Shopee 数据事件中心的设计和实现

林锋

Shopee 技术平台团队负责人

Geekbang. 极客邦科技 InfoQ

极客邦科技
会议推荐2019

5月

QCon 北京
全球软件开发大会
大会：5月6–8日
培训：5月9–10日

QCon 广州
全球软件开发大会
培训：5月25–26日
大会：5月27–28日

6月

GTLC GLOBAL TECH LEADERSHIP CONFERENCE 上海
技术领导力峰会
时间：6月14–15日

GMTC 北京
全球大前端技术大会
大会：6月20–21日
培训：6月22–23日

7月

ArchSummit 深圳
全球架构师峰会
大会：7月12–13日
培训：7月14–15日

10月

QCon 上海
全球软件开发大会
大会：10月17–19日
培训：10月20–21日

11月

GMTC 深圳
全球大前端技术大会
大会：11月8–9日
培训：11月10–11日

AiCon 北京
全球人工智能与机器学习大会
大会：11月21–22日
培训：11月23–24日

12月

ArchSummit 北京
全球架构师峰会
大会：12月6–7日
培训：12月8–9日

About The SPEAKER

林锋 Manager, Platform Engineering

- **2012年加入 Sea Group**
- **目前在 Shopee Engineering & Technology 部门担任 Shopee 技术平台团队负责人**
- **负责容器平台、网关、中间件、服务网格等基础技术平台的建设**

TABLE OF CONTENTS 大 纲

- 起源：缓存更新
- 数据事件中心的设计
- 实际应用和遇到的问题
- 未来规划

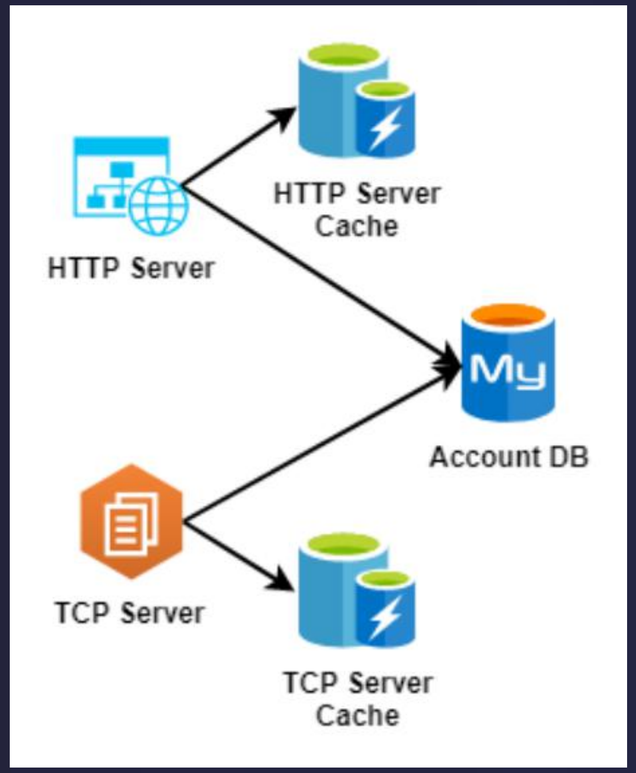
起源：缓存更新

起源

"There are only two hard things in Computer Science: cache invalidation and naming things."

- Phil Karlton

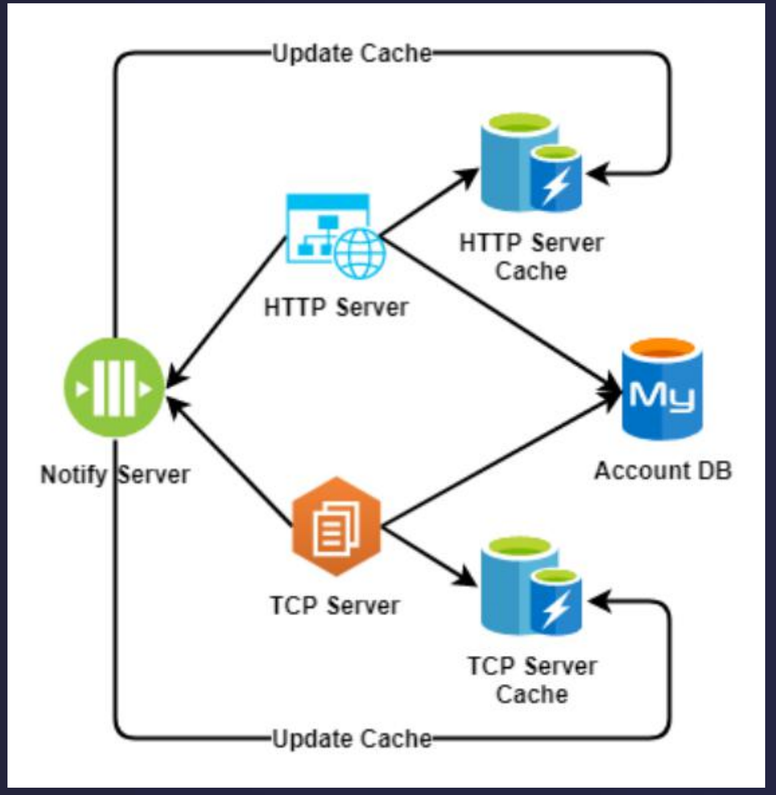
多个应用共享数据，有各自的缓存，如何保证缓存同步？



缓存更新

方案一：应用程序更新

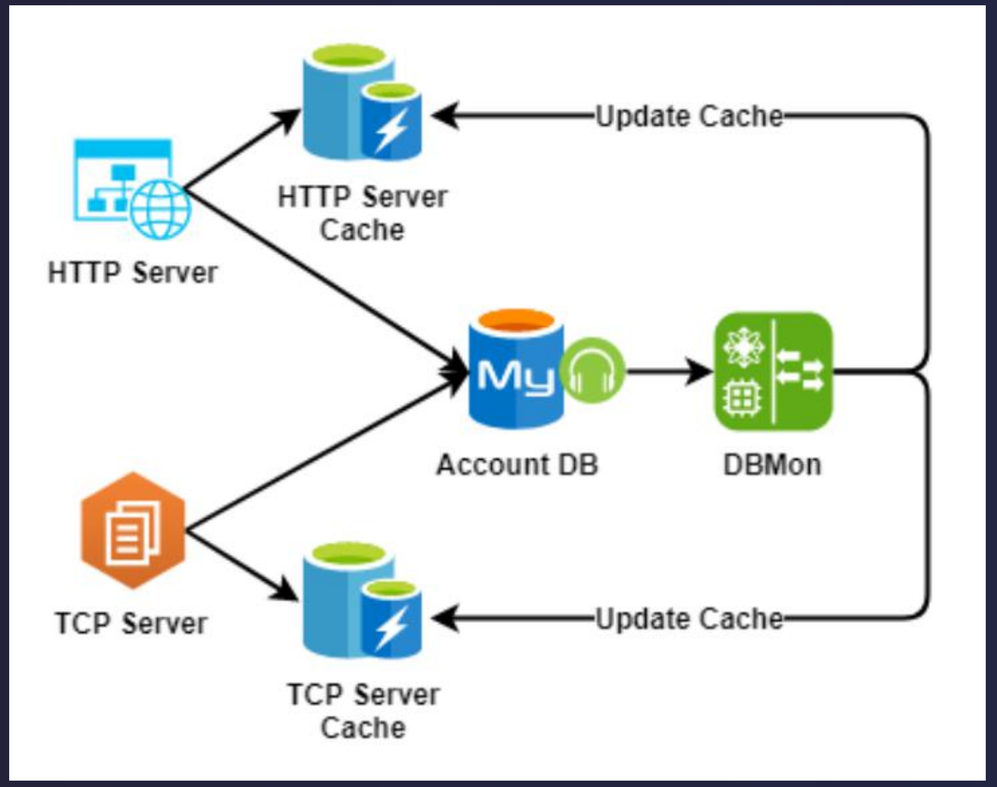
- 代码冗余
- 容易遗漏
- 应用程序间互相耦合
- 外部无法直接更新数据库



缓存更新

方案二：MySQL Trigger

- 应用程序间解耦
- 侵入MySQL，影响稳定性
- 难以维护

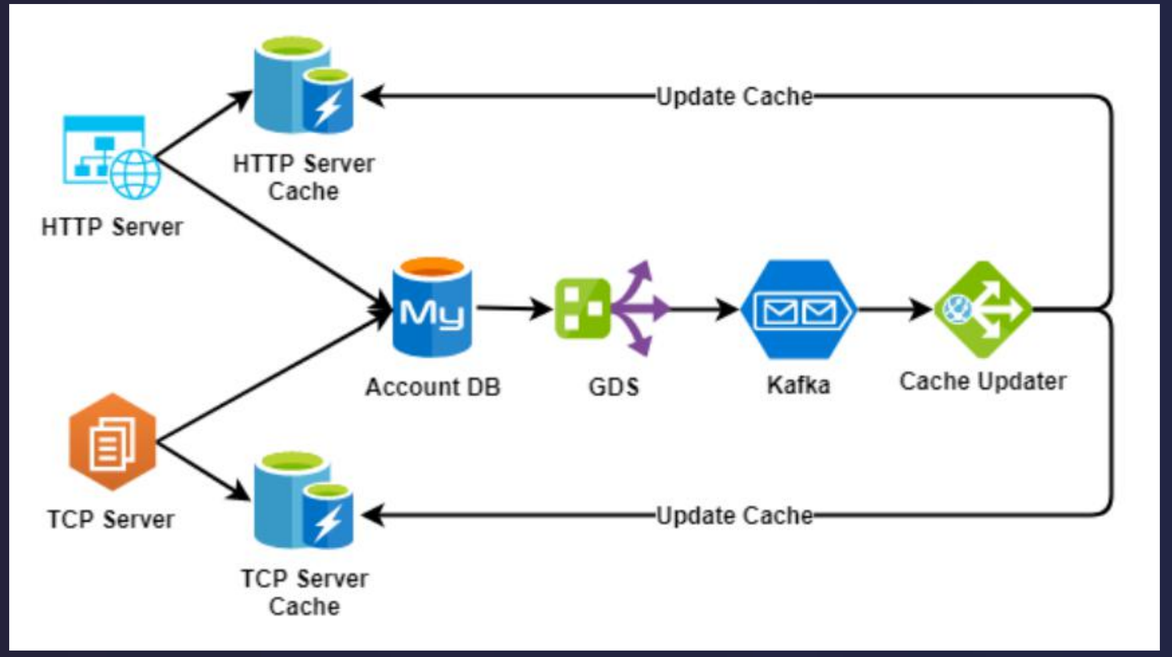


缓存更新

方案三：MySQL Binlog → GDS

（General Data Syncer）

- **应用程序间解耦**
- **对应用程序和MySQL无侵入**
- **易于维护**



GDS

C++

- 读取 Binlog
- github.com/bullsoft/mysql-binlog-events

V1

Golang

- Master Slave Replication
- github.com/siddontang/go-mysql

V2

GDS 事件数据

MySQL中每行数据变化对应一个事件数据

	Protobuf	JSON
编码效率和性能	高	低
易用性	解码需要数据定义 ×	解码不需要数据定义 √

```
{
        "command": "update",
        "table": "test_db.test_tab",
        "timestamp": 1347876077,
        "fieldnum": 3,
        "newrow": {
                "0": "30020",
                "1": "username1",
                "2": "0"
        },
        "oldrow": {
                "0": "30020",
                "1": "username2",
                "2": "16"
        }
}
```

GDS 事件数据

MySQL Binlog 的限制

- 缺少字段名：使用字段位置作为索引
- 整型无法区分 SIGNED / UNSIGNED：始终解释为SIGNED
- 无法区分 CHAR / BINARY：始终HEX编码
- CHAR / VARCHAR 包含 PADDING：移除尾部所有空格和 0x00

字段：name CHAR(8)
SET name="test"
实际存储："test□□□□"

以下查询都能匹配：
WHERE name="test"
WHERE name="test□□□□"

GDS 事件数据存储

	RabbitMQ ×	Redis √	Kafka √√
性能	一般	高	高
重复消费	不支持	支持（客户端逻辑）	支持
保证顺序性	否	是	是

GDS 状态存储

使用 ZooKeeper 存储 GDS 状态

- **Kafka 集群信息**
- **每个小时的偏移位置：方便重放**

缓存更新：CacheUpdater

配置

```
<remove table="test_db.test_tab" cache="gdp" key="GetUser:[uid]" />
<remove table="test_db.test_tab" cache="gdp" key="CheckUser:[uid],[username]" />
```

生成缓存 Key

GetUser:30020

CheckUser:30020,username1

CheckUser:30020,username2

```
{
        "command": "update",
        "table": "test_db.test_tab",
        "timestamp": 1347876077,
        "fieldnum": 3,
        "newrow": {
                "0": "30020",
                "1": "username1",
                "2": "0"
        },
        "oldrow": {
                "0": "30020",
                "1": "username2",
                "2": "16"
        }
}
```

遇到的问题

字母大小写导致缓存更新不完全

- Redis / Memcached 缓存 Key 区分大小写
- MySQL utf8mb4_general_ci 不区分大小写

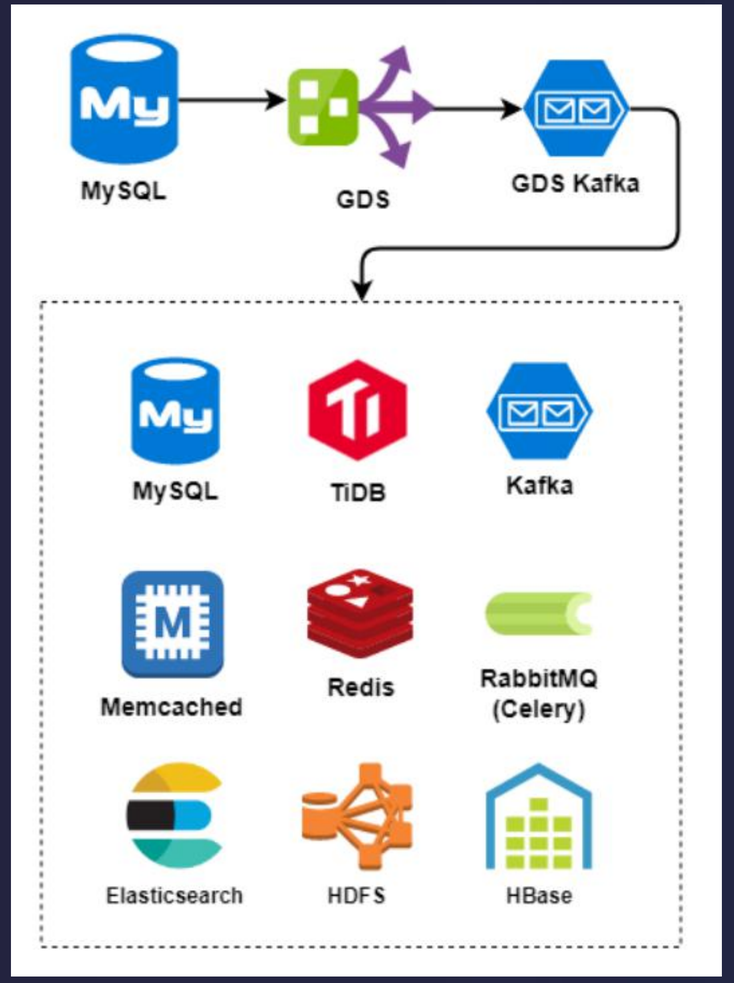
解决方案

- 写入缓存及更新缓存时：生成缓存 Key 时始终将数据字符转换为小写

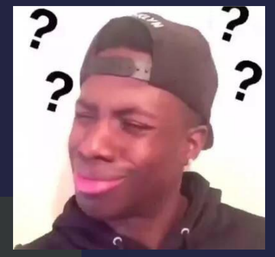
数据事件中心的设计

基于 GDS 的应用

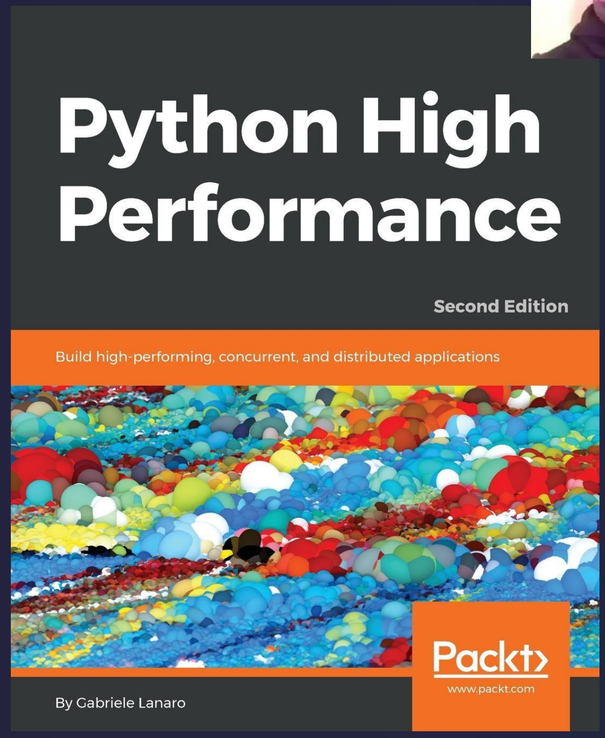
- 缓存更新、同步
- 数据库同步：同构、异构、跨机房
- 回调触发
- 索引维护
- 数据仓库同步



尝试实现通用框架



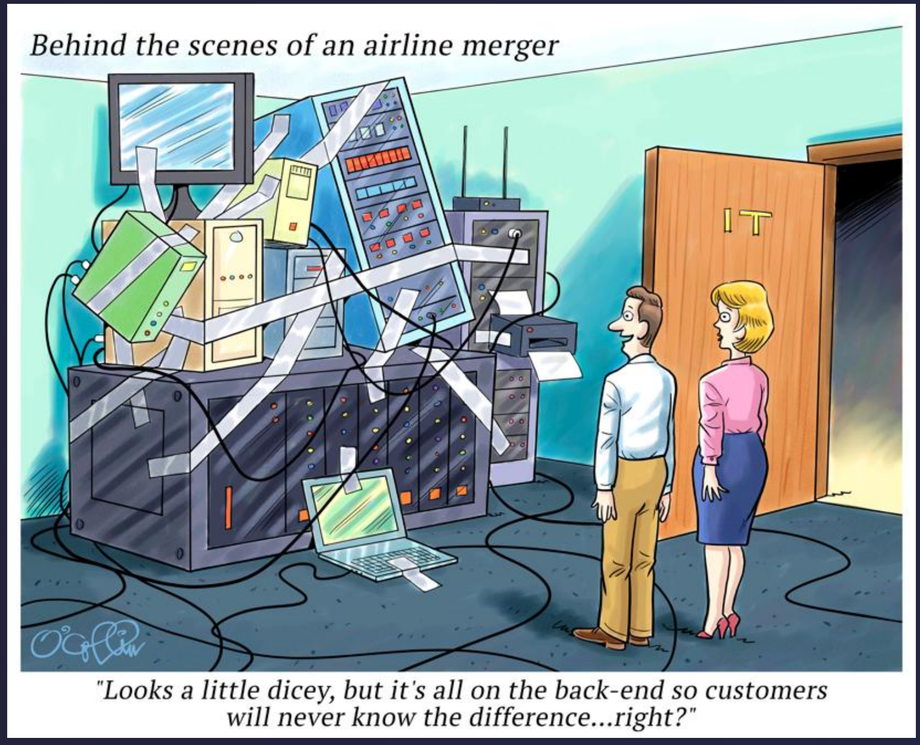
- 使用 Python 实现
- 通过修改代码添加任务
- 难以维护
- 性能问题：最高 1000 TPS



打造通用平台

DEC (Data Event Center)

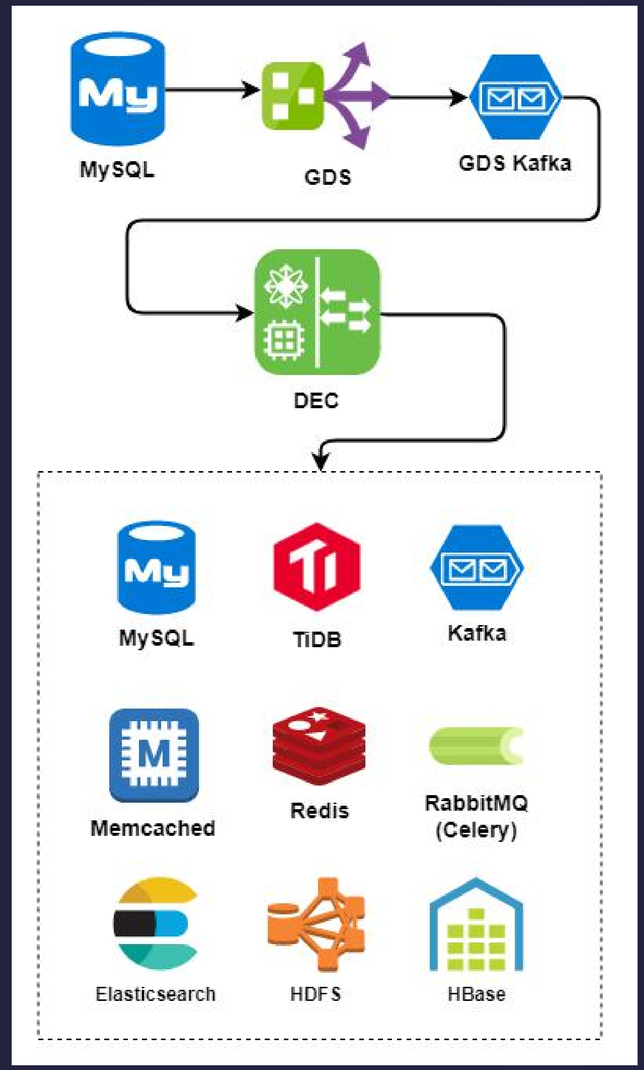
- 易于使用，配置灵活
- 稳定，高可用
- 高性能，可伸缩
- 保证一致性



DEC 任务流程

支持多种写入目标

- MySQL
- TiDB
- Redis
- Memcached
- Kafka
- RabbitMQ (Celery)
- …



任务配置

```
{
    "is_active": true,

    "tasks": [{
        "id": "dec_shopee.promotions.promotion",
        "active": true,

        "source": {
            "table": "shopee_ips_promotion_db.promotion_tab",
            "operations": "insert, update"
        },

        "actions": [{
            "id": "promo_action",
            "type": "db",
            "dst": "shopee_ips_promotion_observer_db",
            "transform": {
                "table": "promotion_cron_tab",
                "primary_key" : "promotion_id",
                "script": "promo_cron.lua",
                "operations_mapping": [
                    {"src": "insert", "dst": "insert" },
                    {"src": "update", "dst": "update" }
                ],
                "columns_mapping": [
                    { "src": "promotion_type" },
                    { "src": "promotion_id" },
                    { "src": "start_time" },
                    { "src": "end_time" },
                    { "src": "status" }
                ]
            }
        }]
    }]
}
```

简单，适合一般任务

使用 JSON 配置管理任务

```
function transform(ev, q)
    q:SetIgnoreCollision(true)

    if (ev:Operation() == CDelete) then
        skip = true
        return
    end

    if validate_time_col(ev, "start_time") == false or
       validate_time_col(ev, "end_time") == false then
        skip = true
        return
    end

    q:CopyAllColumns(ev:New())

    if (ev:Operation() == CInsert) or
    (ev:Operation() == CUpdate) then
    q:AddColumn("mtime", CInteger):Set(NewDateTime():Now():Timestamp())
    end

    if (ev:Operation() == CInsert) then
        q:AddColumn("processed_start", CInteger):Set(0)
        q:AddColumn("processed_end", CInteger):Set(0)
    end

    if (ColumnExists(ev:New(), "promotion_type") == false) then
        q:RaiseError("column [promotion_type] is not present in new or old row!")
        return
    end

    if (ColumnExists(ev:New(), "promotion_id") == false) then
        q:RaiseError("column [promotion_id] is not present in new or old row!")
        return
    end

    if (ev:Operation() == CUpdate) then
        q:BegCond()
            q:Condition("promotion_type", Eq, ev:New()["promotion_type"]:Val())
        q:EndCond()

        q:AndCond()
            q:Condition("promotion_id", Eq, ev:New()["promotion_id"]:Val())
        q:EndCond()
    end

    q:Commit()
end

function validate_time_col(ev, col_name)
    if ColumnExists(ev:New(), col_name) then
        if ev:New()[col_name]:Val() == CNull then
            return false
        end
    end
    return true
end
```

灵活，适合复杂任务

支持 LUA 脚本

任务配置

数据解析、过滤、生成支持：

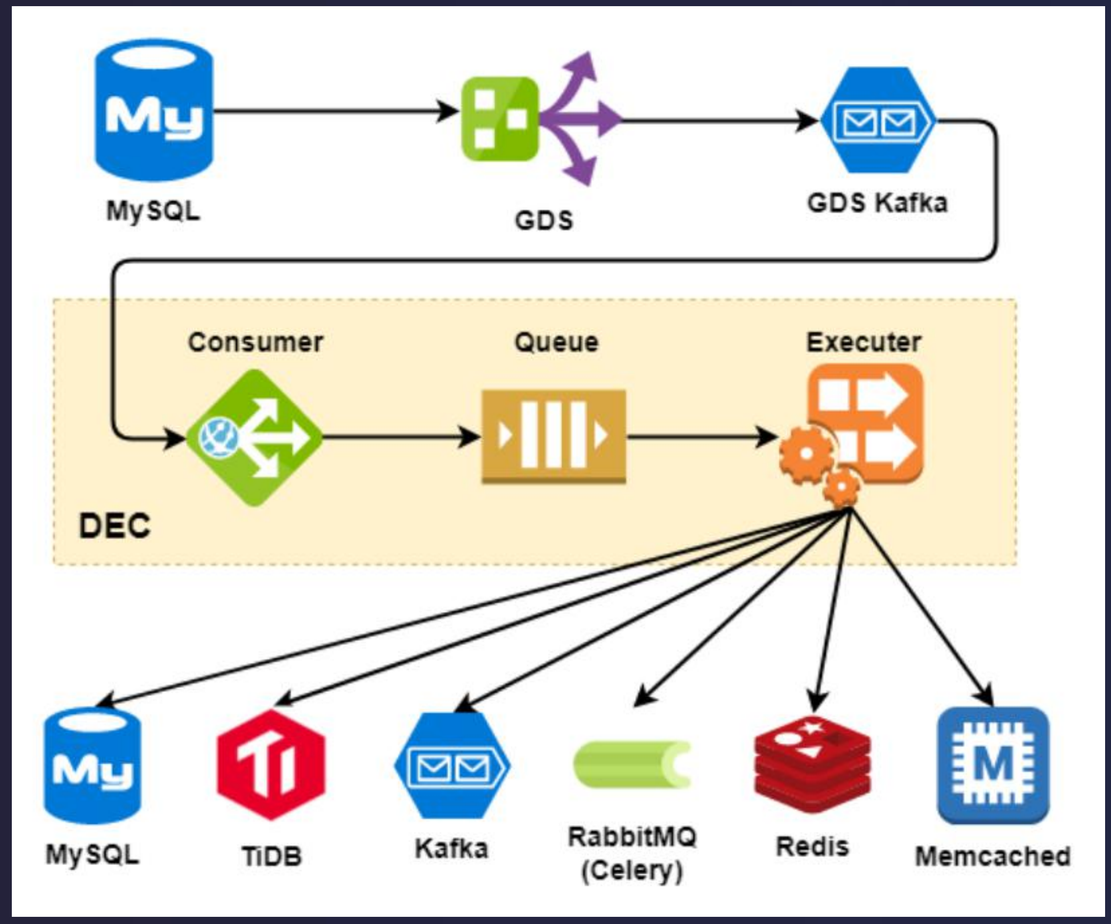
- **分库分表**
- **字段映射**
- **类型转换**
- **简单数据处理**
- **JSON / Protobuf 解析和生成**

系统架构

Consumer
消费 GDS 事件，根据配置解析和过滤数据，生成任务数据

Queue
内部任务消息队列 (Kafka)

Executer
执行任务



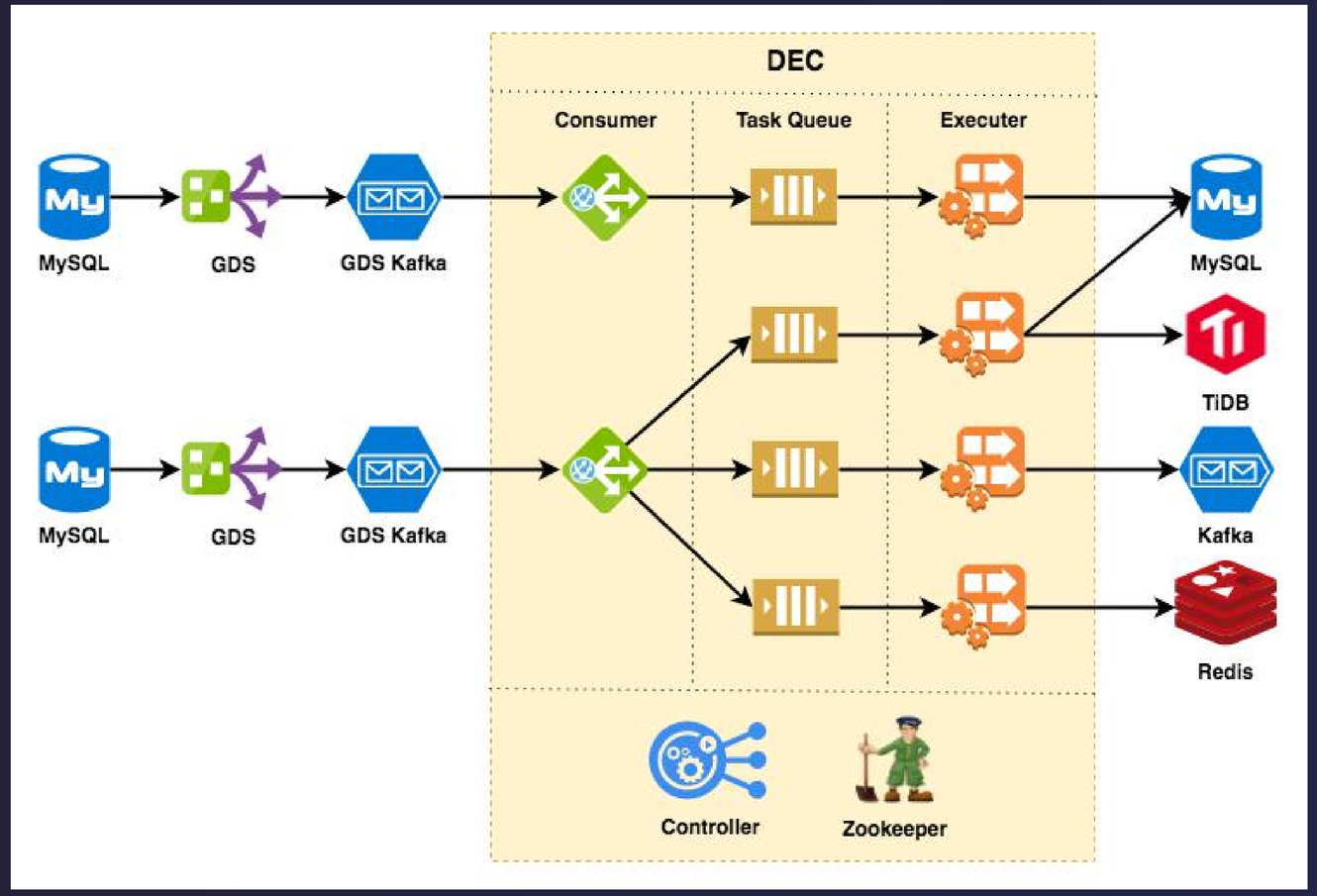
系统架构

Consumer 保证效率

- GDS 每个数据库一个队列
- Kafka 消费不支持过滤
- 同一源数据库的任务由一个 Consumer 统一处理

Executer 保证稳定性

- 每个任务都有独立的队列和 Executer
- Executer 互不影响



控制中心

Controller

- 管理调度 Consumer 和 Executer
- 监控任务状态

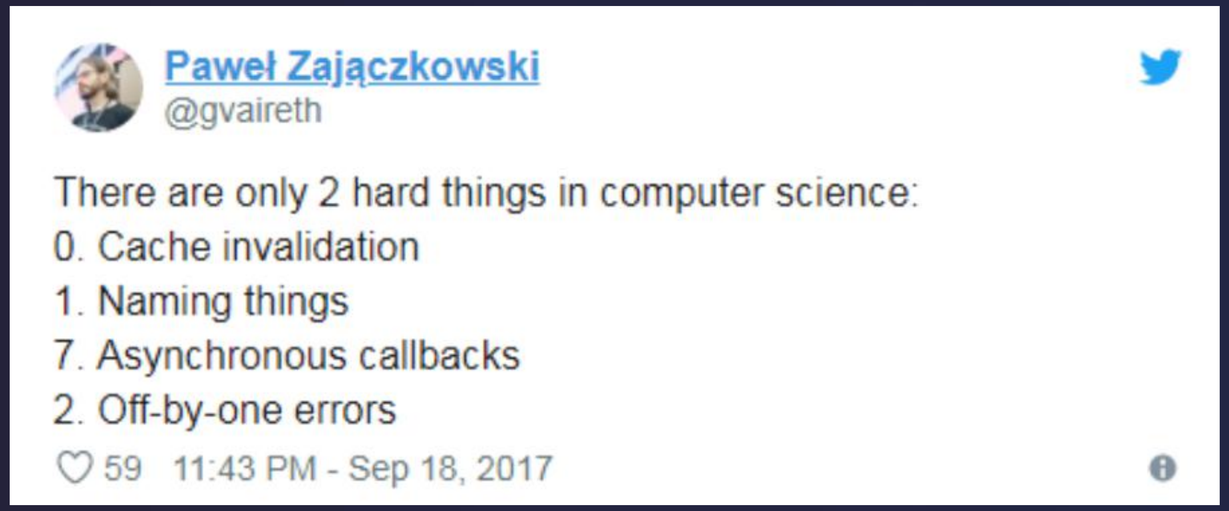
ZooKeeper

- 保存任务配置
- 记录任务执行状态（当前消息位置）

高性能

Kafka / MySQL 单线程同步读写存在瓶颈

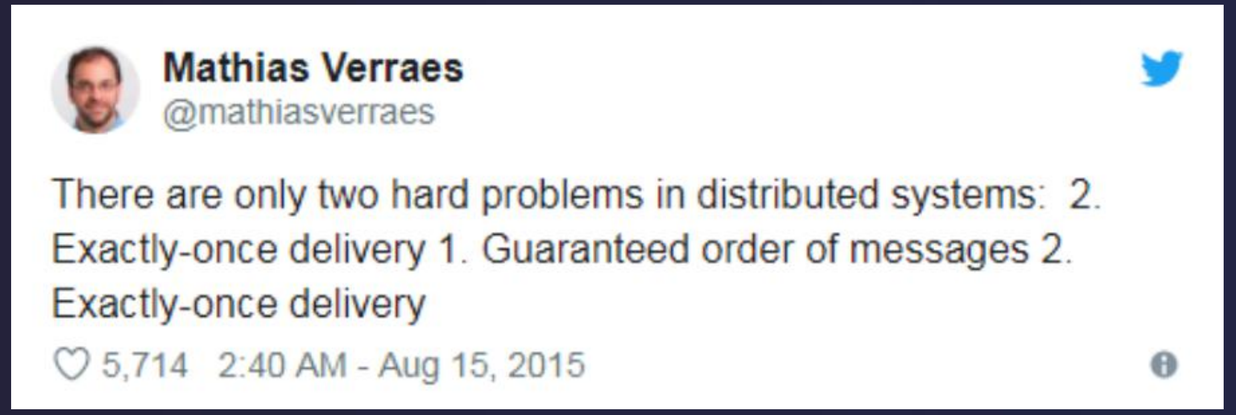
- **多 Goroutine 并发读写**
- **异步写入**
- **如何保证一致性？**



一致性

可选的一致性保证

- 顺序性
- At Least Once
- At Most Once
- Exactly Once



一致性

顺序性

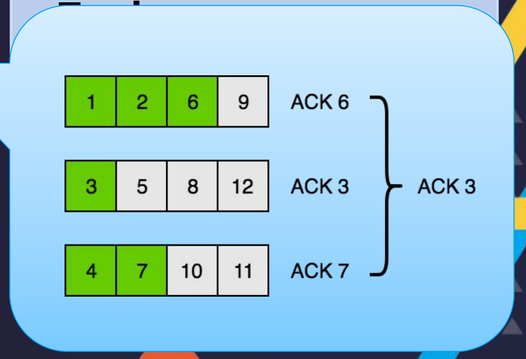
- 可以并发处理消息
- 写入时排队
- 或选择保证同一 Key 的消息在同一 Goroutine 中顺序处理

At Least Once

- 适用于满足幂等性的任务
- 写入失败重试
- 收到 ACK 后将偏移写入 Zookeeper
- 并发读写时记录 ACK 消息偏移最小值，不保证顺序

At Most Once

- 失败不重试
- 投递消息前将偏移写入



Exactly Once

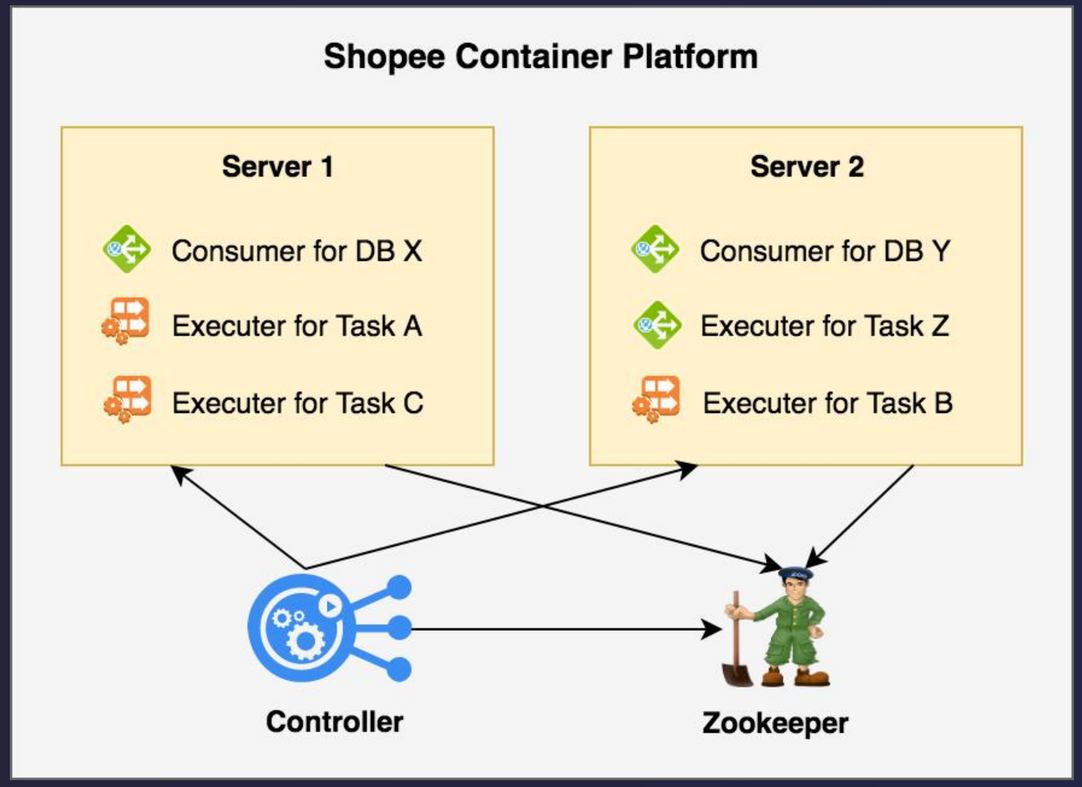
- 暂未实现
- 严格实现需要目标存储支持事务
- GDS 生成消息唯一 ID
- GDS 和 Consumer 实现 At Least Once
- 在 Executer 部署 Redis 进行消息去重检查

事务

- 一个任务可以包含多个操作
- 可选择操作依次执行或同时执行
- 可选择任务或操作失败时的重试策略
  - 跳过（不支持操作回滚）
  - 有限次数重试
  - 不断重试

可用性

- Controller 在容器平台上创建和管理 Consumer 和 Executer
- 容器平台保证 Consumer 和 Executer 的可用性，健康检查失败时自动重启
- 内部任务队列依赖 Kafka 自身的可用性



伸缩性

- 多个 Goroutine 并发执行任务
  - 写入时排队
  - 根据 Key 分发任务
- 暂不支持多个节点执行同一任务

实际应用和遇到的问题

实际应用

目前已有 20+ 任务运行在 DEC 上

- 分库分表数据的索引表维护
- 异构数据库同步
- 数据库重构过程中新旧数据表同步
- 数据修改记录的审计表维护
- 数据变更触发回调任务
- 缓存更新

单任务最高 10000+ TPS

分库分表数据的索引表维护

订单表

- order_{order_id/1000000}
  - 主键: order_id
  - 字段: user_id, shop_id, status, data

DEC

订单索引表

- order_user_{user_id % 1000}
  - 索引: user_id
  - 字段: order_id
- order_shop_{shop_id % 1000}
  - 索引: shop_id
  - 字段: order_id

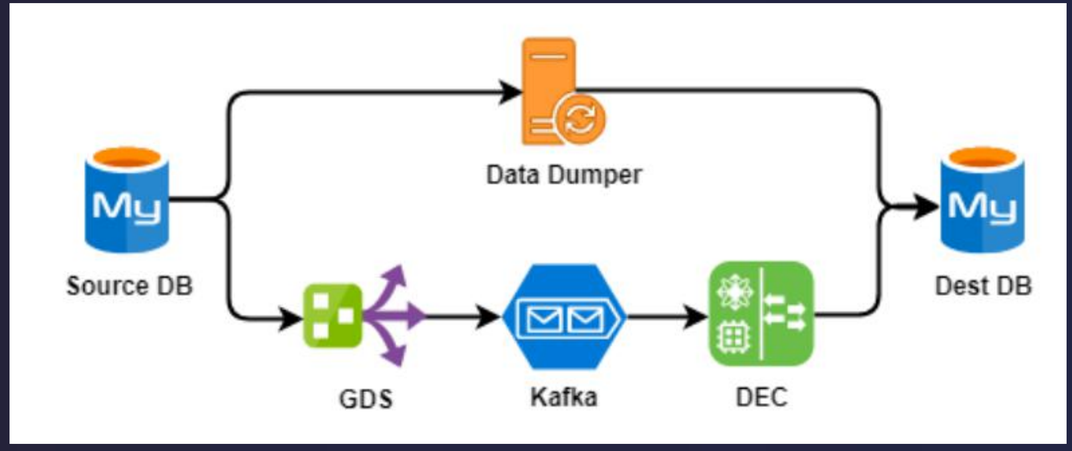
数据变更触发回调任务

问题一：数据表同步初始化

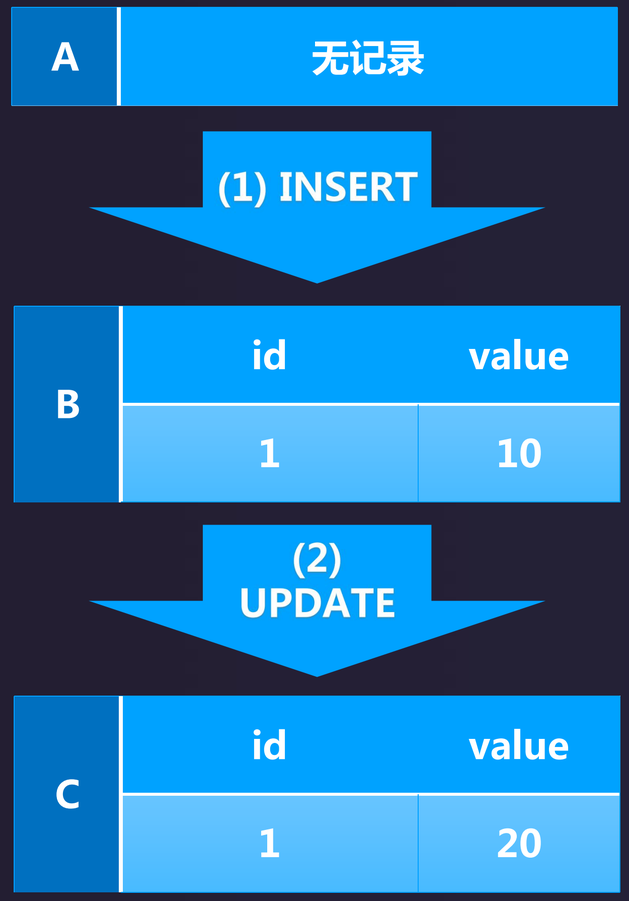
DEC 可以实现数据表的实时同步，但业务通常要求同步已有数据表到新表，这时已有的数据需要手动导入，分别操作可能导致数据不一致

解决方案：

- **实现通用的数据库导出导入工具**
- **先运行 DEC 同步，再执行数据导入，保证最终一致性**



问题一：数据表同步初始化



DEC

- INSERT -> INSERT IGNORE
- UPDATE -> UPSERT (INSERT ... ON DUPLICATE KEY UPDATE ...)

数据导入工具

- INSERT IGNORE

CASE 1	CASE 2	CASE 3	CASE 4
导入 (A) √ -	导入 (B) √ INSERT IGNORE id=1, value=10	导入 (C) √ INSERT IGNORE id=1, value=20	DEC (1) √ INSERT IGNORE id=1, value=10
DEC (1) √ INSERT IGNORE id=1, value=10	DEC (1) × INSERT IGNORE id=1, value=10	DEC (1) × INSERT IGNORE id=1, value=10	DEC (2) √ UPSERT id=1,value=20
DEC (2) √ UPSERT id=1,value=20	DEC (2) √ UPSERT id=1,value=20	DEC (2) √ UPSERT id=1,value=20	导入 (B) × INSERT IGNORE id=1, value=10

问题二：Consumer 无法隔离

同一数据库的多个任务共享 Consumer，无法单独停止或重放单个任务

解决方案：

- **允许任务使用独立 Consumer 执行**

问题三：Kafka 卡住

Kafka 0.11.0 修改消息格式导致客户端卡住

原因：客户端最大请求大小大于 Kafka 服务端 max.message.bytes

解决方案：

- **设置客户端最大请求大小小于 max.message.bytes**

https://kafka.apache.org/documentation/#upgrade_11_message_format

One of the notable differences in the new message format is that even uncompressed messages are stored together as a single batch. This has a few implications for the broker configuration max.message.bytes, which limits the size of a single batch. First, if an older client produces messages to a topic partition using the old format, and the messages are individually smaller than max.message.bytes, the broker may still reject them after they are merged into a single batch during the up-conversion process. Generally this can happen when the aggregate size of the individual messages is larger than max.message.bytes. There is a similar effect for older consumers reading messages down-converted from the new format: if the fetch size is not set at least as large as max.message.bytes, the consumer may not be able to make progress even if the individual uncompressed messages are smaller than the configured fetch size. This behavior does not impact the Java client for 0.10.1.0 and later since it uses an updated fetch protocol which ensures that at least one message can be returned even if it exceeds the fetch size. To get around these problems, you should ensure 1) that the producer's batch size is not set larger than max.message.bytes, and 2) that the consumer's fetch size is set at least as large as max.message.bytes.

问题四：任务延迟

Kakfa 读写延迟导致任务整体延迟高

解决方案：

- **异步读写**
- **调优 Kafka 生产者和消费者参数**

```
// Producer
queue.buffering.max.messages = 100000
queue.buffering.max.ms = 2

// Consumer
fetch.min.bytes = 100000
fetch.wait.max.ms = 10
```

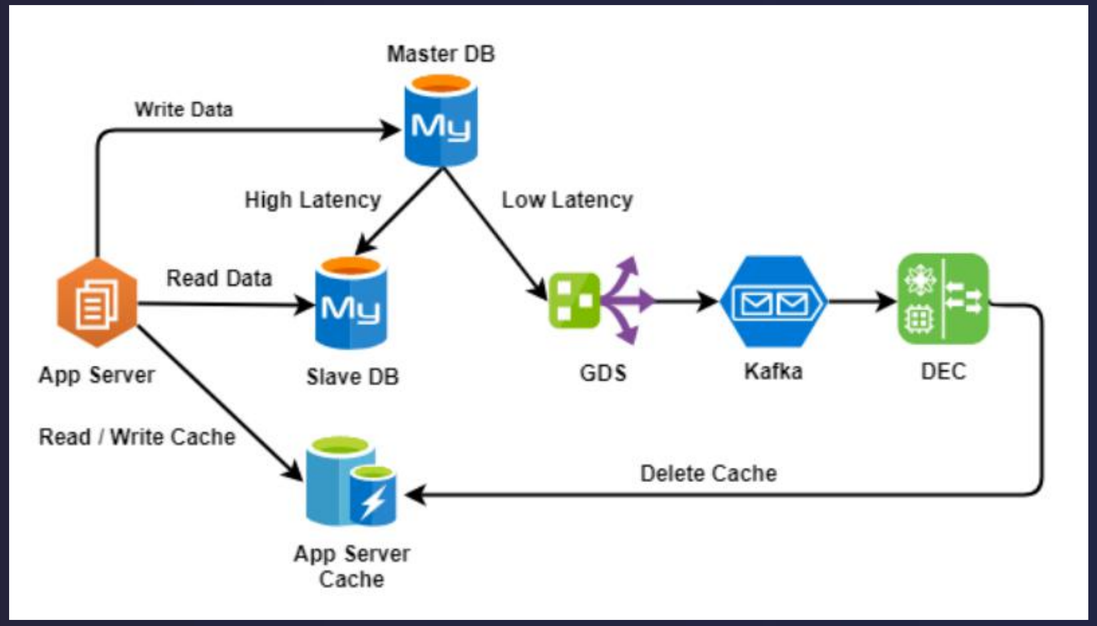
目前任务延迟在 20ms 以内

问题五：Slave 延迟不一致

如果 GDS Replication 延迟低于服务使用的 Slave 延迟，会导致缓存清除后，服务从 Slave 仍然读到旧数据写入缓存

解决方案：

- **部署额外一套 GDS 挂载在主从延迟最高的 Slave 下**
- **重复清理缓存**



未来规划

未来规划

极客时间全部课程任学

喊老板来买单！

- 精选 13+ 热门职位的学习路径，包括架构、运维、前端工程师等
- 根据不同技术岗位能力模型匹配合适的课程
- 一键设置购买条件，成员按需选课，自主制定学习计划
- 享充值满赠优惠，帮老板省钱，团队免费学习

立即申请

Geekbang 极客邦科技 | TGO 鲲鹏會

TGO鲲鹏会

汇聚全球科技领导者的高端社群

全球12大城市

850+高端科技领导者

使命 Mission

为社会输送更多优秀的科技领导者

愿景 Vision

构建全球领先的有技术背景优秀人才的学习成长平台

扫描二维码，了解更多内容

THANKS

Geekbang. 极客邦科技 InfoQ

林锋

Shopee