MongoDB在vivo评论中台的应用案例

作者：vivo互联网技术

原文：https://mongoing.com/archives/81631

1.业务背景

随着公司业务发展和用户规模的增多，很多项目都在打造自己的评论功能，而评论的业务形态基本类似。当时各项目都是各自设计实现，存在较多重复的工作量；并且不同业务之间数据存在孤岛，很难产生联系。因此我们决定打造一款公司级的评论业务中台，为各业务方提供评论业务的

快速接入能力。在经过对各大主流app评论业务的竞品分析，我们发现大部分评论的业务形态都具备评论、回复、二次回复、点赞等功能。具体如下图所示：

涉及到的核心业务概念有：

【主题 topic】 评论的主题，商城的商品、应用商店的app、社区的帖子。

【评论 comment】用户针对于主题发表的内容。

【回复 reply】用户针对于某条评论发表的内容，包括一级回复和二级回复。

2.为什么选择MongoDB

团队在数据库选型设计时，对比了多种主流的数据库，最终在MySQL和MongoDB两种存储之进行抉择。

Mysql：	MongoDB：
优势： • 关系型数据库，严格的ACID，事务支持好 • 结构化查询语言（SQL），查询便捷 • 成熟，坐拥庞大的社区，稳定性好	优势： • 类关系型数据库，无固定行列，易扩展，bson文档方式存储，支持文档嵌套 • 硬盘+内存方式存储，读写速度优秀 • 高可用，复制集方式数据冗余，自动读写分离，自动故障恢复 • mongodb集群，支持一键水平扩展和自动迁移
劣势： • 严格的scheme，结构化存储，元数据管理麻烦，不利于动态扩展 • 写入和查询速度相对于nosql较为一般 • 海量数据的存储支持较差，需要进行水平拆分或垂直拆分 • 水平扩展和迁移需要借助第三方工具，成本高，运维难	劣势： • 事务支持性较差，只保证最终一致性 • 不支持join等关联查询操作 • 社区成熟度、文档丰富度不如mysql

关于MongoDB事务及一致性部分，官方专家补充：

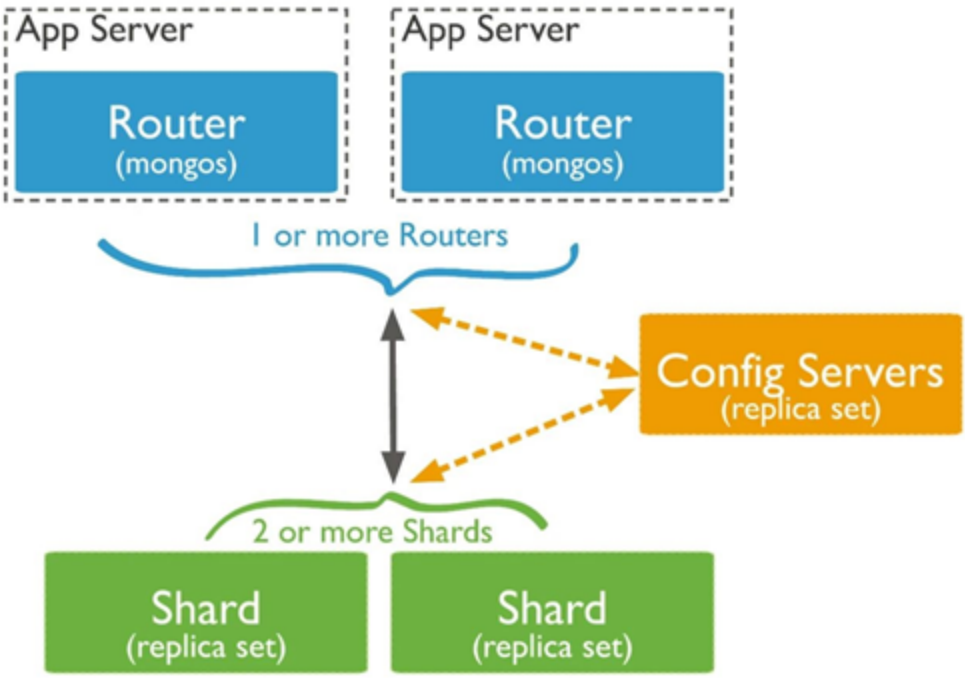
简单评价一下，ACID事务是在MongoDB 4.2开始支持的新特性。现在已经5.0了，应该来说还是比较稳定的。 一致性的问题没这么简单能说清楚。MongoDB支持的是可调一致性，根据你的需要**调整一致性的强度。**有一篇论文可以参考下（http://www.vldb.org/pvldb/vol12/p2071-schultz.pdf），以前社区成员也有分享过解读（https://mongoing.com/archives/77853）。
简单地说结论，在分布式环境下要求强一致是不现实并且不必要的。可调一致性解决了绝大部分场景的问题。 至于join的问题，join跟分布式是矛盾的存在。不要指望在分布式环境中join能工作得很好，谁都做不到（少数特殊场景除外）。所以你要考虑的是：分布式和JOIN你要哪一个？另外，很多场景下的JOIN其实根本不必要，只是范式用习惯了可能根本没注意到这点。关于这个我年前在线上讲过关于数据模型设计的讲座，也可以看下（https://mongoing.com/archives/81483）。
由于评论业务的特殊性，它需要如下能力:
【字段扩展】业务方不同评论模型存储的字段有一定差异，需要支持动态的自动扩展。
【海量数据】作为公司中台服务，数据量随着业务方的增多成倍增长，需要具备快速便捷的水平扩展和迁移能力。
【高可用】作为中台产品，需要提供快速和稳定的读写能力，能够读写分离和自动恢复
而评论业务不涉及用户资产，对事务的要求性不高。因此我们选用了MongoDB集群作为了最底层的数据存储方式。

3.MongoDB在评论中台的应用

3.1 MongoDB集群知识

集群架构

由于单台机器存在磁盘/IO/CPU等各方面的瓶颈，所以MongoDB提供集群方式的部署架构，如图所示：



主要由以下三个部分组成：

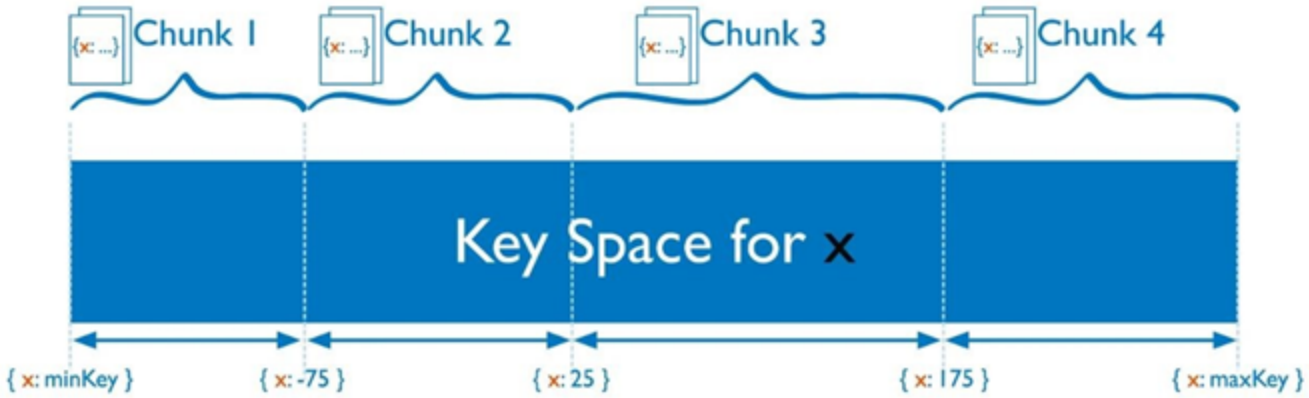
mongos：路由服务器，负责管理应用端的具体链接。应用端请求到mongos服务后，mongos把具体的读写请求转发应的shard节点上执行。一个集群可以有1~N个mongos节点。

config：配置服务器，用于分存储分片集合的元数据和配置信息，必须为复制集(关于复制集概念戳我) 方式部署。mongos通过config配置服务器合的元数据信息。

shard：用于存储集合的分片数据的MongoDB服务，同样必须以复制集方式部署。

片键

MongoDB数据是存在collection(对应MySQL表)中。集群模式下，collection按照片键（shard key）拆分成多个区间，每个区间组成一个chunk，按照规则分布在不同的shard中。并形成元数据注册到config服务中管理。



分片键只能在分片集合创建时指定，指定后不能修改。
分片键主要有两大类型：

- hash分片：通过hash算法进行散列，数据分布的更加平均和分散。支持单列和多列hash。
- 范围分片：按照指定片键的值分布，连续的key往往分布在连续的区间，更加适用范围查询场景。单数据散列性由分片键本身保证。

3.2 评论中台的实践

片键的选择

MongoDB集群中，一个集合的数据部署是分散在多个shard分片和chunk中的，而我们希望一个评论列表的查询最好只访问到一个shard分片，因此确定了范围分片的方式。

起初设置只使用单个key作为分片键，以comment评论表举例，主要字段有{“_id”:唯一id,”topicId”:主题id,”text”:文本内容,”createDate”:时间},考虑到一个主题id的评论尽可能连续分布，我们设置的分片键为topicId。随着性能测试的介入，我们发现了有两个非常致命的问题：

- jumbo chunk问题
- 唯一键问题

jumbo chunk：
官方文档中，MongoDB中的chunk大小被限制在了1M-1024M。分片键的值是chunk划分的唯一依据，在数据量持续写入超过chunk size设定值时， MongoDB集群就会自动的进行分裂或迁移。而对于同一个片键的写入是属于一个chunk，无法被分裂，就会造成 jumbo chunk问题。
举例：若我们设置1024M为一个chunk的大小，单个document 5KB计算，那么单个chunk能够存储21W左右document。考虑热点的主题评论(如微信评论)，评论数可能达到40W+，因此单个chunk很容易超过1024M。超过最大size的chunk依然能够提供读写服务，只是不会再进行分裂和迁移，长久以往会造成集群之间数据的不平衡。

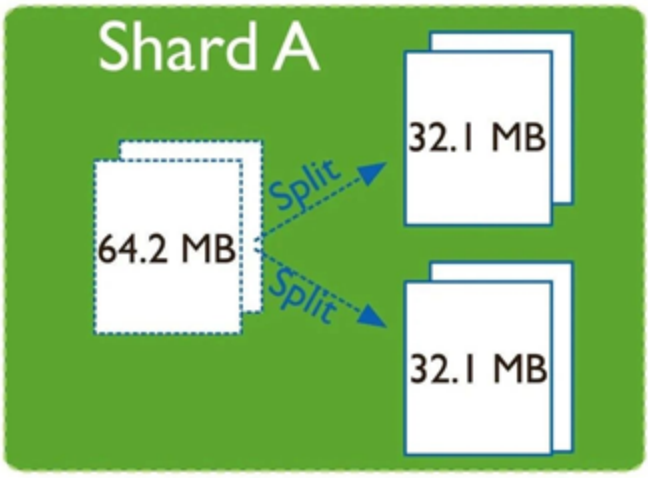
唯一键问题：
MongoDB集群的唯一键设置增加了限制，必须是包含分片键的；如果_id不是分片键，_id索引只能保证单个shard上的唯一性。

- You cannot specify a unique constraint on a hashed index
- For a to-be-sharded collection, you cannot shard the collection if the collection has other unique indexes
- For an already-sharded collection, you cannot create unique indexes on other fields

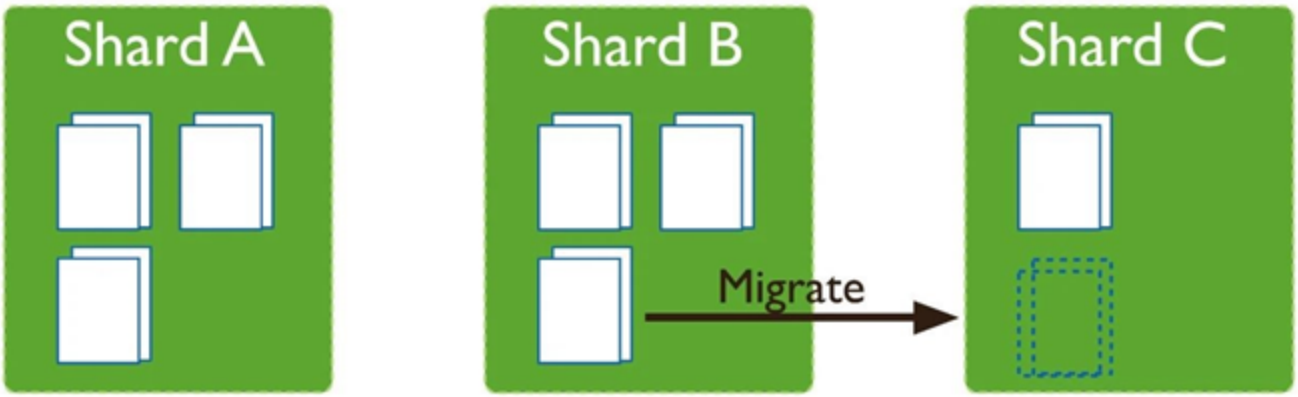
因此我们删除了数据和集合，调整topicId 和 _id 为联合分片键 重新创建了集合。这样即打破了chunk size的限制，也解决了唯一性问题。

迁移和扩容

随着数据的写入，当单个chunk中数据大小超过指定大小时(或chunk中的文件数量超过指定值)。MongoDB集群会在插入或更新时，自动触发chunk的拆分。



拆分会导致集合中的数据块分布不均匀，在这种情况下，MongoDB balancer组件会触发集群之间的数据块迁移。balancer组件是一个管理数据迁移的后台进程，如果各个shard分片之间的chunk数差异超过阈值，balancer会进行自动的数据迁移。



balancer是可以在线对数据迁移的，但是迁移的过程中对于集群的负载会有较大影响。一般建议可以通过如下设置，在业务低峰时进行。

```
db.settings.update(
{ _id: "balancer" },
{ $set: { activeWindow : { start : "<start-time>", stop : "<stop-time>" } } },
{ upsert: true }
```

)

MongoDB的扩容也非常简单，只需要准备好新的shard复制集后，在mongos节点中执行：

`sh.addShard("<replica_set>/<hostname><:port>")`

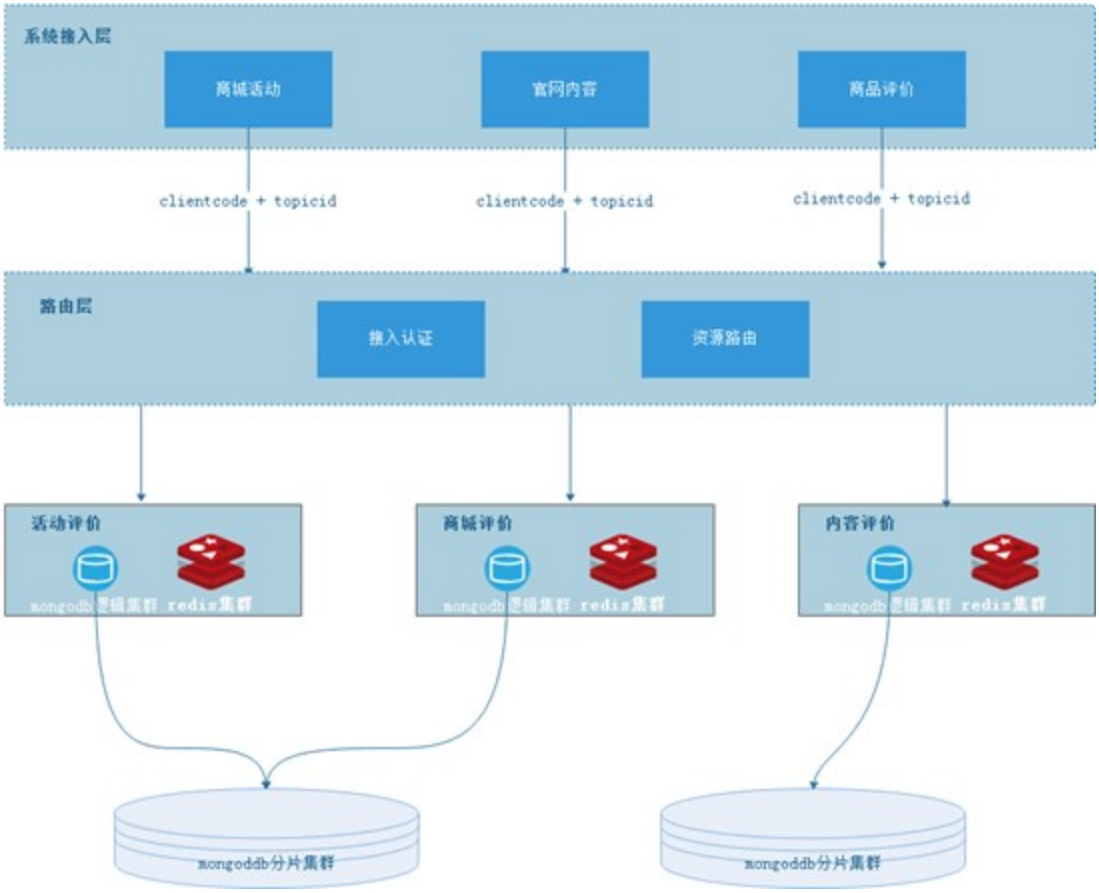
扩容期间因为chunk的迁移，同样会导致集群可用性降低，因此只能在业务低峰进行。

集群的扩展

作为中台服务，对于不同的接入业务方，通过表隔离来区分数据。以comment评论表举例，每个接入业务方都单独创建一张表，业务方A表为 comment_clientA，业务方B表为comment_clientB，均在接入时创建表和相应索引信息。但只是这样设计存在几个问题：

- 单个集群，不能满足部分业务数据物理隔离的需要
- 集群调优(如split迁移时间)很难业务特性差异化设置
- 水平扩容带来的单个业务方数据过于分散问题

因此我们扩展了MongoDB的集群架构：



1. 扩展后的评论MongoDB集群 增加了 【逻辑集群】和【物理集群】的概念。一个业务方属于一个逻辑集群，一个物理集群包含多个逻辑集群。
2. 增加了路由层设计，由应用负责扩展spring的MongoTemplate和连接池管理，实现了业务到MongoDB集群之间的切换选择服务。
3. 不同的MongoDB分片集群，实现了物理隔离和差异调优的可能。

3.3 自研MongoDB事件采集处理平台及应用

评论中台中有很多统计数据的查询：评论数、回复数、点赞数、未读回复数等等，由于评论数据量较大，单个业务的数据量都在亿级别以上，浏览器、短视频等内容型的业务评论数据量都是在百亿级别，前台业务查询时对数据库做实时count性能肯定是无法达到要求的，因此我们选择使用空间换时间的方式，在数据表中增加相关的统计数据字段，每次有新的评论发表或者状态变更时对该字段的值做$inc原子操作。这种方式确实能够提升数据查询的效率，在评论这种读多写少的场景下十分合适。

带来的问题：

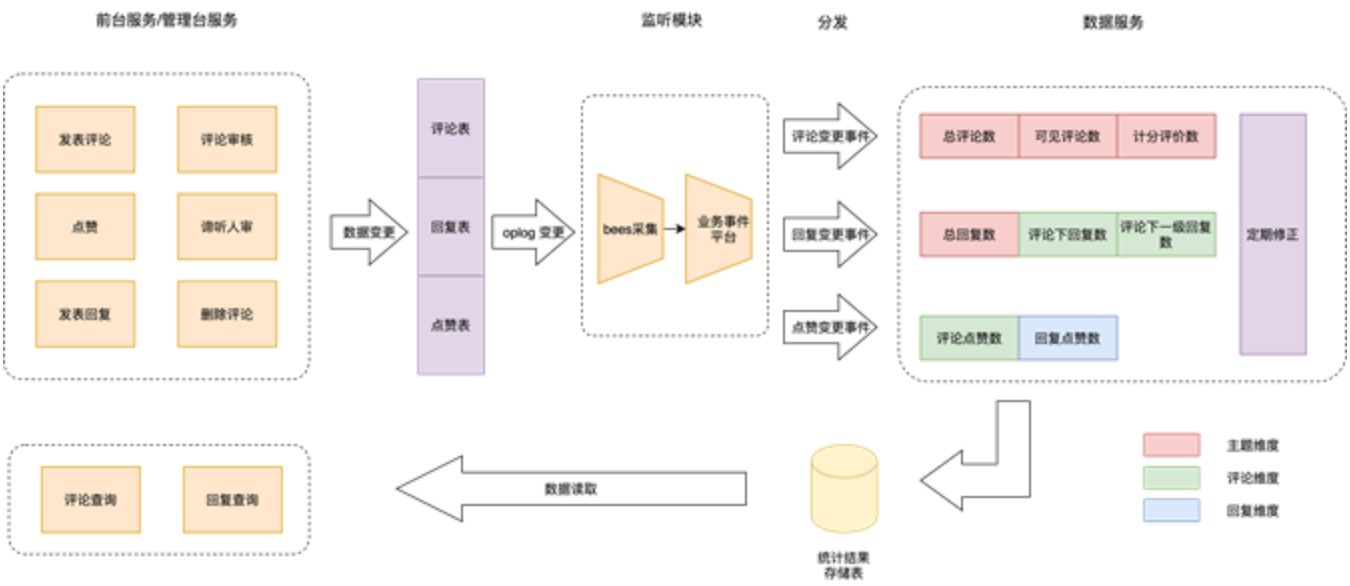
作为中台类项目，需要适配vivo不同的评论业务场景，随着业务发展，这部分统计口径和类型也会随之变动，各种统计类型越来越多（目前已有20多种统计），在代码主流程中耦合的统计逻辑越来越越重，并且各种统计逻辑的代码散落在系统代码的各个角落，系统也越来越臃肿，并且代码变更时很容易带来统计数据不准的问题，这类问题会影响评论及回复的下拉展示的效果，举一个较为典型的例子，如果当前评论下有1条回复，但是统计错误，误认为没有回复，则当前人回复信息则无法显示，如果需要修复此类问题，我们发现涉及面比较广，无法收敛。因此我们对统计相关的逻辑进行了一次大的重构。

事件数据服务

我们选择事件驱动的方式将统计逻辑从主流程中解构出来，每一个统计都有其对应的变更事件，例如：

- 评论发表事件 > 评论数统计点赞、
- 取消点赞事件 > 点赞数统计
- 仅自身可见事件 > 用户仅自身可见数统计

但是这个事件如何发出来呢？一开始我们想的是在主流程中根据场景发送不同类型的事件，但是这种方式不够灵活，每次增加一种统计类型就需要增加一种事件类型。转换下思路，其实每发出一种事件其本质上是数据库某些数据发生了变化，那我们为何不监听数据库的操作日志，采集数据的变更，通过对比变更前后的数据来自定义事件，进而进行数据的统计，整体流程如下：



其中监听模块负责监听MongoDB数据变更，并对变更事件进行处理和分发，当被监听的表发生数据变更时会将变更前后的数据通过mq的方式分发给数据服务。数据服务专门负责评论中台数据统计、数据刷新等功能，这些功能对系统负载影响比较大，需要和主系统解耦。

从上图可以看到，整个监听模块包括了”bees采集“和”业务事件平台“两个部分，一个负责采集，一个负责对事件进行处理和分发，这里为什么还需要一个独立的平台进行事件处理和分发，有2个

原因：

1. 直接采集的事件，其事件内容依然无法满足业务需求，无法满足对比变更前后的数据来自定义事件，因此需要对相应的事件数据进行合并，具体的实现在后文进行介绍。
2. 变更事件需要按照业务方自定义的条件进行过滤，只需要将满足条件的事件分发给下游数据服务。

因此这里复用了vivo自研的业务事件平台，通过low Code方式对事件数据进行处理和过滤后，发给下游的数据服务，整个流程为bees采集—>事件平台—>评论数据服务，接下来我们先介绍下MongoDB的采集系统部分。

MongoDB采集架构和流程

对于MongoDB的采集，我们复用了vivo自研的在Bees大数据采集平台能力，基于这个平台的DB数据采集的子组件（bees-dbsync），我们开发了用于MongoDB的数据采集的链路，适用于实时获取MongoDB的变更信息，实现了对于MongoDB的全量和增量的采集能力。

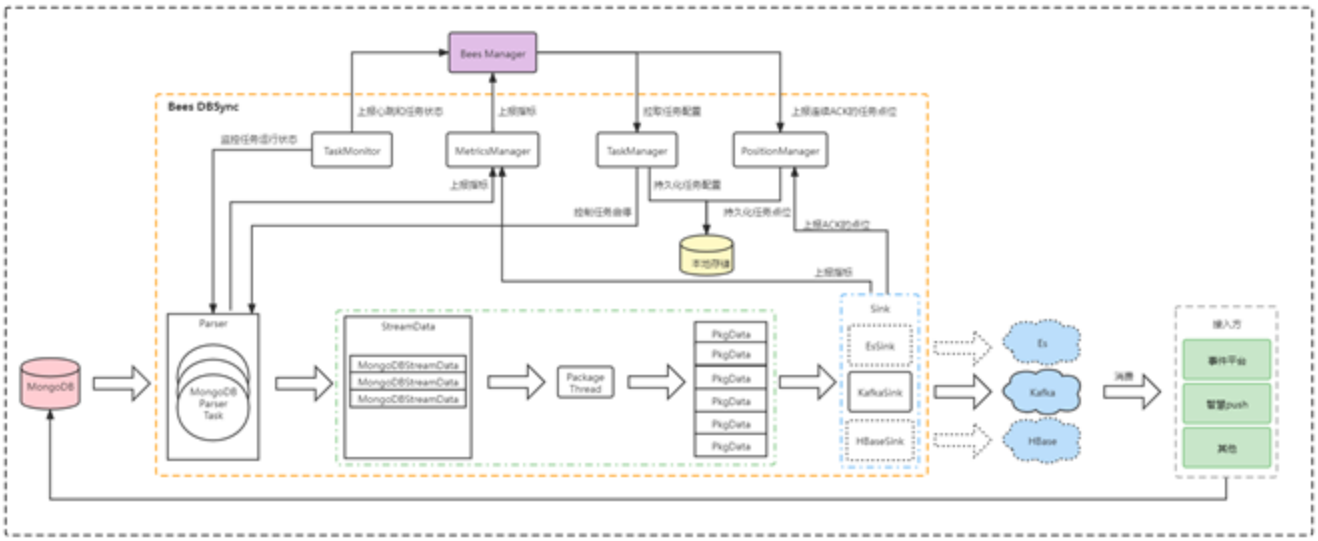
功能介绍

现阶段MongoDB采集模块可以支持的功能如下：

1. 支持在任务接入后进行一次全量采集，全量采集结束后会根据用户配置的延迟时间开启增量采集。
2. 支持针对单一任务的启动、停止、修改等操作。操作完成后会实时进行相应的变更。
3. 增量采集断点续传功能。在任意时刻停止任务后进行恢复采集，任务将由上一次完成的采集点位后一个点位开始继续进行采集。在oplog大小保证数据完整的条件下，不会有数据漏采的情况发生。
4. 全量、增量采集均支持按照用户自定义的kafka分区发送规则发送至相应的分区。
5. 全量、增量采集均支持任务状态和数据量监控。
6. 全量、增量采集均支持根据副本集、sharding变化自动开启采集。

方案架构

在采集系统中，关于MongoDB的采集模块架构图如下：



采集系统中，MongoDB采集模块内部结构如上图橙色方框内所示。主要包括：

- MongoDBParser首先将对数据进行解析和过滤。对于MongoDB全量数据采集，MongoDBParser将直接对全表执行find()操作，对于MongoDB增量数据采集，MongoDBParser将读取local库下的oplog.rs表，根据递增的ts值依次获取新增的oplog事件，由于ts值的唯一性，已经发送完成的 ts值将作为历史点位信息保存在Bees-DBSync本地并上传到Bees-Manager的数据库中，成为断点续传功能的必须条件。
- StreamData，具体由MongoDBStreamData实现，负责存放Bees-DBSync从业务MongDB解析后的数据。
- PackageThread，负责对MongoDBStreamData格式的数据进行任务信息的填充，并将数据打包成统一的PkgData数据格式，用以保证MongoDB、MySQL等发送格式的一致性。
- PkgData，是Bees-DBSync内部处理后统一的数据格式。
- KafkaSink模块，负责对目的端信息进行处理，根据用户配置的不同分区发送规则，将PkgData格式的数据发送到不同的Kafka分区。

数据完整性保障

借鉴了RingBuffer设计思路：

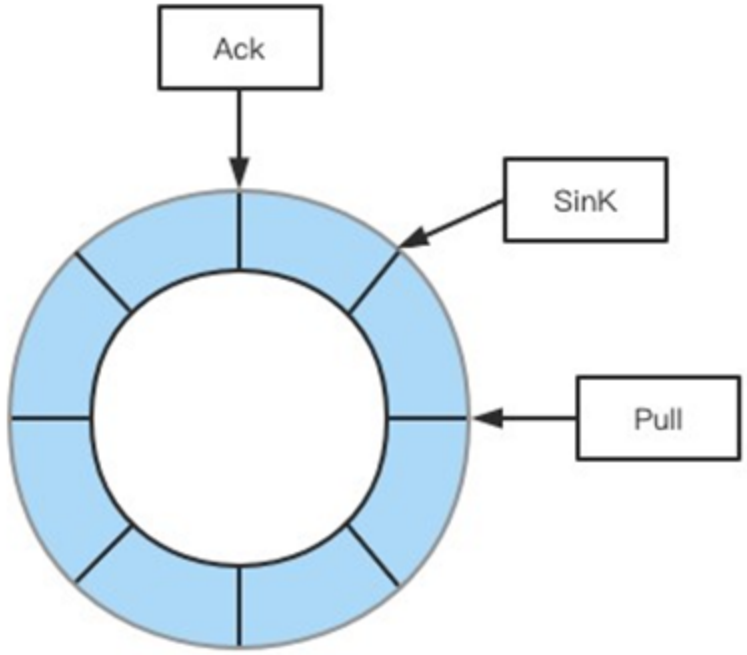
定义了三个点位：

- Pull：最后一次拉取oplog的点位
- Sink：最后一次发送Kafka的点位
- Ack：最后一次成功Sink到Kafka的点位

三个点位的关系是Ack <= Sink <= Pull，三个点位记录了日志采集情况，出现异常情况时可以从记录的点位恢复采集，最终保障数据完整性。

采集流程

MongoDB采集任务执行流程图如下：

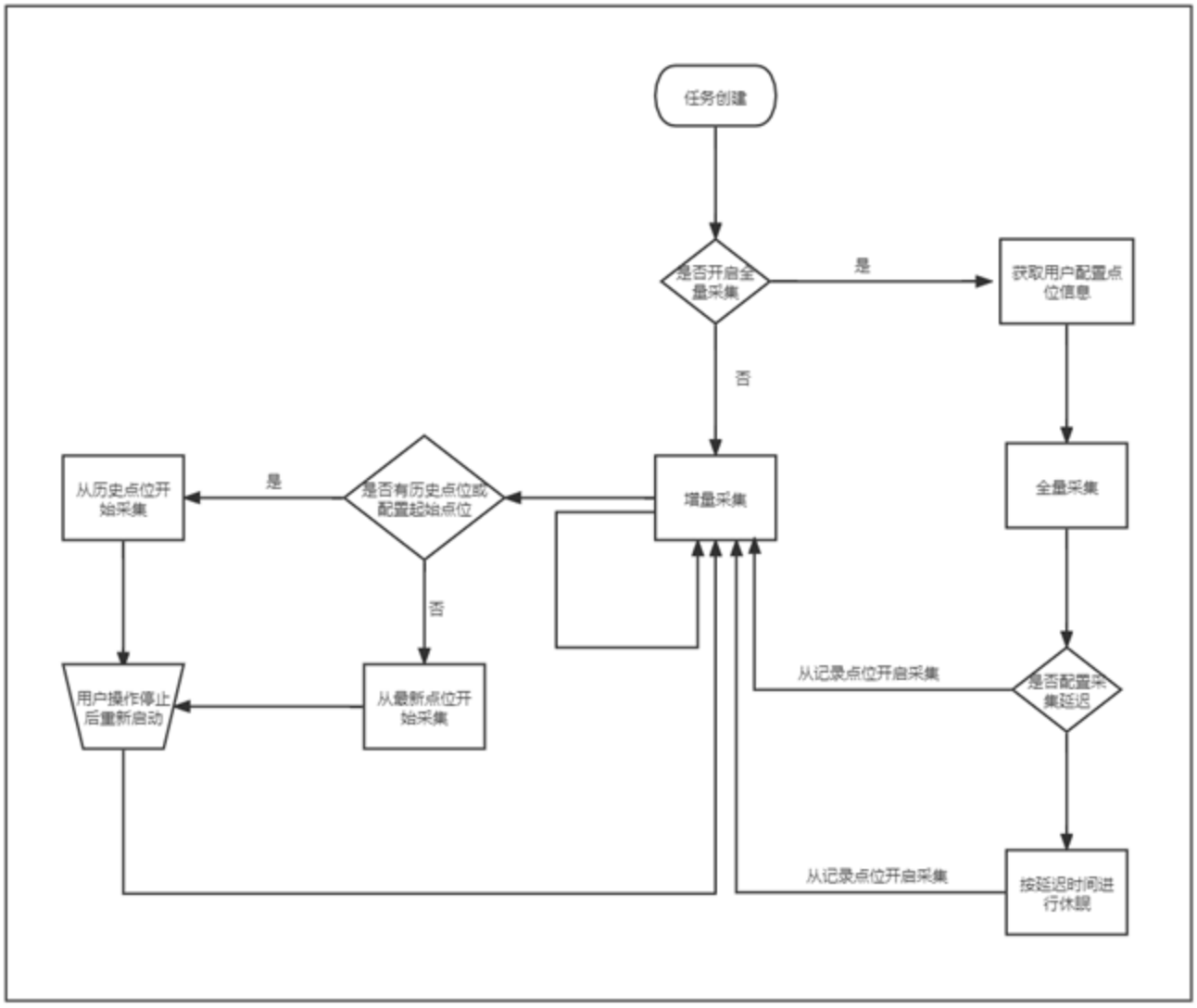


以上部分是采集系统的整理架构和流程，接下来介绍下，对变更事件进行合并和处理的架构和流程。

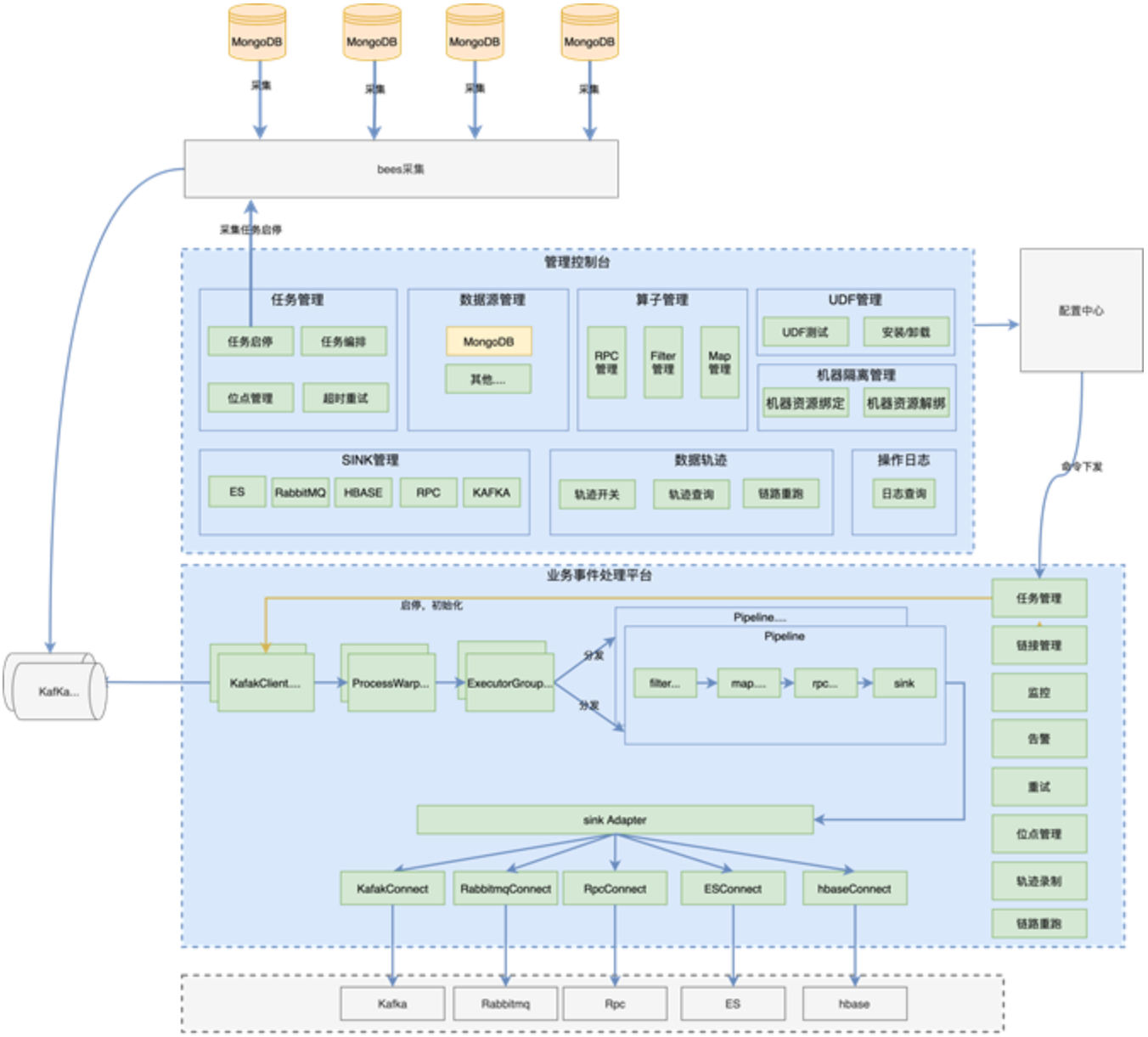
MongoDB变更事件处理平台

在这里我们复用了vivo自研的事件处理平台的能力，新增了对MongoDB变更事件的处理，这个平台的建设目的是面向我们的服务端开发同学，通过画布拖拽方式帮助业务对采集到的存储变更事件进行处理和分发。平台提供了一个低延迟的流式处理解决方案，支持画布的方式对采集到的MongoDB变更事件进行过滤（filter）、转化处理（map、udf）和输出（sink），支持UDF插件的方式自定义处理采集到的数据，并将处理的数据分发到不同的输出端。目前支持将MongoDB采集的事件通过filter、map后sink到es，hbase，kafka，rabbitmq以及通过dubbo方式通知到不同下游。

业务事件平台的整体架构图



在评论中台的业务场景中，评论中台需要依据MongoDB字段发生变更的条件来进行过滤和统计，例如，需要更加字段“commentNum”进行判断，当前字段的值是否是从少变多，是否数值有增加，这就需要根据MongoDB的变更前的值以及变更后的值进行比较，对符合评论业务场景条件的文档内容进行向下传递，因此我们需要捕获到MongoDB变更前的值，以及未变更字段的值。
如下图所示，需要集合对Oplog的变更事件的采集，组合成完整的文档内容，当前文档内容放在data属性下，old属性下为涉及到修改的字段在变更前的值。



事件平台系统架构图

但是我们发现，采集MongoDB的Oplog或者基于change stream都无法完全满足我们的需求。MongoDB原生的Oplog不同于binlog，Oplog只包含当前变更的字段值，Oplog缺少变更前的值，以及未变更字段的值，change stream也无法拿到变更前的值，基于这样的问题，我们引入Hbase字段多版本的特性，先预热数据到Hbase中，然后通过查询到的字段旧版本值，进行数据合并，构成完整的变更事件。

MongoDB事件数据合并流程

```
{
    "busiType": "UPDATE",
    "data": {
        "_id": "383981450485760",
        "canEvalNum": 21,
        "commentNum": 21,
        "topicGenre": "内容分发",
        "totalCommentNum": 21,
        "updateTime": 1638861970497
    },
    "old": {
        "canEvalNum": "20",
        "commentNum": "20",
        "totalCommentNum": "20",
        "updateTime": "1638860991695"
    },
    "table": "topic_info_clienttest380",
    "traceId": "7b5c357a4d99a88dea07f5670f91d2295e3ce604ea0e286480251af89ab3e6a0"
}
```


4.写在最后

MongoDB集群在评论中台项目中已上线运行了两年，过程中完成了约20+个业务方接入，承载了百亿级评论回复数据的存储，表现较为稳定。BSON非结构化的数据，也支撑了我们多个版本业务的快速升级。而热门数据内存化存储引擎，较大的提高了数据读取的效率。另外我们针对MongoDB实现了异步事件采集能力，进一步扩展了MongoDB的应用场景。

但对于MongoDB来说，集群化部署是一个不可逆的过程，集群化后也带来了索引，分片策略等较多的限制。因此一般业务在使用MongoDB时，副本集方式就能支撑TB级别的存储和查询，并非一定需要使用集群化方式。

以上内容基于MongoDB 4.0.9版本特性，和最新版本的MongoDB细节上略有差异。

关于作者：

vivo互联网技术：百灵评论项目开发团队，鲁班事件平台开发团队，bees数据采集团队

参考资料：

https://docs.mongodb.com/manual/introduction/