级安全
支持物理专享，支持数据库审计，支持加密等

可用性：99.999%

数据可靠性：99.99999%

TDSQL的功能优势

分布式事务处理概述

为什么，是什么，怎么做

分布式数据库的来由和挑战

数据量和访问量的压力导致分库分表

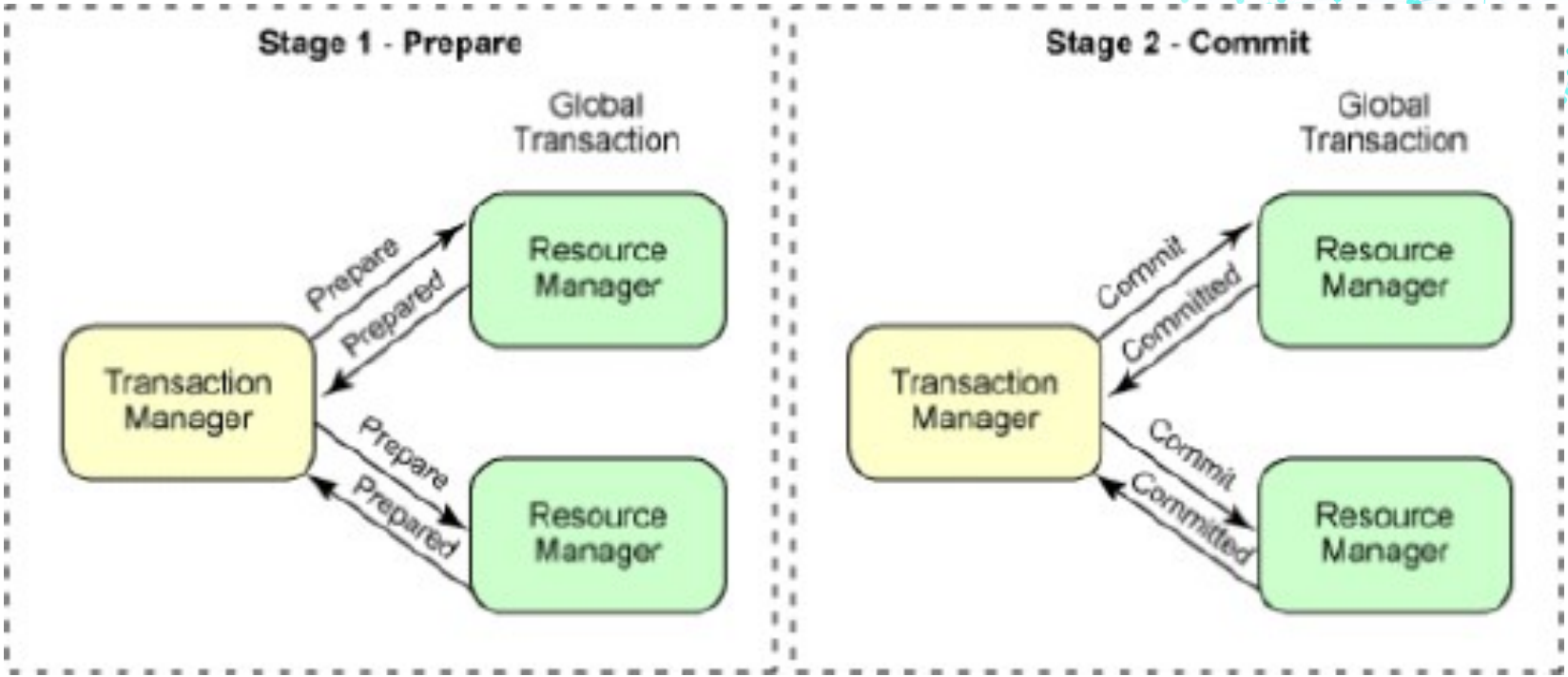
- 数据量与访问负载带来的扩展需求(scalability) ---扩容（分库分表）
- 将数据分摊到多个set（存储空间限制）
- 将负载分摊到多个set（网络和IO带宽限制，CPU和内存等资源瓶颈）
- 目标：水平扩容(scale out)后数据库集群性能提升；使用方式与单节点DB相同
- 理想目标：水平扩容后性能线性提升（数据和系统耦合性导致不可能）
- 业内现状：大多只可以访问一个set --- 数据一致性的要求
  - 需要分布式事务才能透明和可靠地在一个事务中写入多个set
- 业内现状： 大多无法做跨set的表连接和子查询
  - 需要分布式查询处理

分布式事务处理

实际业务需要分布式事务

- 同一个事务中写数据到多个set上
- 挑战：数据一致性与容灾
- 应对：定制实现应用需求比如转账
  - 技术门槛非常高，大多数中小公司做不到
  - 但他们是才是云计算(DaaS)的主力用户
- 应对：分布式事务
  - 对用户透明，没有额外的技术门槛
  - 确保分布式事务的ACID属性
  - 两阶段提交协议
  - 配合分布式查询处理效果更好

两阶段提交算法


Stage 1 - Prepare
Global Transaction
Transaction Manager
Prepare
Prepared
Prepare
Prepared
Resource Manager
Resource Manager
Stage 2 - Commit
Global Transaction
Transaction Manager
Commit
Committed
Commit
Committed
Resource Manager
Resource Manager

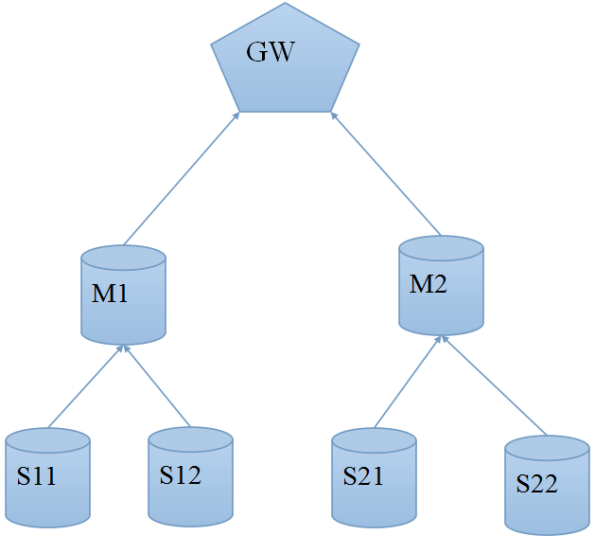
两阶段提交算法

异常处理是关键

- prepare失败
- prepare超时
- commit失败
- commit超时
- TM crash
- RM crash
- 全局死锁
- 其他

TDSQL与分布式事务

- TDSQL的典型部署架构
  - 网关（GW）
    - 任意数量，通常每个DB实例配一个
    - group shard模式解析SQL语句
  - set（1主2备）
    - 1个（noshard)或多个(group shard)
    - set1:{M1, S11, S12}; set2: {M2, S21, S22}
  - agent （每个DB实例1个）
    - 监控DB实例，完成集群下发的任务
- 网关支持用户发送多条写入SQL到多个set
  - 小表广播（一个基本静态的小表复制到所有set）
  - 多行插入语句
  - 多行更新删除
  - DDL语句
- 所有访问多个set的事务都是分布式事务
  - 内部自动识别，对用户透明
  - 两阶段提交
  - DDL语句仍然没有事务支持



TDSQL XA 示例

以全局事务T1为例，网关收到客户端SQL语句后发给后端set的SQL语句

客户端SQL语句	set1	set2
begin;		
insert into t1 values(1,2000);	XA START 'xa-gtid-1';	
	insert into t1 values(1,2000);	
insert into t1 values(3,4000);		XA START 'xa-gtid-1'
		insert into t1 values(3,4000);
commit;	xa end 'xa-gtid-1'; xa prepare 'xa-gtid-1';	
		xa end 'xa-gtid-1'; xa prepare 'xa-gtid-1';
	insert into xa.commit_log values('xa-gtid-1');	
	xa commit 'xa-gtid-1';	
		xa commit 'xa-gtid-1';

MySQL/MariaDB的XA支持

- SQL语句
  - XA START 'gtid', gtid是任意唯一字符串，长度受限
  - XA END 'gtid'
  - XA PREPARE 'gtid'
  - XA COMMIT 'gtid'
  - XA ROLLBACK 'gtid'
  - XA RECOVER 'gtid': 得到处于prepared状态的事务列表
- Mariadb-10.1.9与Percona-server-5.7.17的对比
  - mariadb XA 问题很多无法使用
    - 异常情况丢失prepared事务：客户端断开连接，mysqld crash或者正常关闭，主备切换
    - xa prepare 时并不会写入事务binlog
  - mysql 相对来说更可靠
    - 可以抵抗上述错误，但是也有bug，已经发现并解决了很多个XA bug，并报告给MySQL官方团队；
    - xa prepare时候写入事务binlog，提交时刻写入提交binlog（二者可以分开）；
  - 需要对mysql的XA 功能做强同步改造以适应TDSQL
- XA Commit语句可能失败
  - flush table with read lock （FTWRL)期间，xa commit直接出错并且innodb 事务回滚，binlog中事务状态出错，xa recover 中这个事务丢失了（mysql bug, 已解决）
  - percona xtrabackup会执行FTWRL命令做备份

TDSQL分布式事务处理关键算法

TDSQL 分布式事务处理概述

- XA 事务协调器（transaction manager, TM)实现在TDSQL 的proxy（网关）中，与网关深度耦合，只能处理支持mysql 客户端协议和标准的 SQL XA命令的数据库
- 使用commit log持久化记录事务提交决定，commit log写入后端set
- 一个tdsql 的set（一主两备）最多只会同时有1个节点故障，因而是永远存在和可访问的。更多节点故障情形在TDSQL XA容灾设计中不考虑。
- 网关实例之间互不知晓互不通信，一个group shard集群中可以有任意个网关。
- 网关无持久状态，无需容灾
  - 可以随时加入集群和退出集群
  - 退出不会丢失commit-log中的事务。
- 一个RM在任何时候并不需要知道任何网关(TM) 或者其他RM的存在。
- 主备切换不丢失分布式事务
  - 一个mysql的set中，只要主节点上面成功执行完毕xa prepare GT，那么GT的binlog一定持久存储于该set的某个备机。
    - XA PREPARE 与XA COMMIT 都做强同步等待
  - 闪回处理
- TM的ID在一个集群的历史上也全局唯一
  - 确保XA事务ID唯一
  - 由TM(proxy)的安装者、创建者分配指定。

TDSQL XA的事务协调器(TM)

- 在网关(proxy)中实现TM功能
  - 分布式事务对用户透明(start transaction/commit/rollback)
  - 允许显式事务中多条语句分别发给多个set
    - 每个事务都是全局事务(Global_txn)
    - 跟踪每个事务访问set 和访问方式（read only?)
    - 对只写入1个set的事务的优化 --- 1阶段提交
  - 支持autocommit下单条语句写访问多个set
  - DDL语句不支持分布式事务，比如drop table t1; t1是一个shard表
    - 因为mysql-5.7 DDL语句不支持事务处理
- 全局事务ID的分配和维护
  - 全局唯一但是全局无序
  - 网关 节点id全局唯一（手动配置）
  - seqno（序列号）在网关内唯一

TDSQL XA的commit-log

- 每个GT的commit log是XA协调器对这个事务的稳定可靠的提交决定
- 写入commit log的GT一定会完成提交
  - 即使某分支暂时提交失败
- 持久存储commit log 到后端set（shard表），自动获得tdsql容灾支持
- 发送XA commit到RM(GT)之前先写commit log
  - 写入出错则回滚全局事务，写入成功后才执行2PC的第二阶段
- 定期删除已提交的事务的commit log
  - agent 定期完成

TDSQL XA的两阶段提交算法要点

- agent的作用
  - 周期性做XA RECOVER检查prepared 本地事务
  - 提交commit log中决定提交的本地事务
  - abort超时处于prepared状态的本地事务
- 主备切换处理
  - 未写入commit log的事务：回滚事务
  - 写入commit log的事务：agent完成提交
  - 成功执行了XA PREPARE 的事务其binlog必然到达了备机
- 网关崩溃处理
  - 不需要做任何处理，写入commit log的事务一定会完成提交，不过client需要重新连接来查询
  - 未写入commit log的事务一定会rollback
    - 由agent完成

TDSQL XA 与强同步

- TDSQL强同步
  - 性能优于官方半同步实现
  - 优点：异步等待，不阻塞批量组提交流程，不占用线程池的工作线程
  - 每个XA事务有两个binlog event group
    - 独立且在binlog文件中通常不连续存放
- TDSQL XA强同步等待时机
  - XA PREPARE
    - 确保第一部分binlog到达备机
  - XA COMMIT
    - 确保第二部分binlog到达备机
  - XA ROLLBACK
    - 未prepare的话不等待
    - 确保第二部分binlog到达备机

TDSQL XA的隔离级别

- 分布式事务的隔离级别最高可以达到serializable。
  - 与后端set的隔离级别相同
- 分布式事务的MVCC select 快照一致性
  - 无法确保select语句得到全局一致的快照
  - 举例：GT1 做跨set 的select时，GT2正在提交
    - 发送GT1的select语句到set1上面时，T11的快照包含T21
    - 发送GT1的select语句到set2上面时，T12的快照不包含T22
  - 如果需要严格的一致性，不要使用MVCC
    - 使用select ... lock in share mode/for update
    - 把后端set的隔离级别设置为 serializable
- 容灾与MVCC数据一致性保障
  - 2pc只能做到最终一致性，MVCC下可能读取到没有完全提交完毕的GT的改动
    - 这样的全局事务必然会完成提交
  - 一般情况下，不一致的时间窗口很小；
    - 需要agent提交时会时间窗口会比较长
  - TM crash或者主备切换导致agent补充提交则有较长时间窗口部分数据不一致

04

TDSQL XA 异常与错误处理

出错才是正常现象，一切顺利是偶然现象

TDSQL XA 异常情况处理

- 客户端连接断开
- 提交GT之前，某个RM返回操作错误
  - TM发送rollback到所有RM(GT)
- phase 1 TM 发送prepare的命令，有RM返回失败或者超时未返回
  - TM发送rollback给所有RM
- phase 1 成功后，有RM crash，但是TM 会发送xa commit给RM(GT)
- commit log写入失败或者超时未返回
- phase 2 RM提交返回失败，或者提交操作超时未返回
- TM crash，有GT分别处于active，prepared，committing状态
- 局部死锁与全局死锁
- 主备切换
- 后端set的主或者备节点crash

TDSQL XA的死锁处理

- 单一MySQL进程内部死锁
  - MySQL返回死锁错误（ER_LOCK_DEADLOCK）
  - Proxy发送rollback GT，并且返回ER_LOCK_DEADLOCK给client
- 全局死锁
  - 单个master上无死锁，全局事务的等待关系构成全局死锁
  - 举例：GT1和GT2分别在set1和set2上更新1行R1和R2，事务分支：GT1 {T11, T12}，GT2{T21, T22}
    - T11锁住R1，T21等待R1行锁：T21->T11，即GT2->GT1
    - T22锁住R2，T12等待R2行锁：T12->T22，即GT1->GT2
  - 解决：网关从每个后端set上查询 innodb_trx（加列xid）和innodb_lock_waits表，得到LT(GT)的等待关系图。
  - 合并上述图，构造GT间等待关系
  - 找到环路，kill掉当前GW发起的1个db连接，返回错误
  - 定期执行与语句超时激活

TDSQL XA的主备切换与闪回

- 主备切换的原因
  - 软硬件维护或故障需要DBA手动发起
  - 软硬件故障，网络故障，程序错误导致主机crash或者失联，自动发起切换
  - 老主机crash之前未等到备机ack的事务都是多余的
    - 重新加入set变为备机后，老主机多出来的事务需要闪回
- 普通事务的闪回
  - 反向执行多余的row log event
- XA 事务的闪回
  - 参考新master的commit log
    - 回滚commit log上不存在的prepared xa txns
    - 提交commit log上存在的prepared xa txns

05

对MySQL的改进

解决了MySQL-5.7.17 XA和replication方面的多个bug

TDSQL XA容灾测试与MySQL Bug Fixes

- 创建模拟的账户数据，海量数据，分库分表
- 客户端发起大量转账事务
- 高负载，大并发访问，大数据量测试
- 每阶段10分钟，结束后验证主备数据相同，账户总余额不变
- 每个阶段频繁随机kill master，slave
  - 发起主备切换
- 每个阶段频繁stop slave; start slave;
- 发现并解决了mysql的很多bug，多数在replication与XA 的交叉领域：
  - https://bugs.mysql.com/bug.php?id=87130
  - https://bugs.mysql.com/bug.php?id=87385
  - https://bugs.mysql.com/bug.php?id=87389
  - http://bugs.mysql.com/bug.php?id=84442
  - http://bugs.mysql.com/bug.php?id=84323
  - http://bugs.mysql.com/bug.php?id=84297
  - http://bugs.mysql.com/bug.php?id=84345
  - https://bugs.mysql.com/bug.php?id=84499

Q&A

Thanks

关注开源数据库论坛