深入理解otter

七锋 2013-07-04

Agenda

1. 中美同步需求
2. otter架构&设计
   - 如何解决"差"网络
   - 如何避免双向回环
   - 如何处理数据一致性
   - 如何高效同步数据
   - 如何高效同步文件
   - 如何支持系统HA
   - 如何处理特殊业务同步
   - 如何处理机房容灾
3. 相关产品对比
4. 其他

业务场景

1. 杭州/美国异地机房双向同步

a. 业务性 (定义同步表，同步字段)

b. 隔离性 (定义同步通道，对应一个具体业务，多个通道之间互相隔离)

c. 关联数据 (同步db数据的同时，需要同步图片，比如产品表)

d. 双A写入 (避免回环同步，冲突处理，数据一致性保证)

e. 事务性 (没有严格的事务保证，定义表载入顺序)

f. 异构性 (支持mysql/oracle)

2. 扩展业务

a. 数据仓库增量数据 (整行记录，根据变更主键反查)

b. 业务cache更新 (更新db成功的同时，刷新下cache中的值)

c. 数据全库迁移 (建立任务队列表/触发全库记录变更)

d. 多库合并同步 (product/product_detail需要尽可能保证加载顺序)

设计关注要点

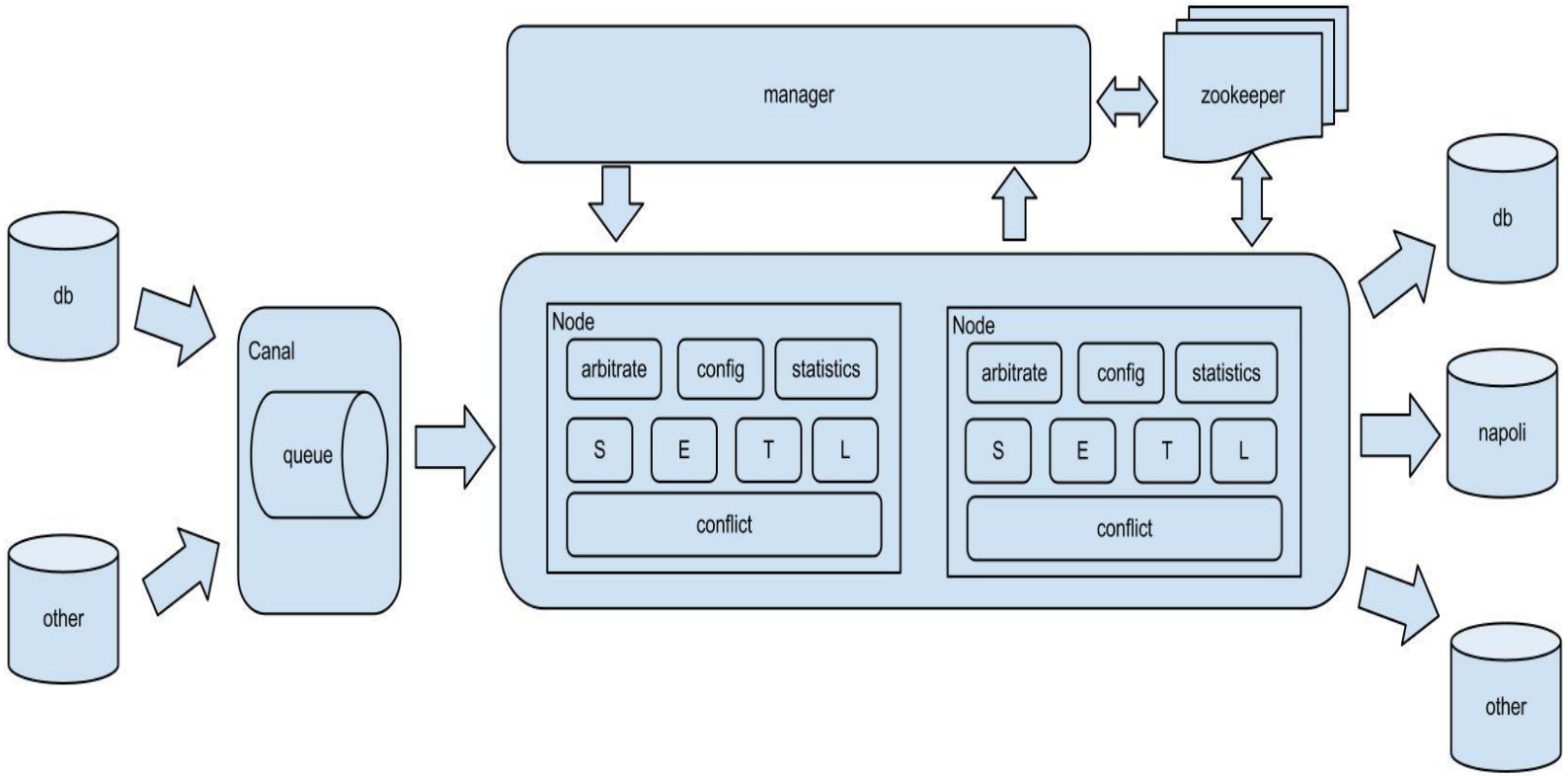
硬性要求:

1. 数据不能丢失 (变更数据一定要成功应用到目标库)
2. 数据最终一致性 (双向两边记录要保证最终一致性)

客观因素:

1. 中美网络延迟 (平均200ms)
2. 中美传输速度 (2~6MB/s)
3. 文件同步 (20000条记录可达800MB文件)
4. 同步按需隔离 (不同业务之间同步互不影响,同步有快慢)
5. 事务性支持 (允许业务定义表的同步加载的顺序性)

整体架构


manager
zookeeper
db
Canal
queue
other
Node
arbitrate
config
statistics
S
E
T
L
conflict
Node
arbitrate
config
statistics
S
E
T
L
conflict
db
napoli
other

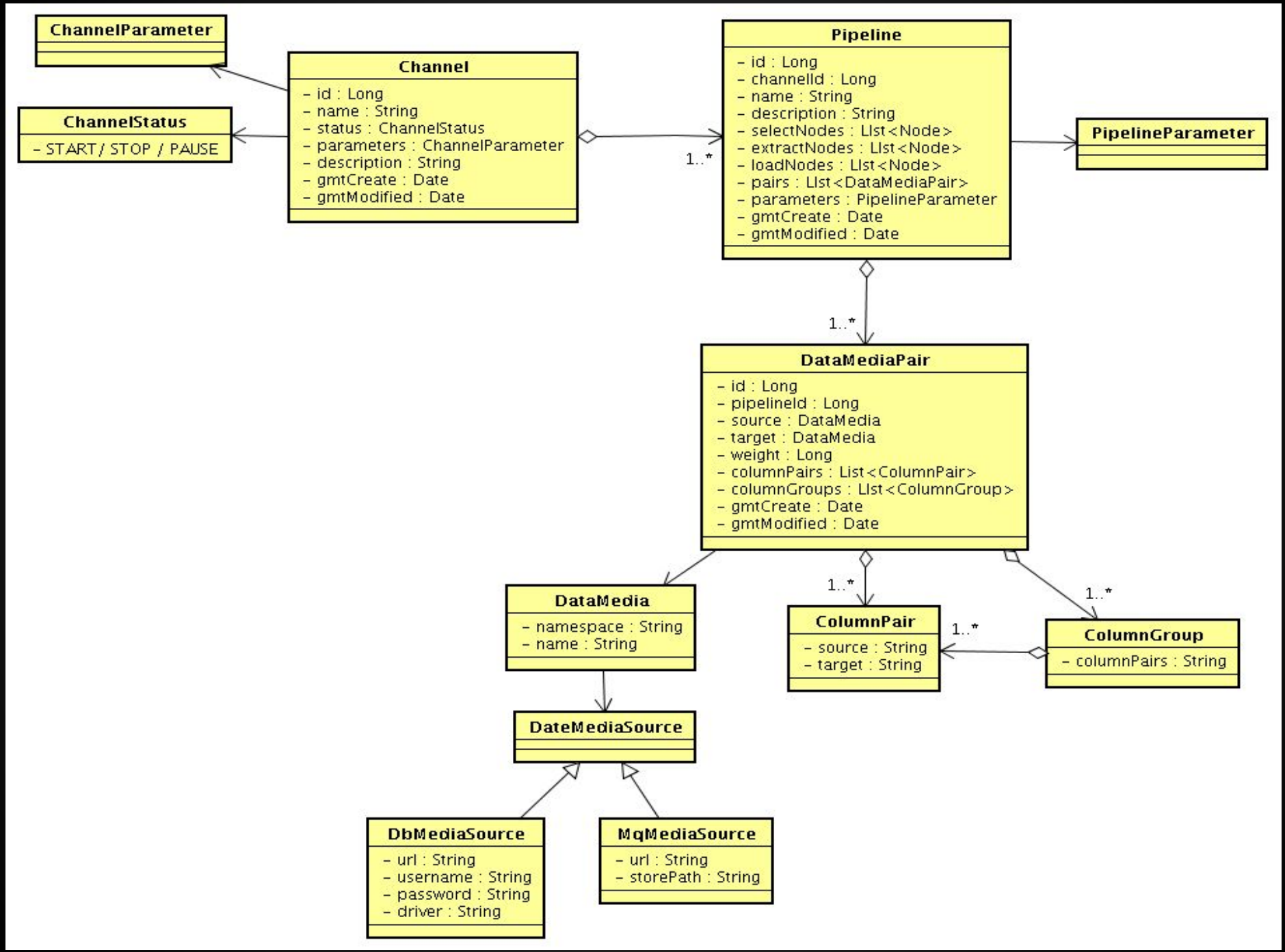
整体架构

otter整体模块

- manager (提供web页面进行同步管理)
- arbitrate (分布式调度，可跨IDC机房)
- node (同步过程setl)
- canal (同步数据来源)

大集群化部署

- 1个manager集群 + 多个IDC机房node组成


ChannelParameter
Channel
- id : Long
- name : String
- status : ChannelStatus
- parameters : ChannelParameter
- description : String
- gmtCreate : Date
- gmtModified : Date
ChannelStatus
- START / STOP / PAUSE
1..*
Pipeline
- id : Long
- channelId : Long
- name : String
- description : String
- selectNodes : List<Node>
- extractNodes : List<Node>
- loadNodes : List<Node>
- pairs : List<DataMediaPair>
- parameters : PipelineParameter
- gmtCreate : Date
- gmtModified : Date
PipelineParameter
1..*
DataMediaPair
- id : Long
- pipelineId : Long
- source : DataMedia
- target : DataMedia
- weight : Long
- columnPairs : List<ColumnPair>
- columnGroups : List<ColumnGroup>
- gmtCreate : Date
- gmtModified : Date
1..*
1..*
1..*
DataMedia
- namespace : String
- name : String
ColumnPair
- source : String
- target : String
1..*
ColumnGroup
- columnPairs : String
DateMediaSource
DbMediaSource
- url : String
- username : String
- password : String
- driver : String
MqMediaSource
- url : String
- storePath : String

名词解释

Pipeline：从源端到目标端的整个过程描述，主要由一些同步映射过程组成

Channel：同步通道，单向同步中一个Pipeline组成，在双向同步中有两个Pipeline组成

DateMediaPair：根据业务表定义映射关系，比如源表和目标表，字段映射，字段组等

DateMedia：抽象的数据介质概念，可以理解为数据表/mq队列定义

DateMediaSource：抽象的数据介质源信息，补充描述DateMedia

ColumnPair：定义字段映射关系

ColumnGroup：定义字段映射组

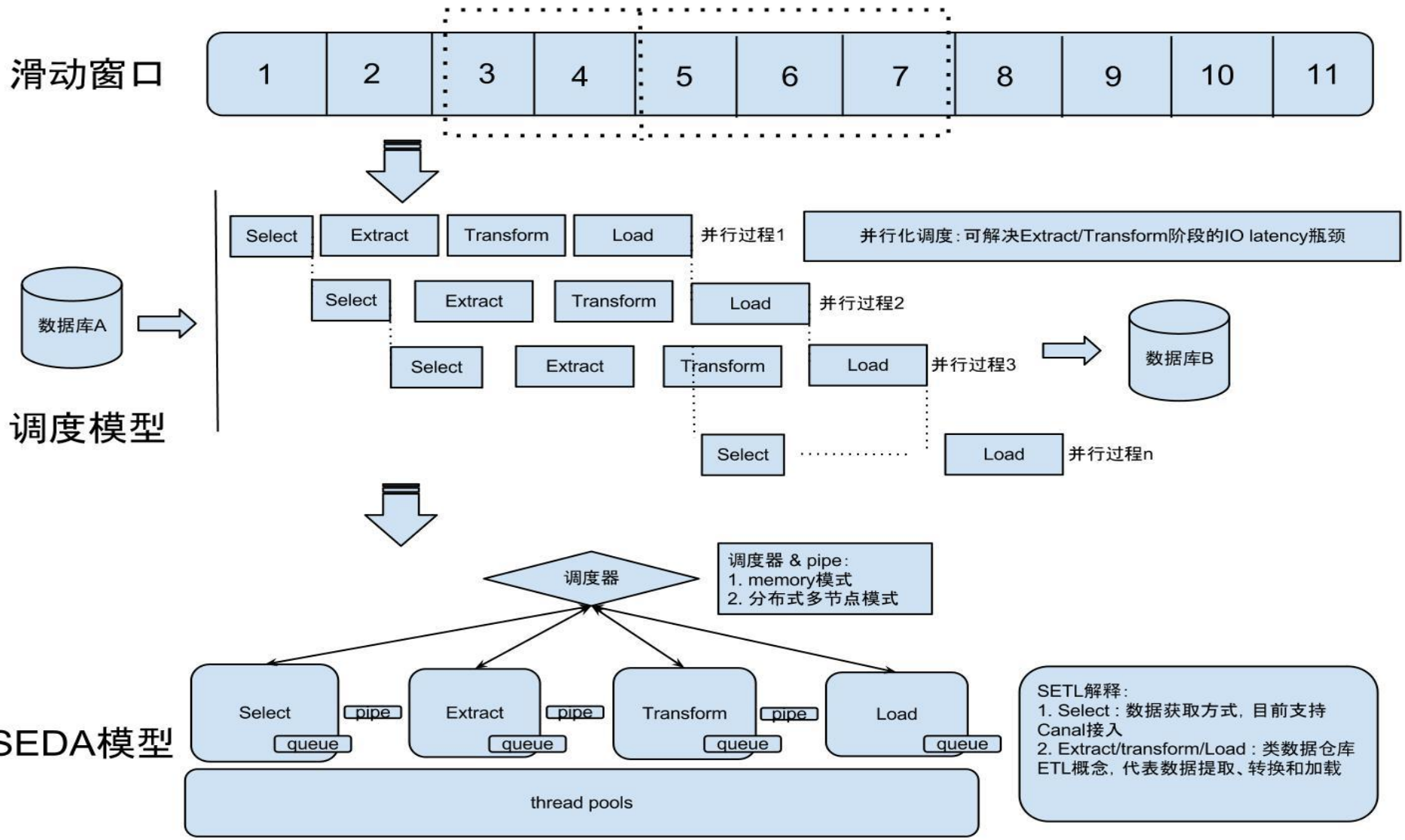
Node：处理同步过程的工作节点，对应一个jvm

如何解决"差"网络

TCP/IP协议 !!!!!


TCP传输模型
Nagle算法
滑动窗口
拥塞控制
分包策略(MTU)
丢包监测机制

如何解决"差"网络


滑动窗口
1
2
3
4
5
6
7
8
9
10
11
数据库A
调度模型
Select
Extract
Transform
Load
并行过程1
并行化调度：可解决Extract/Transform阶段的IO latency瓶颈
Select
Extract
Transform
Load
并行过程2
Select
Extract
Transform
Load
并行过程3
数据库B
Select
Load
并行过程n
调度器
调度器 & pipe：
1. memory模式
2. 分布式多节点模式
SEDA模型
Select
pipe
queue
Extract
pipe
queue
Transform
pipe
queue
Load
queue
thread pools
SETL解释：
1. Select：数据获取方式，目前支持Canal接入
2. Extract/transform/Load：类数据仓库ETL概念，代表数据提取、转换和加载

如何解决"差"网络

1. 并行化 (TCP滑动窗口模型)

    a. 梯形模型 (otter3)

原理:取一批2w数据，分成5小份，每份分配一个process处理，每份数据都处理完成后，再取下一批数据.

4w条记录，时间估算:(S+E+T)*2+100*2+10L

    b. 四边形模型 (otter4)

原理:每次取4000条数据，每完成一批，立马开启一个新的批次，尽可能保证一直有5个批次在处理.

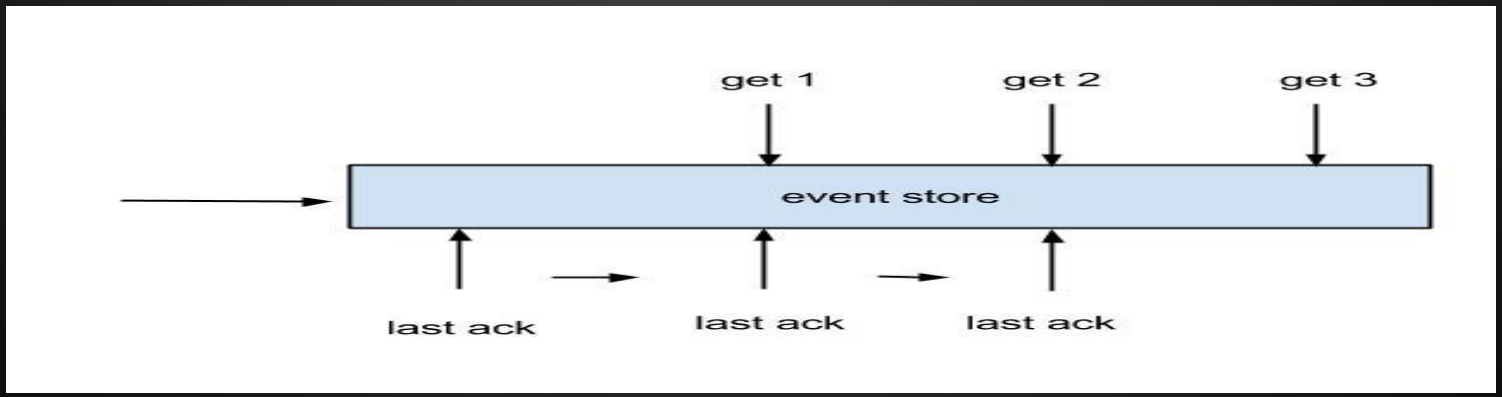
4w条记录，时间估算:(S+E+T)*1+100+10L

四边形模型在总时间上会有优势，无停顿感

如何解决"差"网络

1. 并行化（(TCP滑动窗口模型)

如何保证数据不丢：2pc.（get/ack)
如何处理重传协议：get/ack/rollback
如何支持并行化：多get cursor+ack curosr



如何解决"差"网络

1. 并行化（TCP滑动窗口模型)

基本思路:

a. 90%的情况都是正常的，异步ack机制
   i. get/ack利用tcp/ip双工，无I/O抢占

b. 出现异常，处理代价比较高，需要锁定所有操作
   i. 锁定分布式Permit，阻塞所有同步进程+线程
   ii. rollback get cursor
   iii. 开启分布式Permit，允许线程工作
   iiii. retry get

如何解决"差"网络

调度模型:
1. 正常运行调度流程
假如并行度为3

```
------------------------------------------------------------>时间轴
| ProcessSelect
--> 1
     --> 2
          -->3
              --> 1
| ProcessTermin
    ---> 1 ack
         ----> 2ack
               ---> 3ack
```

a. ProcessSelect拿到数据后，丢入pool池进行异步处理，并通知ProcessTermin顺序接收termin信号
b. 同一时间在s/e/t/l流水线上的数据受并行度控制，满了就会阻塞ProcessSelect，避免取过多的数据，只会多取一份，等待其中一个s/e/t/l完成
c. ProcessTermin接受到termin信号
   i. 会严格按照发出去的batchId/processId进行对比，发现不匹配，发起rollback操作.
   ii. 会根据terminType判断这一批数据是否处理成功，如果发现不成功，发起rollback操作

2. 异常调度流程
假如并行度为3

```
|-->1 --> 2 -->3(ing)
```

a. 当第1份数据,ProcessTermin发现需要rollback，此时需要回滚2,3份数据的批次. (可能第2,3份数据还未提交到s/e/t/l调度中)
   i. 如果2批次数据已经提交，等待2批次termin信号的返回，此时需要阻塞ProcessSelect，避免再取新数据
   ii. 如果第2批次数据未提交，直接rollback数据，不再进入s/e/t/l调度流程
b. 当所有批次都已经处理完成，再通知ProcessSelect启动 (注意: 这里会避免rollback和get并发操作，会造成数据不一致)

3. 热备机制
a. Select主线程会一直监听mainstem的信号，一旦抢占成功，则启动ProcessSelect/ProcessTermin线程
b. ProcessSelect/ProcessTermin在处理过程中，会检查一下当前节点是否为抢占mainstem成功的节点，如果发现不是，立马停止，继续监听mainstem
c. ProcessSelect进行get数据之前，会等到ProcessTermin会读取未被处理过termin信号，对上一次的selector进行ack/rollback处理
     i. 注意: ProcessSelect进行get数据时，需要保证batch/termin/get操作状态保持一致，必须都处于同一个数据点上

如何解决"差"网络

1. 并行化（TCP滑动窗口模型）

Nagle算法支持:(合并数据据包处理)

a. 构建RingBuffer (内存控制模式/数量控制模式)

b. 指定batchSize获取

i. 内存大小

ii. 记录数

c. 指定定batchSize + timeout获取

i. timeout = -1 ,即时获取，有多少取多少

ii. timeout = 0，阻塞至满足batchSize条件

iii. timeout > 0，阻塞指定的时间或者满足batchSize.

建议值:batchSize=4000(约4M) , timeout=500，内存控制模式

如何解决"差"网络

2. 调度算法

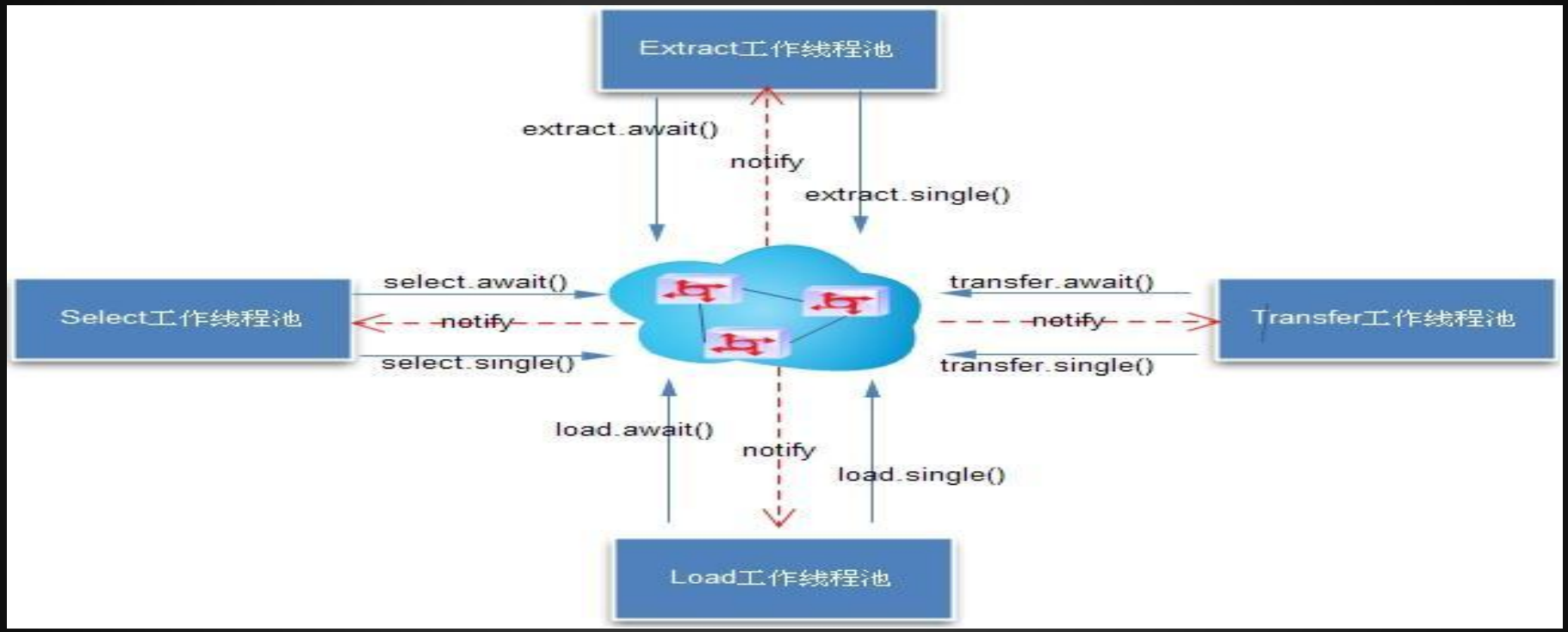
顺序性保证(令牌id):

a. select每同步一批数据，申请令牌自增id
b. 每个阶段负责传递数据+令牌id
c. load时按照令牌id顺序处理

调度模型(SEDA):

a. 共享thread pool，解决流控机制
b. 划分多stage，提升资源利用率
c. 统一编程模型，支持同机房，跨机房不同的调度算法

如何解决"差"网络



SEDA调度模型

- a. await模拟object获取锁操作
  b. notify被唤醒后提交任务到thread pools
  c. single模拟object释放锁操作，触发下一个stage

如何解决"差"网络

2. 调度算法

中美网络RTT = 200ms , zookeeper一次写入=10ms

调度成本估算:

a. zookeeper + zookeeper watch (完全分布式)

10 * 4 + 200 * 2 + 200 = 640ms

b. zookeeper + rpc (sticky分布式，尽可能选择同节点)

10 + 100 + 200 = 310ms

c. memory + memory (内存调度，单机房)

0ms

d. memory + rpc (跨机房调度，最优实现，待完成??)

0 + 100 + 100 = 200ms

如何解决"差"网络

3. 数据传输

stage间数据传递:pipe管道

stage | pipe | stage

pipe实现(数据TTL控制):

a. in memory

b. rpc call (<100kb)

c. file(gzip) + http多线程下载

如何解决"差"网络


TCP传输模型
Nagle算法
数据大小
两级buffer区
第一级，一直接收来源数据(存在大小限制 = 滑动窗口 * 二级buffer大小)
第二级，等待数据发送处理
等待时间
-1
有多少取多少，接近NO_DELAY模式
超时 or 满足数据大小
永久 or 满足数据大小
滑动窗口
数据重排
binlog增量数据发送顺序，需要和写入目标数据库顺序一致
数据全量导入/导出，可以无需重排
数据ack
单条ack
网络传输 > ack代价
保证滑动窗口的及时移动
批量ack
网络传输 < ack代价
完成一个滑动窗口一次ack
重传协议
后退n协议
canal.rollback
快速重传
canal.rollback(batchId)
拥塞控制
动态传输速率模型
基于RTT
同步到数据库速度
基于变更量
大流量，数据订正
增加网络资源
多网络通道传输
公网传输
分包策略(MTU)
小数据
on-fly快速传输
大数据
单文件存储 + 多tcp链接传输
提高压缩率
提升传输效率
丢包监测机制
计时器
数据TTL
数据2分钟，文件10分钟
等待ACK超时
一次数据包传输，默认2分钟，动态修改

如何避免双向回环

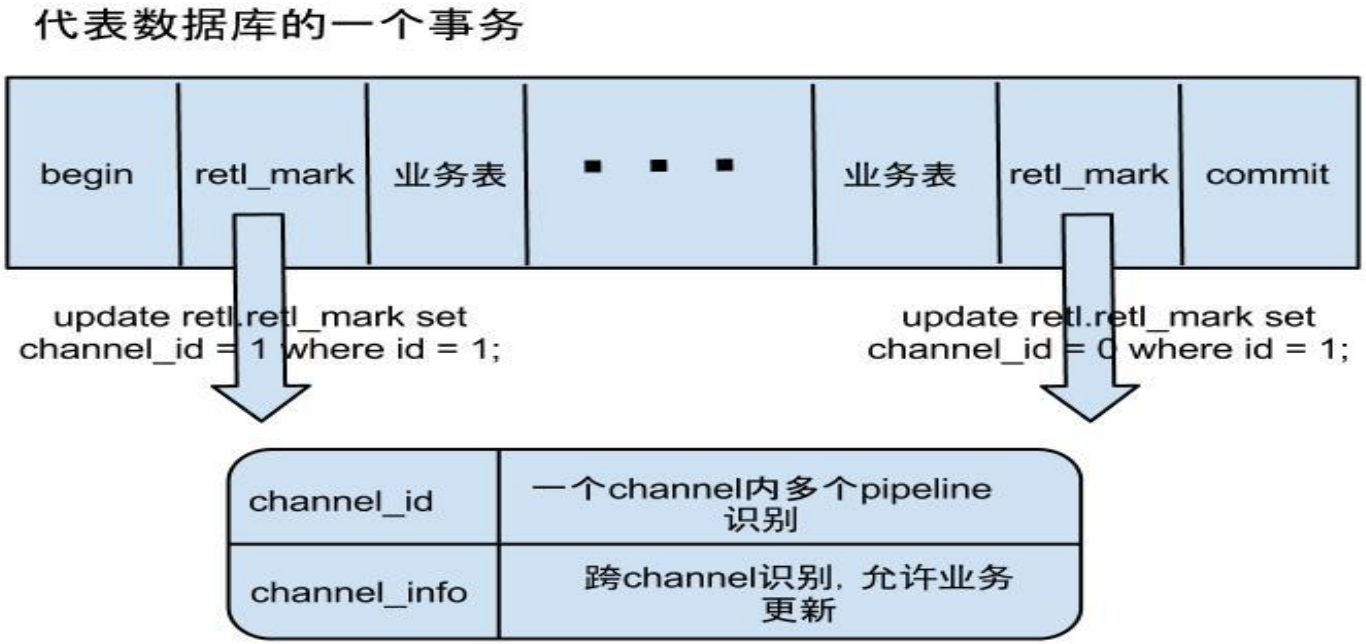
实现思路:

1. 利用事务机制, 在事务头和尾中插入otter同步标识
2. 解析时识别同步标识, 判断是否需要屏蔽同步.

几点注意:

1. 基于标准SQL实现
   可以支持mysql/oracle等异构数据库的双向同步
2. 事务完整解析&完整可见性
   事务被拆开同步, 会出现部分回环同步, 数据不一致

如何避免双向回环



几点注意：

1. retl.retl_mark表，默认初始化1000条记录，300一下属于otter内部系统使用，300 ~ 1000，属于业务系统使用

2. retl_mark表channel_info的变更需要和数据库当前值不一致，否则会出现屏蔽同步失败(mysql针对update前后值一样，不记录binlog)

如何处理数据一致性

业务场景:

a. 多地写入

b. 同一记录，同时变更

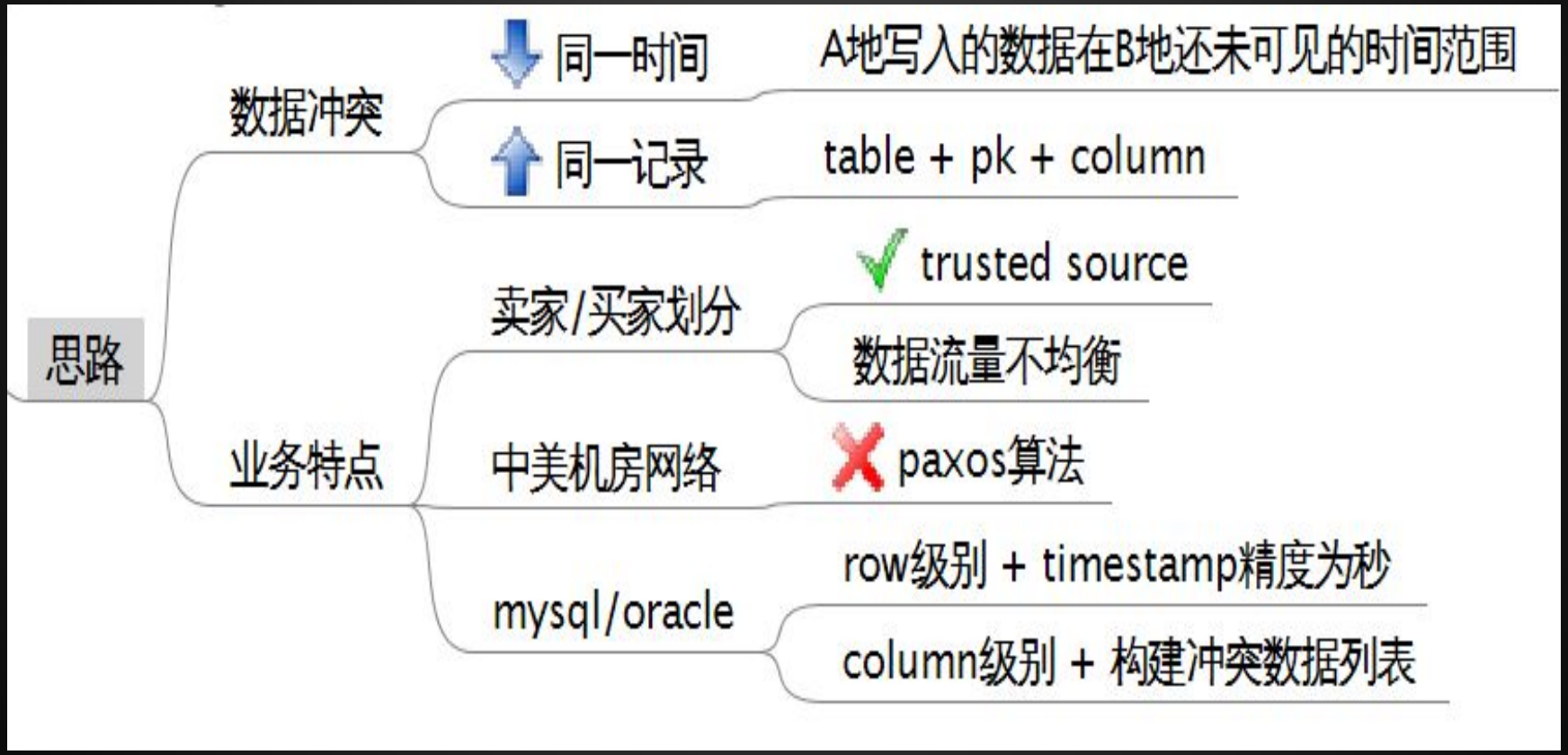
同一:具体到某一张表，某一条pk，某一字段

同时:A地写入的数据在B地还未可见的一段时间范围

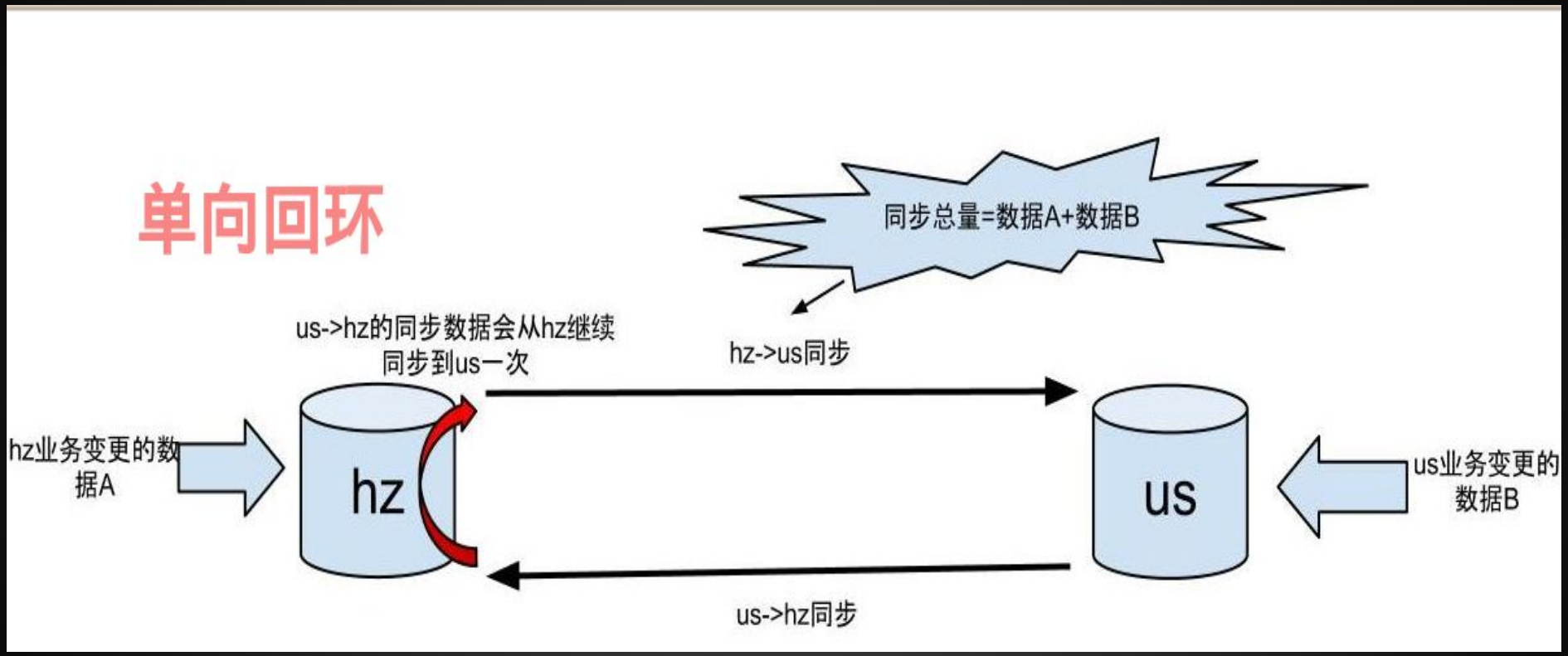
方案:

1. 检测（事前处理)
2. 补救（事后处理)

如何处理数据一致性


思路
数据冲突
同一时间
A地写入的数据在B地还未可见的时间范围
同一记录
table + pk + column
业务特点
卖家/买家划分
trusted source
数据流量不均衡
中美机房网络
paxos算法
mysql/oracle
row级别 + timestamp精度为秒
column级别 + 构建冲突数据列表

如何处理数据一致性



思路;数据最终一致性。基于trust source + 流量不均衡(杭州多，美国少)

如何处理数据一致性

单向回环流程:

- us->hz同步的数据，会再次进入hz->us队列
- hz->us同步的数据，不会进入us->hz队列(回环终止)

存在的问题:

a. 存在同步延迟时,会出现版本丢失/数据交替性变化

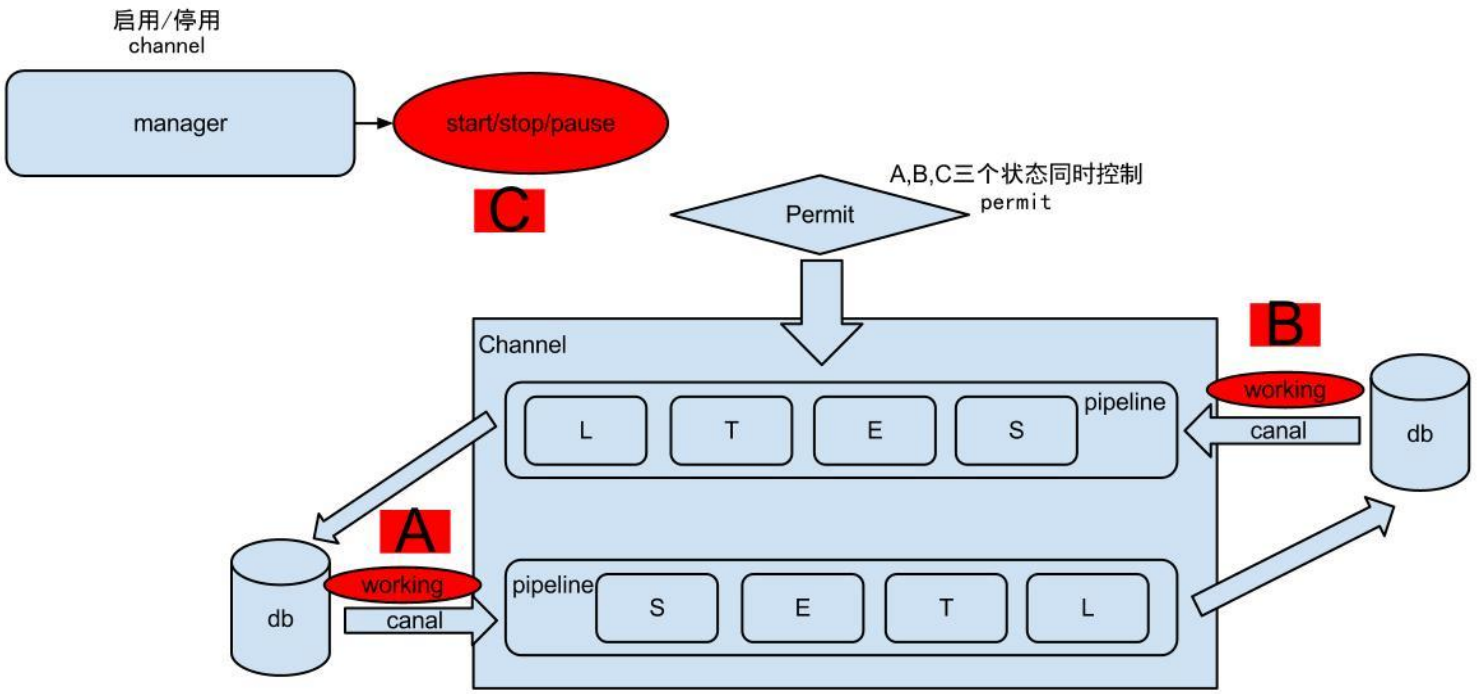
解决方案:

a. 反查数据库同步 (以数据库最新版本同步，解决交替性)

b. 字段同步 (降低冲突概率)

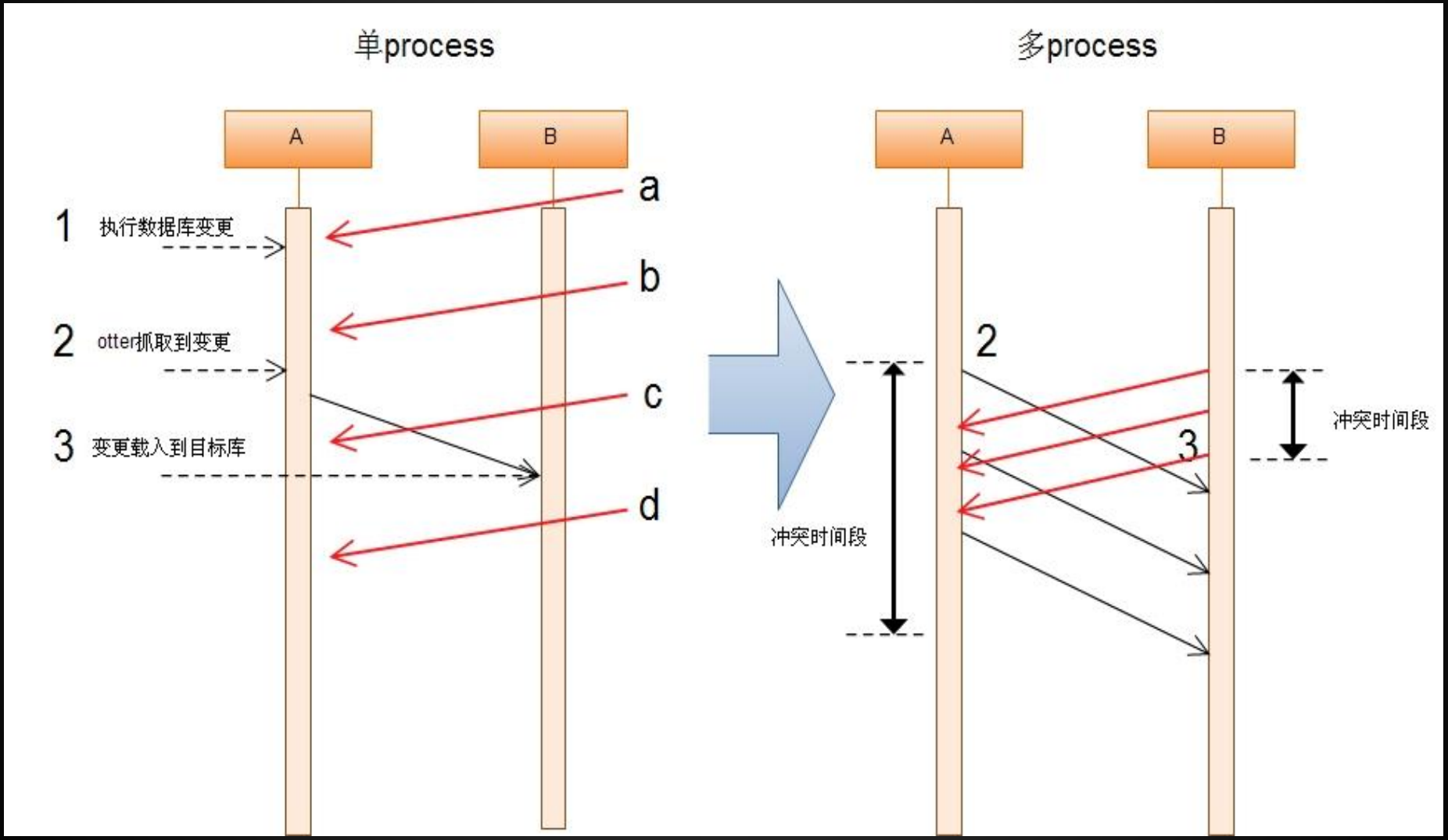
c. 同步效率 (同步越快越好，降低双写导致版本丢失概率)

如何处理数据一致性(分布式Permit)



注意:A,B,C三点状态都正常才允许进行同步(解决数据单向覆盖)

如何处理数据一致性(预研)



如何处理数据一致性(预研)

针对单process情况：

1. 出现数据不一致的可能性为双向同步中 2->3的时间段有交叉，从图中就是2->3和c的记录时间有交叉。
2. 针对a,b,d的场景，需要考虑binlog解析的延迟，时间交叉的范围判断需要加上允许的延迟时间。

针对多process情况：(binlog取出batch的数据后，进行多process同步，完成后一起提交)

1. 不一致的时间范围可扩大到batch的时间交叉.

几点设计说明：

1. 补救的触发时间点，以一方向同步的batch为单位，另一方向的同步process记录，需要进行存储(可以只存储对应的tid + pk value + column name)
2. 时间交叉范围 = (batch第一个process的起始时间 - binlog的解析延迟) -> (batch最后一个process的结束时间 + binlog的解析延迟)
3. 时间数据查询算法：
    a. 存储数据， processid , startTime , endTime (暂定存储在zookeeper中，kv不支持list的存储模式)
    b. 获取方式， 给定startTime / endTime，获取对应有时间交叉的processid

基本思路：

a. 基于同一时间的理论，找出存在时间交集的同步数据批次

b. 在交叉同步数据批次中，找出同一数据的记录，可以精确到pk或者column. (优势：减少单向回环同步的数据)

c. 发起类似单向回环同步，保证数据最终一致性.

如何高效同步数据

1. 数据最小化
    a. 数据合并
        i. 详见合并机制
    b. 数据压缩
        i. 数据protobuf存储，再gzip压缩, 20%的压缩率

2. 数据并行化
    a. S/E/T/L并行调度
    b. join并行化
    c. load并行化 (pk hash + weight)

如何高效同步数据(数据合并)

1. insert + insert -> insert (数据迁移+数据增量场景)
2. insert + update -> insert (update字段合并到insert)
3. insert + delete -> delete
4. update + insert -> insert (数据迁移+数据增量场景)
5. update + update -> update
6. update + delete -> delete
7. delete + insert -> insert
8. delete + update -> update (数据迁移+数据增量场景)
9. delete + delete -> delete

说明.

1. insert/行记录update 执行merge sql，解决重复数据执行
2. 合并算法执行后，单pk主键只有一条记录，解决并行load的效率

如何高效同步数据(load并行化)

pk hash算法:

需求描述:提升同步性能，按table粒度并行时，改善大表同步问题

解决方案:根据table + pk hash后进行并行提交

优化方案:合并相同执行sql的pk hash结果，进行batch提交 (id排序，mysql顺序写，减少网络交互）

weight算法:(业务事务性支持)

业务需求: 事务中顺序更新offer_detail，offer表，同步时插入保证顺序

解决方案: 定义offer_detail(weight=1),offer(weight=2)，按权重从小到大插入，保证在一个批次数据中offer_detail的变更要优先于offer表变更插入

如何高效同步数据(load并行化)

pk hash + weight算法:

a. 根据weight不同，构建多个weight bucket

b. 按weight顺序，对每个weight bucket执行pk hash算法

pk hash + weight + 多库复制:(文件和数据)

业务描述:

a. 数据库load完成后，发送数据到mq，或者更新cache

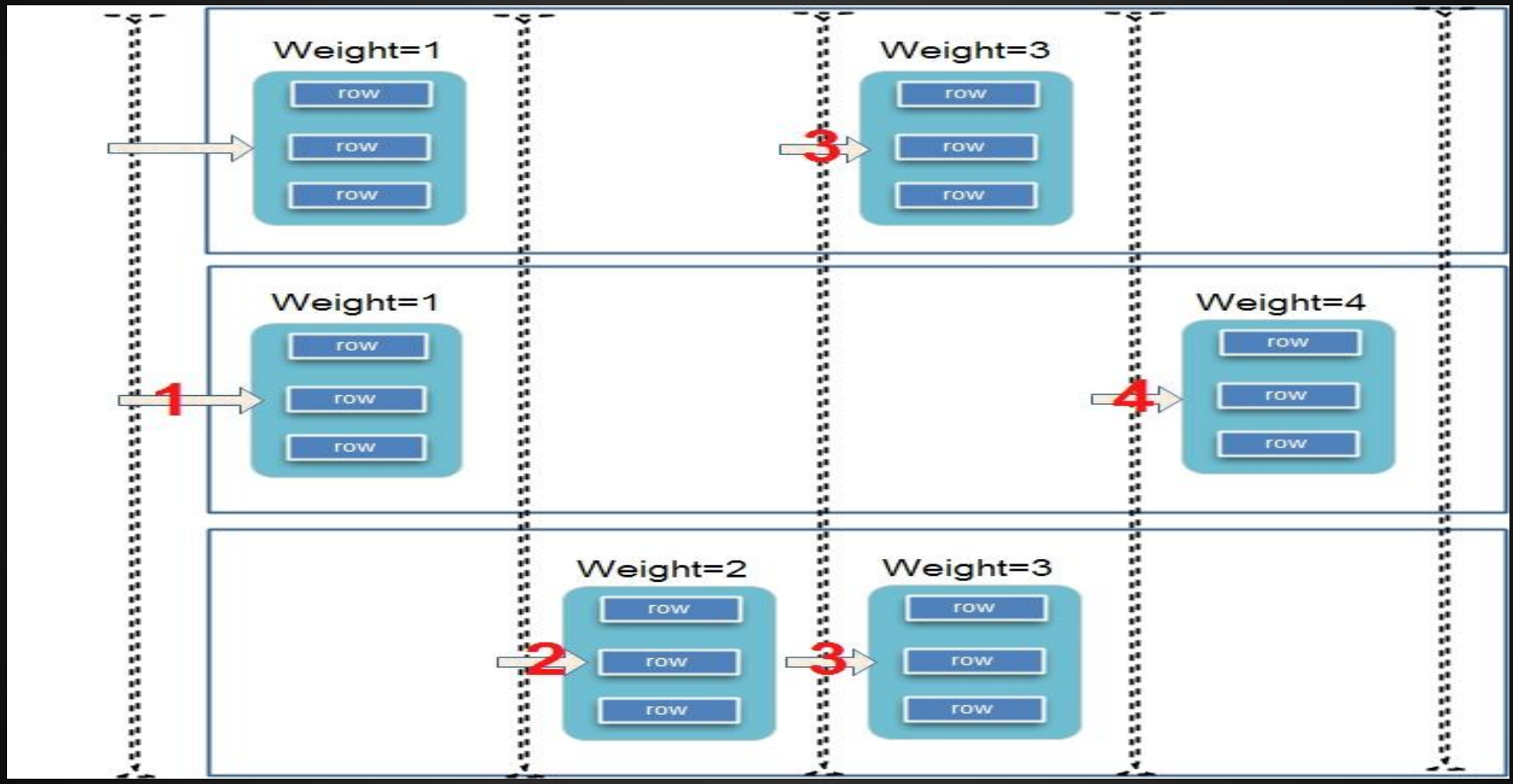
b. 会员变更数据，需要同步到多个目标数据库

算法描述:

a. 每个库创建一份load实例，并接入weight controller调度

b. 每个库按pk hash+weight混合算法进行调度，单库的weight bucket的调度由weight controllert的统一控制

如何高效同步数据(load并行化)



二维线程池weight调度:

纬度一:多库载入 ， 纬度二:单库pk hash

如何高效同步数据

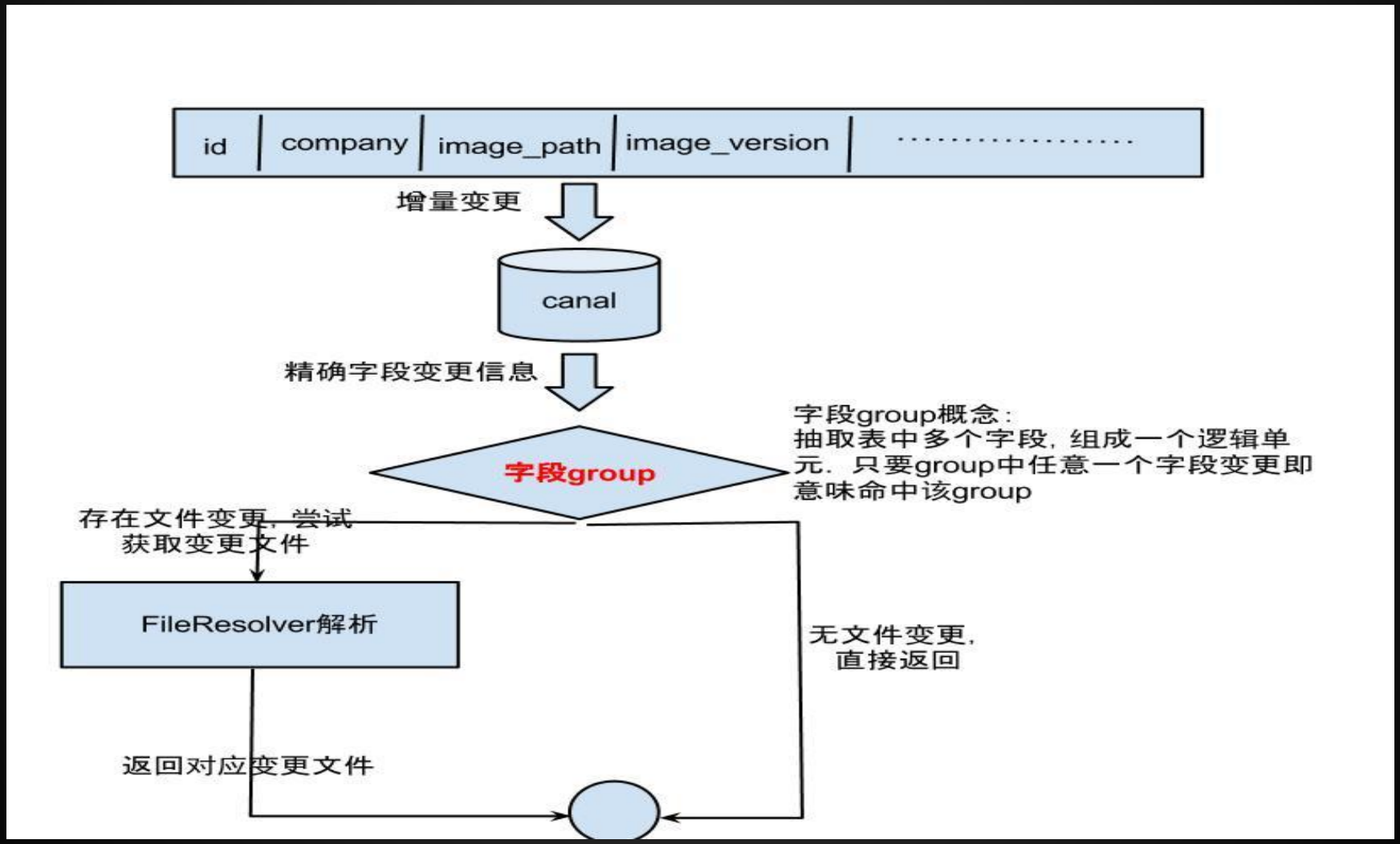
最近1天数据同步量(7月4号统计)

a. 记录数:568748541　(5.7亿)
b. 大小:377805439534 (351GB, 压缩后约为70GB)
c. 平均记录大小:664 byte
d. 高峰期带宽占用:(80%的数据产生于工作时间)
　　70GB * 80% * 2 / (8 * 3600) = 4MB/秒

如何高效同步文件

1. 文件最小化
    a. 文件变化判断
    b. 文件重复同步判断
    c. 数据压缩
        i. gzip压缩, 80%的压缩率
2. 文件并行化
    a. S/E/T/L并行调度
    b. 多线程压缩&传输&同步
3. 多网络
    a. 公网, 中美专线, 香港专线

如何高效同步文件(文件变化判断)



如何高效同步文件(文件重复同步判断)

重复同步判断依据:

1. 源文件的最后修改时间比目标文件的最后修改时间旧
2. 源文件和目标文件大小一致

注意点:

1. 文件存储时区问题.
2. 可以优化为根据文件md5比较

调用时机:

1. extract在生成文件压缩包时，会先调用美国节点，提交文件重复同步判断的请求. (多了一次网络开销)

如何高效同步文件(多线程优化)

1. 多线程压缩
   a. 多个线程并发请求aranda服务，下载为本地临时文件
   b. 压缩线程串行压缩
   目前线程数默认为10.

2. 多线程传输 (pull模式) 比如需要杭州传文件到美国
   a. 会在杭州启动一个http服务. (目前为嵌入式jetty)
   b. 美国启动多线程下载器，多socket下载文件

几点注意:

   a. sendfile技术，减少jvm内存使用，后续优化
   b. aria2c下载器 (参数：-k 2M -j 50 -s 16 -x 16)

如何高效同步文件(多线程优化)

最近1天数据同步量(7月4号统计)

a. 记录数:19495302　(2000w, AE产品占据50%以上)

b. 大小:11380354704 93 (1.03TB, 压缩后约830GB)

c. 平均记录大小:57kb

d. 高峰期带宽占用:

i. 假定:80%的数据产生于工作时间

压缩:830GB * 80% * 2 / (8 * 3600) = 47MB/秒

未压缩:1.03TB * 80% * 2 / (8 * 3600) = 59MB/秒

ii. 假定:文件同步时间段均匀分布

压缩:830GB * 2 / (24 * 3600) = 20MB/秒

未压缩:1.03TB * 2 / (24 * 3600) = 25MB/秒

如何支持系统HA

基本思路:

a. 使用zookeeper临时节点，会话失效，节点自动删除
b. manager监听node节点，冻结期设计为90秒
    i. 冻结期内，不做任何处理
    ii. 超过冻结其后，检查节点是否已恢复
        1. 如果恢复，不做任何处理 (考虑系统发布)
        2. 未恢复，进入系统HA流程.
c. HA流程
    i. 查找节点对应的同步任务
    ii. 针对每个任务发起RESTART指令. (tcp重传协议)
    iii. 阻塞分布式permit，rollback数据，开启同步.

如何处理特殊业务

1. 同步映射
   a. 1 : 1 映射, (offer -> offer，最简单业务)
   b. n : 1映射，(offer[1-32] -> offer)
   c. 1 : n 映射，(offer -> offer , offer_log) 数据多路复制

2. 视图同步
   a. 表名不同 (ocndb.member -> crmg.cbu_member)
   b. 字段名不同 (member_id -> vaccount_id)
   c. 字段类型不同 (number(11,2) -> varchar(32))
   d. 字段个数不同 (1:n映射,1个字段复制到目标多个字段)

如何处理特殊业务

扩展点：

1. FileResolver

   解决数据和文件的关联关系

2. EventProcessor

   自定义数据处理，可以改变一条变更数据的任意内容

运维方式：

a. 支持class和源码的载入

b. manager管理源码，运行时动态推送&编译

局限：无法像精卫自定义依赖lib库，无法做复杂的业务处理

如何处理机房容灾

a. zookeeper集群容灾

leader/follower： hz(3台) + cm3(2台) + cm4(2台)

observer : us(2台) 读节点，加速读请求

b. manager杭州多机房部署

node节点客户端容灾，链接失败后切到下一台.

c. node跨IDC机房部署

i. 依赖manager的HA监控机制 (node无法自己监控自己)

otter初步性能指标

吞吐量：

1. insert 30~40w/min
2. delete 60w/min

latency：

1. 本地机房+单向同步　100ms
2. 中美机房+单向/双向同步　2s
3. 中美机房+文件 10s

重要：

1. load并行线程设置很重要，取决目标库载入能力
2. latency的几个经验值，要根据数据量和高峰期做继续评估

otter4 vs otter3

otter3：

a. 文件同步 1000 / min， 60MB/min
b. 数据记录 20000 / min

otter4：

a. 文件同步 8000 / min，500MB/min
b. 数据记录 400000 / min

otter4相比于otter3，是一个数量级上的飞跃

otter"慢"在哪里？

类似产品：

a. 精卫 延迟<100ms

b. drc 延迟<1s

otter"慢"点：

a. 中美200ms延迟 vs 青岛70ms延迟

b. 中美2~6MB带宽 vs 青岛千兆光纤

Otter4使用约定

1. 同步表必须有主键
2. oracle表不允许使用blob/clob (mysql无此限制)
3. 数据订正（几种case需要和otter团队沟通）
    a. 纯数据订正超过1000w
    b. 带文件订正超过50w
    c. 非映射关系表订正超过5000w
4. 新通道上线步骤 (当前)
    a. 明确同步需求
        i. 单向 / 双向 / 双写(需要明确主要写入站点) / 文件同步
    b. 全量数据初始化
        i. 行记录 + gmt_modified修改
        ii. 插入同步记录到retl_buffer表

Otter4使用约定

5. 数据表字段变更

   a. 只允许新增字段到末尾（删除字段慎重）

   b. 字段新增先加目标库，再加源库

   c. 双向同步，新增字段建议无默认值（可确保同步无挂起）

6. 图片同步，需要先写图片，后插数据

   otter4同步延迟比较低，如果先写数据，后写图片或者两者并发写，就会有一定的概率拿到数据后，反查没有图片，导致图片同步丢失

Otter常见FAQ

1. 同步隔离性

a. otter pipeline按表级别定义同步映射，不同pipeline互不影响

b. 接入erosa+canal，按库存储数据，不同表同步会存在一定影响

2. 同步延迟

取决目标数据库可接受的load并发度 + 地域之间的网络延迟

3. 核心竞争力

a. 并行调度模型，(缓解extract/transform I/O latency问题)

b. 双向同步 / 双A同步 (避免回环同步 / 冲突检测)

c. pk hash + weight并行载入 (极大的提升同步性能)

d. 接入canal，高效获取增量数据，并按变更字段同步 (高效,低latency)

e. 同步映射 / 视图同步 / 数据join / 数据filter (强大的功能支持)

otter资源

1. otter manger

   http://otter.alibaba-inc.com

2. 相关文档

   http://b2b-doc.alibaba-inc.com/display/RC/Otter

   http://b2b-doc.alibaba-inc.com/display/opentech/otter

3. 需求平台

   http://agile.alibaba-inc.com/browse/OTTER

TKS!