DTCC 2010-2022

第十三届中国数据库技术大会

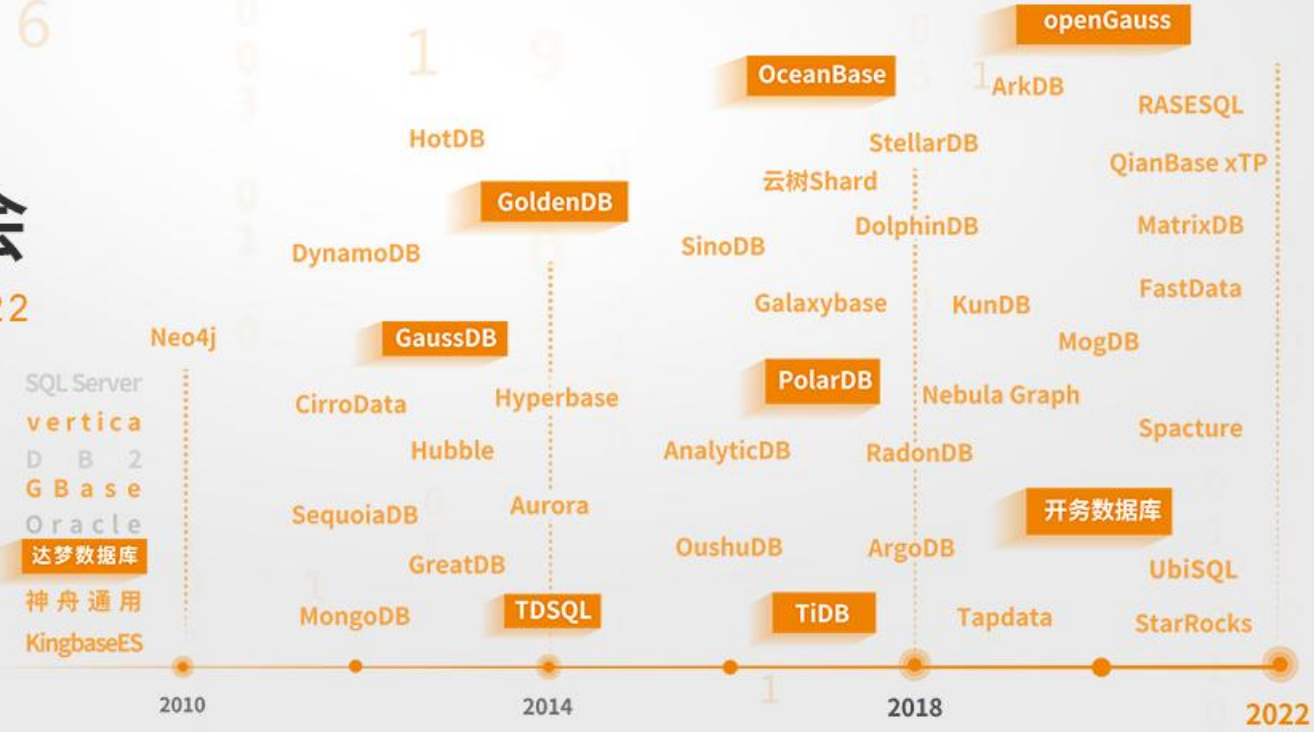
DATABASE TECHNOLOGY CONFERENCE CHINA 2022

数据智能 价值创新

IT168.com ChinaUnix.net ITPUB

线上直播 | 2022/12/14-16

数据来源：数据库产品上市商用时间



KunlunBase数据分布和弹性伸缩功能与原理简介

赵伟 泽拓科技 创始人&CEO

www.kunlunbase.com

KunlunBase

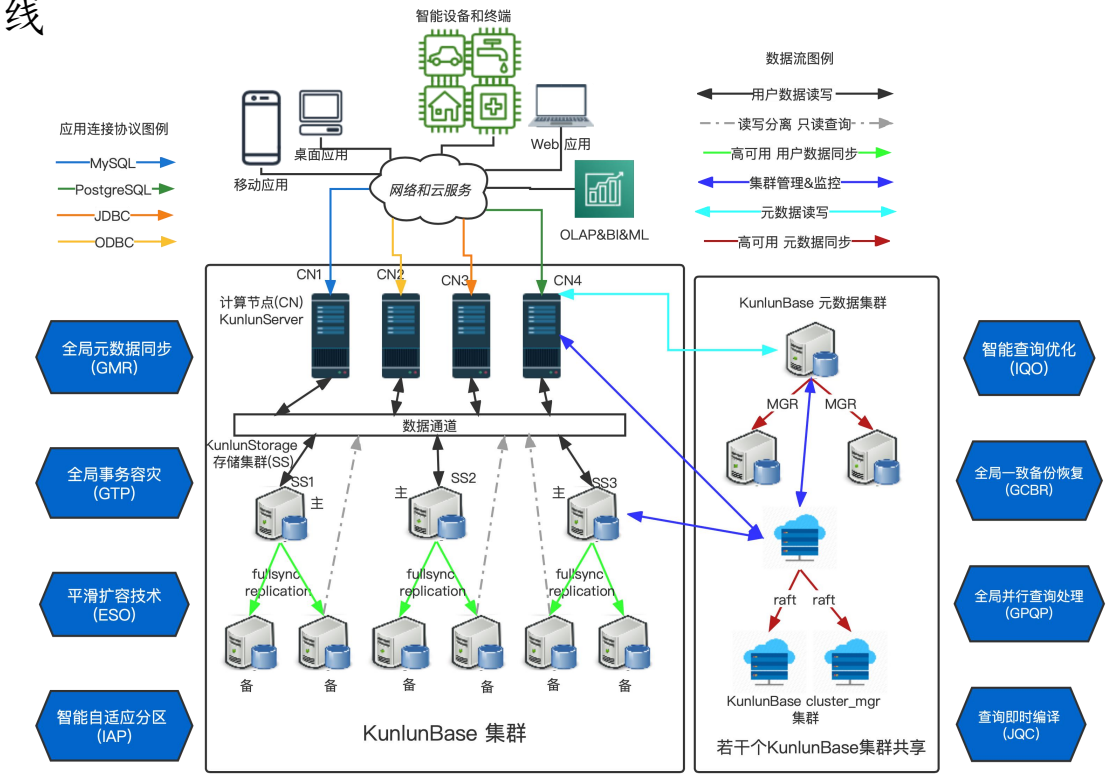
目录

- KunlunBase 架构和特点
- KunlunBase 设计哲学
- 分布式数据库弹性扩容的通用理论
- KunlunBase 数据自动分布方法
  - 经典的表分区方法及其表分布策略
  - 表分布策略适用场景及其实现技术
  - 数据表重分布 -- repartition
- KunlunBase弹性伸缩的特点、优势和实现技术
  - 为什么
  - 故障处理和恢复

KunlunBase

KunlunBase的架构和特点

- **弹性伸缩的计算和存储能力**
  - 多种可定制策略自动完成数据拆分和映射 达到最佳性能
  - 充分利用大量服务器硬件 -- 单台服务器CPU和内存有限
  - 自动、柔性、不停服、无侵入性
- 自动处理软硬件和网络故障 --数据不丢不乱，服务持续在线
  - 高可用 & fullsync强一致性
    - 自动发现主节点故障并选主和主备切换
  - 分布式事务处理
- HTAP: OLTP + OLAP --- 两类负载互不干扰
  - OLTP为主：对应用软件等价于使用MySQL或PostgreSQL
  - OLAP为辅：多层级并行实现高性能
    - 数据分析新场景
    - 通过 TPC-H & TPC-DS
- 融合标准SQL + PostgreSQL + MySQL 应用+工具软件生态
  - PostgreSQL的DDL 和DML语法 和连接协议
  - MySQL的DML语法 和连接协议
  - JDBC, ODBC, 各语言的PostgreSQL和MySQL 客户端 connector



KunlunBase 设计哲学

- 服务专业用户，用户的业务和数据具有最高的价值
  - 数据库系统和应用系统的持续稳定性、可靠性和高性能是最高目标
    - 各种软硬件和网络故障不会导致数据丢失错乱或者数据库服务停止
    - 平稳的高吞吐率和低延时
  - 提升应用软件开发人员的工作效率
    - 应用需求多种多样且异变，导致应用开发工作量很大
    - 把数据管理完全交给数据库系统，不给开发人员和架构师增加复杂性
  - 提升DBA的工作效率，但是DBA是必须的
    - 需要专业的DBA处理生产系统的复杂场景，数据库系统不存在自动驾驶
  - 用户应该、需要也能够理解自己的应用逻辑 以做到 良好的**数据库设计**
    - 应用系统主要数据表的数据分布特征和主要的数据查询方法
    - 执行者：应用系统架构师 & DBA
- 融入生态和借力
  - 合并上游更新
  - 人才资源：大量经验丰富的MySQL DBA
  - 技术资源：第三方工具，文档，经验积累

KunlunBase弹性伸缩 -- 需求和场景

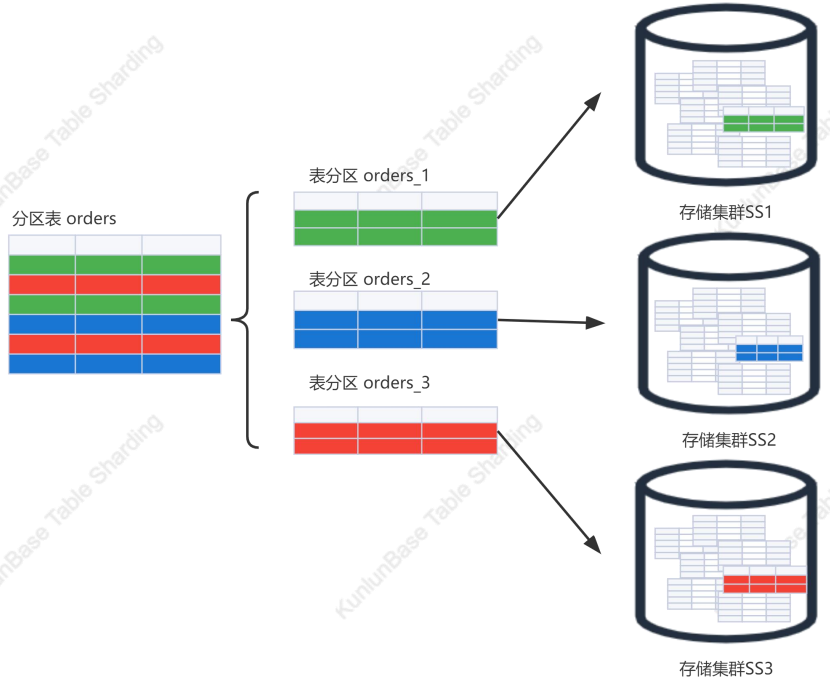
- 集群存储空间不足
  - 云环境：扩大云盘volumn，存储空间几乎可以无限增长
  - 本地环境(on-premise)：增加本地磁盘或者增加计算机服务器
- 集群计算资源不足：一台服务器的CPU和内存空间有限
  - 只能增加计算机服务器，并且实现分布式数据库系统的水平弹性扩容
    - 单机数据库（mysql，PostgreSQL，Oracle等）无法做到
    - 虚拟机的计算资源扩容能力受限于所在的物理机的最大CPU核数和内存空间
  - 无法对数据库系统实现计算资源的池化和自动弹性扩容
    - 工作集如果无法完全装入内存，则频繁换页导致性能极低
    - 应用可以自动弹性扩容，因为其无持久状态，持久状态都在数据库系统中
    - 数据库系统如果不能弹性扩容，那么应用系统整体的处理能力有硬性瓶颈
- 冷热数据分离
  - 部分kunlun-storage shard使用低成本存储系统（sata盘） 存储冷数据
  - 数据表按照时间分区，把老分区搬迁到慢shard

KunlunBase弹性伸缩 实现方法概述

- 使用分布式数据库系统时的数据库 设计和定义
  - old&common：数据表的结构、列数据类型、约束、主键、索引、字符集collation
  - new & cool：数据表分区规则和分布方法
- 支持多种分区规则定义分区表
  - 适应不同的数据分布特征和数据访问特点
  - 等值 VS 范围 ； 分区列不同值的数量多or 少？
- 支持多种表分布策略
  - 把表放到哪个shard上？
  - 最优化性能：对不同类型的表使用不同的分布策略
  - 支持调整分布方式（重分布）
- 增加、减少存储集群：伸缩计算和存储能力
  - 要点：搬迁数据表
- 增加、减少计算节点：伸缩计算能力

KunlunBase数据自动分布方法 -- 概述

- KunlunBase 数据表分布方法：分区规则 + 分布策略
  - 用户选择分区规则类型并设定参数来定义分区表及其 表分区
  - 分区规则：Hash，Range，List，擅长的数据特点和查询特点各不相同
  - 基于分布策略把表分区分配到存储集群
  - 应用读写根分区表，分区规则和分布策略对应用透明
    - 计算节点自动映射数据行读写操作到表分区和存储集群
  - 插入（INSERT）：计算节点应用分区字段到分区规则 来 映射数据行到表分区
  - 查找（删改查）：看过滤条件有无分区字段：
    - 有分区字段：计算节点使用分区字段在目标shard找到数据行；
    - 无分区字段：计算节点在多个shard查找目标表的每个分区
  - 多层级不等高的树：兼顾数据拆分逻辑和数据量



KunlunBase数据自动分布方法 -- 概述

- KunlunBase 表分区的分布策略：把分区表放到哪个storage shard
  - Least Burden：平均负载最低的shard
  - Least Space：占用了最少存储空间的shard
  - Random：随机选择一个shard
  - Mirror（复制表，镜像表）：不常更新的小表，每个shard放一份。（KunlunBase-1.1）
    - 数GB以内
    - 价值：join下推
  - Table Grouping （KunlunBase-1.2）
    - 约束：同组的表放在同一个shard，扩容时整组搬迁
    - 价值：join下推；尽量避免2PC，保持1PC的性能优势
  - 显式指定表分区所在的shard，e.g. create table t1(a int) with (shard=3)
- Repartition：修改表分区规则
  - 旧的分区规则可能不再适应当前表的数据量或者主要查询方法
    - e.g. HASH => RANGE， HASH(4) => HASH(32)
  - 单表和分区表互转：原本对表的数据量估计不准确
  - 不锁表，不影响业务运行
    - 秒级切换，业务无感知

为什么需要多种数据拆分方法

- 如果无脑切块，数据库系统如何查询数据？
  - 去含有该表所有分片的存储节点查找目标数据，但经常只有少量存储节点返回有效数据：**严重浪费系统资源**
  - 只能从存储节点拉取数据到计算节点执行表连接和聚集查询 VS 同时在多个存储节点本地做表连接和聚集查询
    - 前者需要传输的中间数据量太大，大量此类查询会严重消耗系统资源导致大量查询性能严重下降
  - 任何表索引都是全局索引，导致更新性能低，使用索引查询的性能低
- 根据数据分布特征选择查询计划 --- **极致性能**
  - Hash：对分区字段以等值查询为主，且不易估算字段值分布范围，但是每个ID值的数据量分布较均匀
    - 比如：名称，ID，邮箱，手机号 作为分区字段
    - 优点：定义简单，不需要知道字段值的分布
    - 缺点：分区字段值分布很不均匀时可能导致该表的数据分布和系统负载严重不均衡
  - Range：对分区字段同时有大量范围查询，或者分区字段值的数据量分布很不均匀
    - 比如：数量或者ID 作为分区字段
    - 优点：可以对拆分规则做精确定义
    - 缺点：随着数据变化，分不规则可能不再适合
  - List：分区字段的不同值数量较少并且只做等值查询
    - 比如：国家名，省市区名称，数量很少（数百个以内）的事物的名称、种类等枚举值
    - 优点：表定义简单直观
    - 缺点：适用场景较少

多级分区 --兼顾数据拆分逻辑和数据量

- 举例1：按省份做list拆分销售数据
  - 挑战：不同省份的人数差别很大
  - 人口稠密的省份需要按照城市做二级分区，人口稀少的省份无需二级分区
  - 少数超级大都市城市还需要第三级分区
- 举例2：按照网红ID做range 拆分订单表
  - 挑战：超级网红的带货量 千百倍于 普通网红
  - 少量超级网红每个人的ID作为一个分区
  - 按照生产厂商ID对超级网红的分区做二级分区
  - 普通网红ID按照每100个为一个分区

KunlunBase 表分布策略 -- Mirror

- 不常更新的小表，每个shard放一份，以便下推join
- mirror表与其他表t1 join
  - t1是单表：下推到t1所在的分区执行本地join
  - t1是分区表：下推到t1的所有叶子分区所在的shard做本地join
- 技术难点
  - 增加新shard：复制所有mirror表到新shard
    - 这些mirror表可能还在更新，但是不可以锁表
  - 创建mirror表 & 增加新shard 的冲突
    - 确保正在创建的表在新shard存在

KunlunBase 的表分布策略 -- Table Grouping

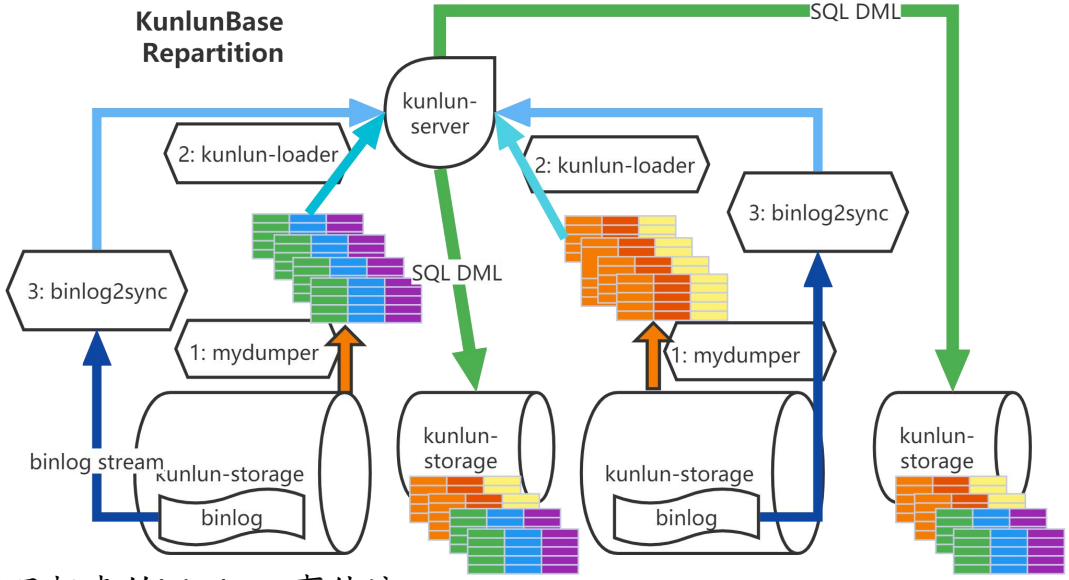
- 同组（group）的所有表或者表分片总是保持在同一个shard
  - 扩容时全组所有表都搬迁到新的shard
  - 系统禁止单独搬迁group中的某个表
  - 每个单表或者表分片可以属于一个table group，也可以不属于任何table group
- 价值
  - 用户拥有对数据表的关联性的完全定制能力
  - 同组的两个或者多个表做join始终可以下推到shard
  - 用户事务主要写入同组的表：避免或者最小化分布式事务两阶段提交的性能开销
- 关于table group的操作
  - 创建table group：指定初始存放group的shard
  - 建表：指定group id，该表被分配到本组所在的shard
  - 把表加入group：DBA要先把表move到目标group所在shard
  - 把表脱离group：脱离后表可以自由独立搬迁，并且不再随group整体搬迁
  - 删除group：其中所有表脱离该group，从而都可以自由独立搬迁

KunlunBase的表重分布（Repartition）

- 定义：修改一个分区表的分区规则
- 单表到分区表：业务规模增长远超预估，单表的数据量过大
- 分区表到单表：业务规模收缩或者预估的表数据量过大
- 修改分区规则：数据分布特点预估不准确
  - e.g. hash改为range
- 修改分区参数：数据量预估不准确
  - e.g. hash分区由8个改为64个
  - e.g. 某些账号成长为超级网红后，将其作为独立的range分区
- 拆分某个过大的表分区
  - e.g. 直播电商：将某个超级网红的表分区进一步拆分为多个分片

KunlunBase表重分布Repartition实现技术

- 前提：用户已经用新的分区规则创建好了目标表 create table t1(like t0, including xx)
- 方法：把源表的数据导出并灌入目标表，然后将此期间对源表的更新导入目标表
- 步骤一：导出表全量数据
  - mydumper dump目标表数据；
  - 传输数据文件到计算节点所在服务器
- 步骤二：加载表全量数据
  - kunlun_loader：基于myloader
  - kunlun_loader自动完成库表名映射并灌入新表
  - kunlun_loader使用KunlunBase的MySQL端口送入计算节点
  - 多表并行dump，并行load
- 步骤三：binlog catch-up --- binlog2sync
  - 在源表分片所在的所有shard上，从mydumper返回的起始位置开始，过滤目标表的binlog 事件流
  - 把针对目标表的row事件转换为SQL语句并做库表名映射
  - 使用KunlunBase的MySQL端口送入计算节点执行SQL语句
- 即将完成时通过计算节点 rename 源表和目标表
  - rename 源表名 为 无关表名
  - rename 目标表名 为 源表名



KunlunBase弹性伸缩 -- 特点和优势

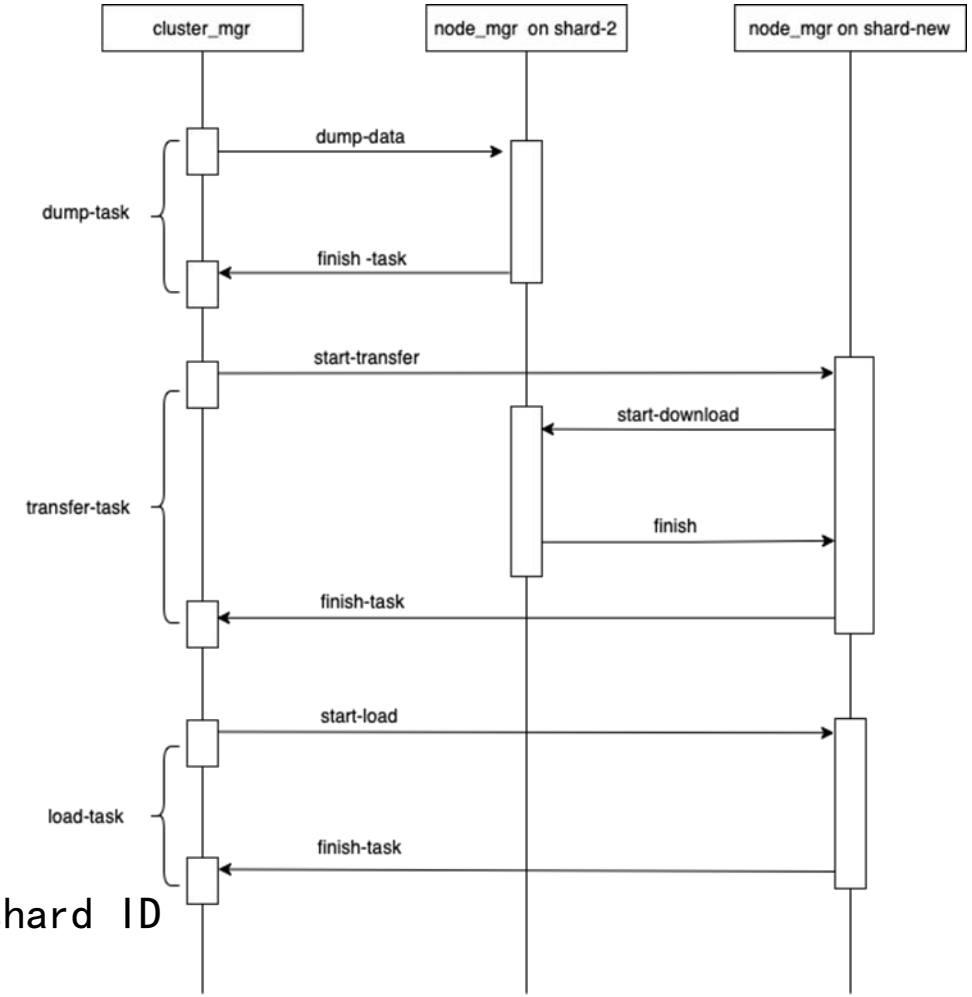
- 表分区弹性增长
  - 每个表的分区数量可以各不相同，可以单表不分区
  - Range和List分区规则下，可以按需动态增加分区；
    - HASH需要建表时创建好所有分区
- 集群规模弹性伸缩
  - 每次可以增加、减少任意个storage shard
  - 核心操作：表搬迁 --- 把数据表从一台服务器的外存搬迁到另一台服务器的外存
- 自动恢复失败任务
  - 节点故障、网络故障、存储资源不足、人工取消任务等
  - 尽量推进任务完成
  - 无法推进完成：清除过程中间数据和系统状态，以便可以随时重试

KunlunBase弹性伸缩 -- 表搬迁核心技术

- 表搬迁：在存储shard之间迁移数据表
  - 显著消耗IO带宽和网络带宽
    - 多种手段避免影响常规业务系统的性能
    - 何时搬迁：用户触发搬迁操作；
    - 限流：控制数据发送节奏，降低网络带宽消耗
  - 用户指定迁移一个shard的哪些表到哪个shard
    - KunlunBase 按照用户选定的策略给出候选列表
    - 搬迁策略：按照排序规则给用户选择表
      - size最大，size最小，最近访问，手工选择
  - 以表为单位迅速迁移大量数据，效率远高于按行迁移的扩容方式
  - 搬迁过程中不锁表，业务系统和最终用户无感知
  - 多个搬迁任务并行运行
- 全自动表搬迁
  - 用户写脚本调用cluster_mgr API 定制表搬迁逻辑
    - 自动选择搬迁哪些表到那个shard
    - 脚本自动选择搬迁的时机或者预约定时的搬迁任务

KunlunBase弹性伸缩--表搬迁核心技术

- 源shard主节点Ms：并行dump数据表
  - mydumper t1,t2...tN -> f1,f2...fN
  - 记下最早的起始binlog位置posX
- 并行压缩每一个数据文件f1,f2...fN -> fc1,fc2...fcN
- 传输每一个数据文件fc1,fc2...fcN到目标shard主节点
- 目标shard主节点Md
  - 解压每个文件fc1,fc2...fcN -> f1,f2...fN
  - myloader并行load每一个表f1,f2...fN
- 建立从Ms到Md的binlog 复制channel
  - 从posX开始复制
  - 扩展mysql实现更好性能：Ms只传输目标表t1,t2...tN的row事件给Md
    - 忽略其他表的row事件
- 即将追平时rename Ms上的表名t1,t2...tN
  - Md重放rename事件后再rename为原表名
- 在一个计算节点上执行alter table DDL修改数据表t1,t2...tN所在的shard ID
  - 其他计算节点重放DDL log完成shard id更新



KunlunBase弹性伸缩--表搬迁故障恢复

- 表搬迁任务有可能发生故障
  - 涉及源和目的 存储集群的主节点：kunlun-storage软硬件节点故障
  - GB级文件和binlog传输：网络故障，网络延时
  - 源和目的节点的本地文件存储：IO负载，可用存储空间
- 故障处理
  - 表搬迁任务状态记入元数据集群，每一步骤完成后推进任务状态
  - 故障恢复：根据任务状态，继续推进或者回滚清理残留的任务数据
  - 继续推进：任务处于表数据dump完成之后，开始追binlog之前的状态；rename原表完成之后
  - 清理：任务处于其他状态
    - nodemgr 清理任务数据文件；用户重新发起table-move任务

KunlunBase计算节点的弹性伸缩

- 为什么需要增加计算节点
  - 增加计算资源（CPU，内存）来集群的增加查询处理能力
  - 应用系统在特殊时段需要数倍于平时的并发连接
- 增加计算节点到KunlunBase集群
  - 新节点自动连接元数据集群获取并重放(replay) 本集群已有的ddl log
  - 达到最新后即可使用，在此之前会拒绝连接
- 收缩：直接删除计算节点实例
  - 用户数据存储在存储集群（storage shard）
- 如何增删计算节点
  - 通过XPanel GUI手动操作
  - 程序、脚本调用cluster_mgr API（http协议，json参数）

THANKS

KunlunBase

2018