MongoShake详细文档

本文是关于MongoShake的详细文档，主要包括背景，简介，应用场景，功能介绍，架构设计，用户使用案例。

# 背景

在当前的数据库系统生态中，大部分系统都支持多个节点实例间的数据同步机制，如Mysql Master/Slave主从同步，Redis AOF主从同步等，MongoDB更是支持3节点及以上的副本集同步，上述机制很好的支撑了一个*逻辑单元*的数据冗余高可用。

跨逻辑单元，甚至跨单元、跨数据中心的数据同步，在业务层有时候就显得很重要，它使得同城多机房的负载均衡，多机房的互备，甚至是异地多数据中心容灾和多活成为可能。由于目前MongoDB副本集内置的主从同步对于这种业务场景有较大的局限性，为此，我们开发了MongoShake系统，可以应用在实例间复制，机房间、跨数据中心复制，满足灾备和多活需求。

另外，数据备份是作为MongoShake核心但不是唯一的功能。MongoShake作为一个平台型服务，用户可以通过对接MongoShake，实现数据的订阅消费来满足不同的业务场景。

# 简介

MongoShake是一个通用的平台型服务，通过读取MongoDB集群的Oplog操作日志，对MongoDB的数据进行复制，后续通过操作日志实现特定需求。日志可以提供很多场景化的应用，为此，我们在设计时就考虑了把MongoShake做成通用的平台型服务。通过操作日志，我们提供日志数据订阅消费PUB/SUB功能，可通过SDK、Kafka、MetaQ等方式灵活对接以适应不同场景(如日志订阅、数据中心同步、Cache异步淘汰等)。集群数据同步是其中核心应用场景，通过抓取oplog后进行回放达到同步目的，实现灾备和多活的业务场景。

# 应用场景举例

1. MongoDB集群间数据的异步复制，免去业务双写开销。
2. MongoDB集群间数据的镜像备份（当前1.0开源版本支持受限）
3. 日志离线分析
4. 日志订阅
5. 数据路由

根据不同的业务需求，结合日志订阅和过滤机制，可以获取关注的数据，达到数据路由的功能。

1. Cache同步

通过日志分析的结果，知道哪些Cache可以被淘汰，哪些Cache可以进行预加载，反向推动Cache的更新。

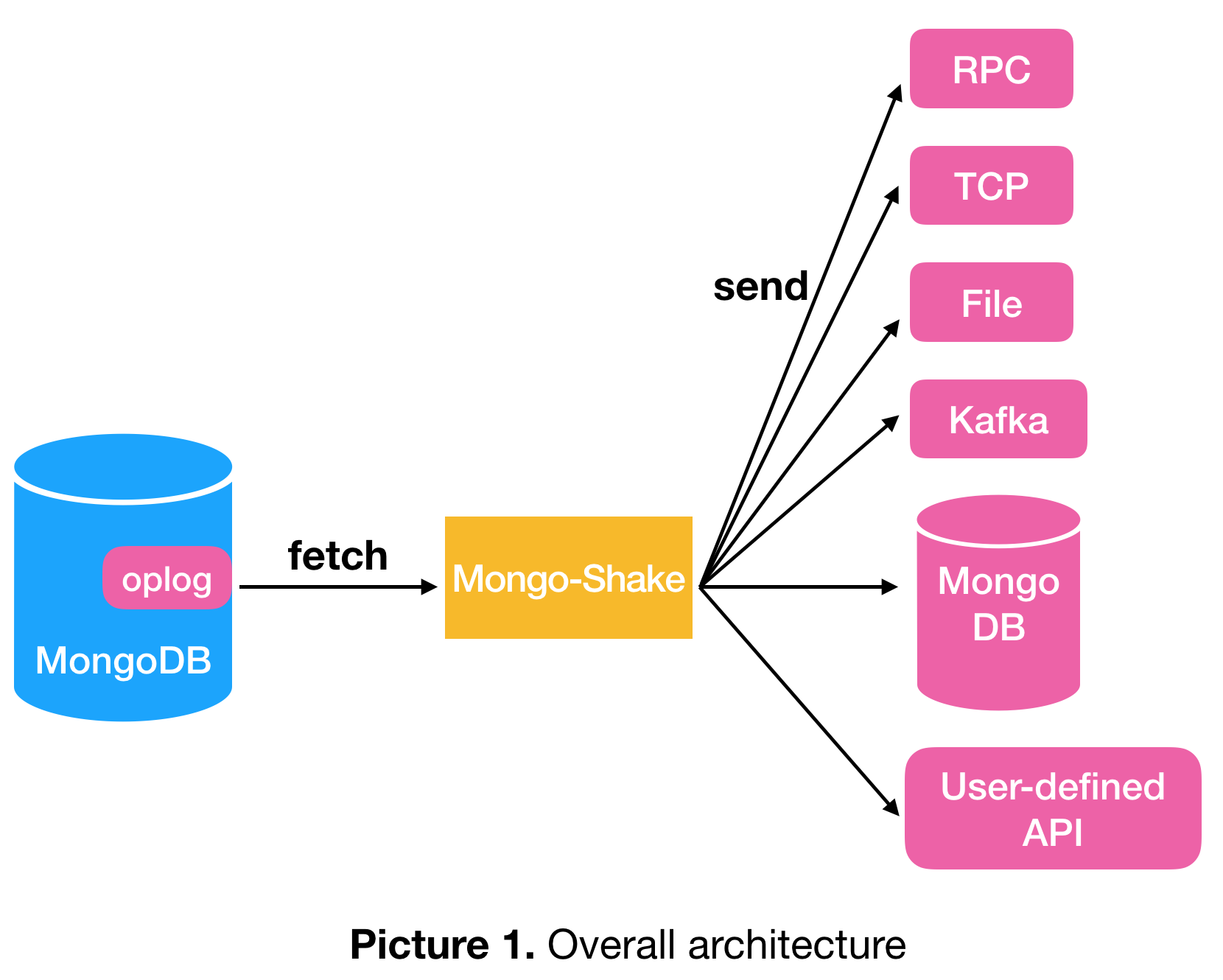
1. 基于日志的集群监控

# 功能介绍

MongoShake从源库抓取oplog数据，然后发送到各个不同的tunnel通道，消费者可以通过对接tunnel获取想要的数据，例如对接Direct通道直接写入目的MongoDB。现有通道类型有：

1. Direct：直接写入目的MongoDB
2. RPC：通过net/rpc方式连接
3. TCP：通过tcp方式连接
4. File：通过文件方式对接
5. Kafka：通过Kafka方式对接
6. Mock：用于测试，不写入tunnel，抛弃所有数据

此外，用户还可以自己创建自己的API进行灵活接入。图1给出了基本的架构数据流图。



MongoShake对接的源数据库支持单个mongod，replica set和sharding三种模式。目的数据库支持mongod和mongos。如果源端数据库为replica set，我们建议对接备库以减少主库的压力；如果为sharding模式，那么每个shard都将对接到MongoShake并进行并行抓取。对于目的库来说，可以对接多个mongos，不同的数据将会哈希后写入不同的mongos。

* **并行复制**

MongoShake提供了并行复制的能力，复制的粒度选项（shard\_key）可以为：id，collection或者auto，不同的文档或表可能进入不同的哈希队列并发执行。id表示按文档进行哈希；collection表示按表哈希；auto表示自动配置，如果有表存在唯一键，则退化为collection，否则则等价于id。

* **HA方案**

MongoShake定期将同步上下文进行存储，存储对象可以为第三方API（注册中心）或者源库。目前的上下文内容为“已经成功同步的oplog时间戳”。在这种情况下，当服务切换或者重启后，通过对接该API或者数据库，新服务能够继续提供服务。

此外，MongoShake还提供了Hypervisor机制用于在服务挂掉的时候，将服务重新拉起。

* **过滤**

提供黑名单和白名单机制选择性同步db和collection。

* **压缩**

支持oplog在发送前进行压缩，压缩格式可以为gzip, zlib, 或deflate。

* **Gid**

一个数据库的数据可能会包含不同来源：自己产生的和从别处复制的数据。如果没有相应的措施，可能会导致数据的环形复制，比如A的数据复制到B，又被从B复制到A，导致服务产生风暴被打挂了。或者从B回写入A时因为唯一键约束写入失败。从而导致服务的不稳定。

在阿里云上的MongoDB版本中，我们提供了防止环形复制的功能。其主要原理是，通过修改MongoDB内核，在oplog中打入gid标识当前数据库信息，并在复制过程中通过op\_command命令携带gid信息，那么每条数据都有来源信息。如果只需要当前数据库产生的数据，那么只抓取gid等于该数据库id的oplog即可。所以，在环形复制的场景下，MongoShake从A数据库抓取gid等于id\_A（A的gid）的数据，从B数据库抓取gid等于id\_B（B的gid）的数据即可解决这个问题。

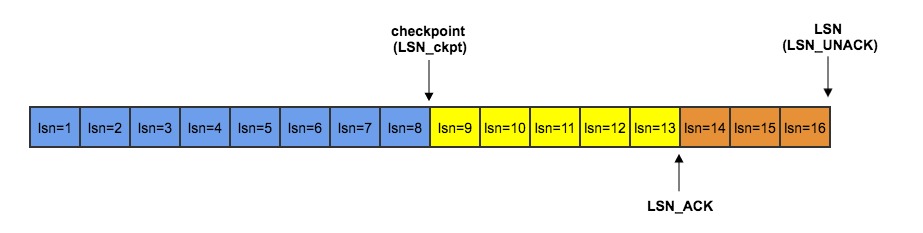
说明：由于MongoDB内核gid部分的修改尚未开源，所以**开源版本下此功能受限，**但在阿里云MongoDB版本已支持（<https://www.aliyun.com/product/mongodb?spm=5176.8142029.388261.351.a7236d3evDQVW1>）。这也是为什么我们前面提到的“MongoDB集群间数据的镜像备份”在目前开源版本下功能受限的原因。

* **Checkpoint**

MongShake采用了ACK机制确保oplog成功回放，如果失败将会引发重传。这主要是为了保证应用层可靠性而设计的，比如解压缩失败等等。为了更好的进行说明，我们先来定义几个名词：

* LSN（Log Sequence Number），表示已经传输的最新的oplog序号。
* LSN\_ACK（Acked Log Sequence Number），表示已经收到ack确认的最大LSN，即写入tunnel成功的LSN。
* LSN\_CKPT（Checkpoint Log Sequence Number），表示已经做了checkpoint的LSN，即已经持久化的LSN。

LSN、LSN\_ACK和LSN\_CKPT的值均来自于Oplog的时间戳ts字段，其中隐含约束是：LSN\_CKPT<=LSN\_ACK<=LSN。



如上图所示，LSN=16表示已经传输了16条oplog，如果没有重传的话，下次将传输LSN=17；LSN\_ACK=13表示前13条都已经收到确认，如果需要重传，最早将从LSN=14开始；LSN\_CKPT=8表示已经持久化checkpoint=8。持久化的意义在于，如果此时MongoShake挂掉重启后，源数据库的oplog将从LSN\_CKPT位置开始读取而不是从头LSN=1开始读。因为oplog DML的幂等性，同一数据多次传输不会产生问题。但对于DDL，重传可能会导致错误。

* **排障和限速**

MongoShake对外提供Restful API，提供实时查看进程内部各队列数据的同步情况，便于问题排查。另外，还提供限速功能，方便用户进行实时控制，减轻数据库压力。

* **冲突检测**

目前MongoShake支持表级别（collection）和文档级别（id）的并发，id级别的并发需要db没有唯一索引约束，而表级别并发在表数量小或者有些表分布非常不均匀的情况下性能不佳。所以在表级别并发情况下，需要既能均匀分布的并发，又能解决表内唯一键冲突的情况。为此，如果tunnel类型是direct时候，我们提供了写入前的冲突检测功能。

目前索引类型仅支持唯一索引，不支持前缀索引、稀疏索引、TTL索引等其他索引。

冲突检测功能的前提需要满足两个前提约束条件：

1. MongoShake认为同步的MongoDB Schema是一致的，也不会监听Oplog的System.indexes表的改动

2. 冲突索引以Oplog中记录的为准，不以当前MongoDB中索引作为参考。

另外，MongoShake在同步过程中对索引的操作可能会引发异常情况：

1. 正在创建索引。如果是后台建索引，这段时间的写请求是看不到该索引的，但内部对该索引可见，同事可能会导致内存使用率会过高。如果是前台建索引，所有用户请求是阻塞的，如果阻塞时间过久，将会引发重传。

2. 如果目的库存在的唯一索引在源库没有，造成数据不一致，不进行处理。

3. oplog产生后，源库才增加或删除了唯一索引，重传可能导致索引的增删存在问题，我们也不进行处理。

为了支持冲突检测功能，我们修改了MongoDB内核，使得oplog中带入uk字段，标识涉及到的唯一索引信息，如：

{

"ts" : Timestamp(1484805725, 2),

"t" : NumberLong(3),

"h" : NumberLong("-6270930433887838315"),

"v" : 2,

"op" : "u",

"ns" : "benchmark.sbtest10",

"o" : { "\_id" : 1, "uid" : 1111, "other.sid":"22222", "mid":8907298448, "bid":123 }

"o2" : {"\_id" : 1}

"uk" : {

"uid": "1110"

"mid^bid": [8907298448, 123]

"other.sid\_1": "22221"

}

}

uk下面的key表示唯一键的列名，key用“^”连接的表示联合索引，上面记录中存在3个唯一索引：uid、mid和bid的联合索引、other.sid\_1

。value在增删改下具有不同意义：如果是增加，则value为空；如果是删除或者修改，则记录删除或修改前的值。

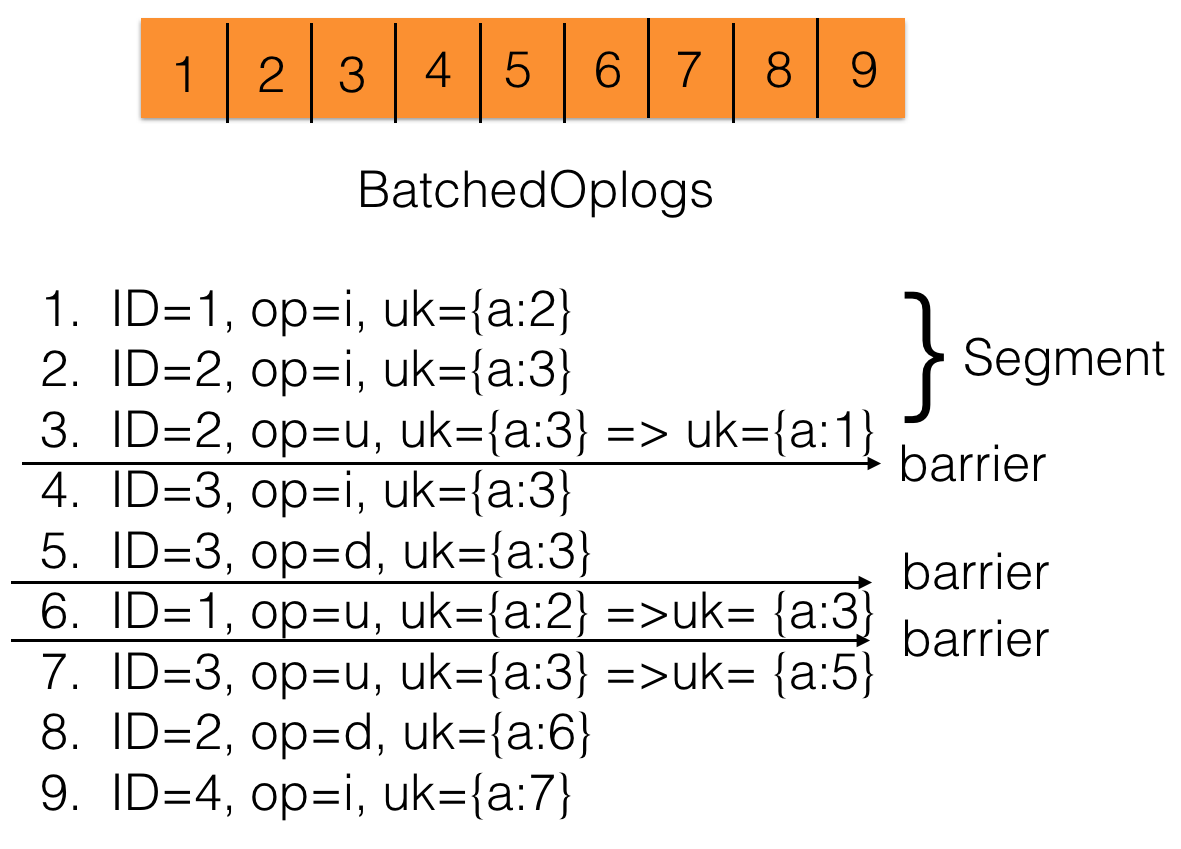
具体处理流程如下：将连续的k个oplog打包成一个batch，流水式分析每个batch之内的依赖，划分成段。如果存在冲突，则根据依赖和时序关系，将batch切分成多个段；如果不存在冲突，则划分成一个段。然后对段内进行并发写入，段与段之间顺序写入。段内并发的意思是多个并发线程同时对段内数据执行写操作，但同一个段内的同一个id必须保证有序；段之间保证顺序执行：只有前面一个段全部执行完毕，才会执行后续段的写入。

如果一个batch中，存在不同的id的oplog同时操作同一个唯一键，则认为这些oplog存在时序关系，也叫依赖关系。我们必须将存在依赖关系的oplog拆分到2个段中。

MongoShake中处理存在依赖关系的方式有2种：

**1. 插入barrier**

通过插入barrier将batch进行拆分，每个段内进行并发。举个例子，如下图所示：



ID表示文档id，op表示操作，i为插入，u为更新，d为删除，uk表示该文档下的所有唯一键, uk={a:3} => uk={a:1}表示将唯一键的值从a=3改为a=1，a为唯一键。

在开始的时候，batch中有9条oplog，通过分析uk关系对其进行拆分，比如第3条和第4条，在id不一致的情况下操作了同一个uk={a:3}，那么第3条和第4条之间需要插入barrier（修改前或者修改后无论哪个相同都算冲突），同理第5条和第6条，第6条和第7条。同一个id操作同一个uk是允许的在一个段内是允许的，所以第2条和第3条可以分到同一个段中。拆分后，段内根据id进行并发，同一个id仍然保持有序：比如第一个段中的第1条和第2，3条可以进行并发，但是第2条和第3条需要顺序执行。

**2. 根据关系依赖图进行拆分**

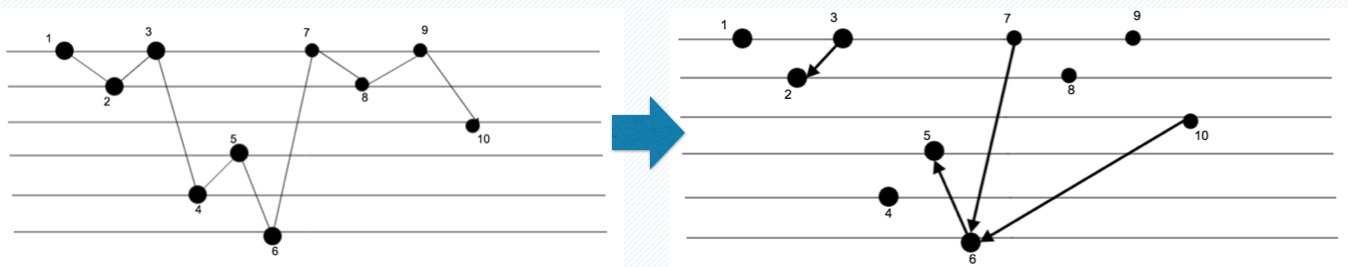
每条oplog对应一个时间序号N，那么每个序号N都可能存在一个M使得：

* 如果M和N操作了同一个唯一索引的相同值，且M序号小于N，则构建M到N的一条有向边。
* 如果M和N的文档ID相同且M序号小于N，则同样构建M到N的一条有向边。
* 由于依赖按时间有序，所以一定不存在环。

所以这个图就变成了一个有向无环图，每次根据拓扑排序算法并发写入入度为0的点即可:

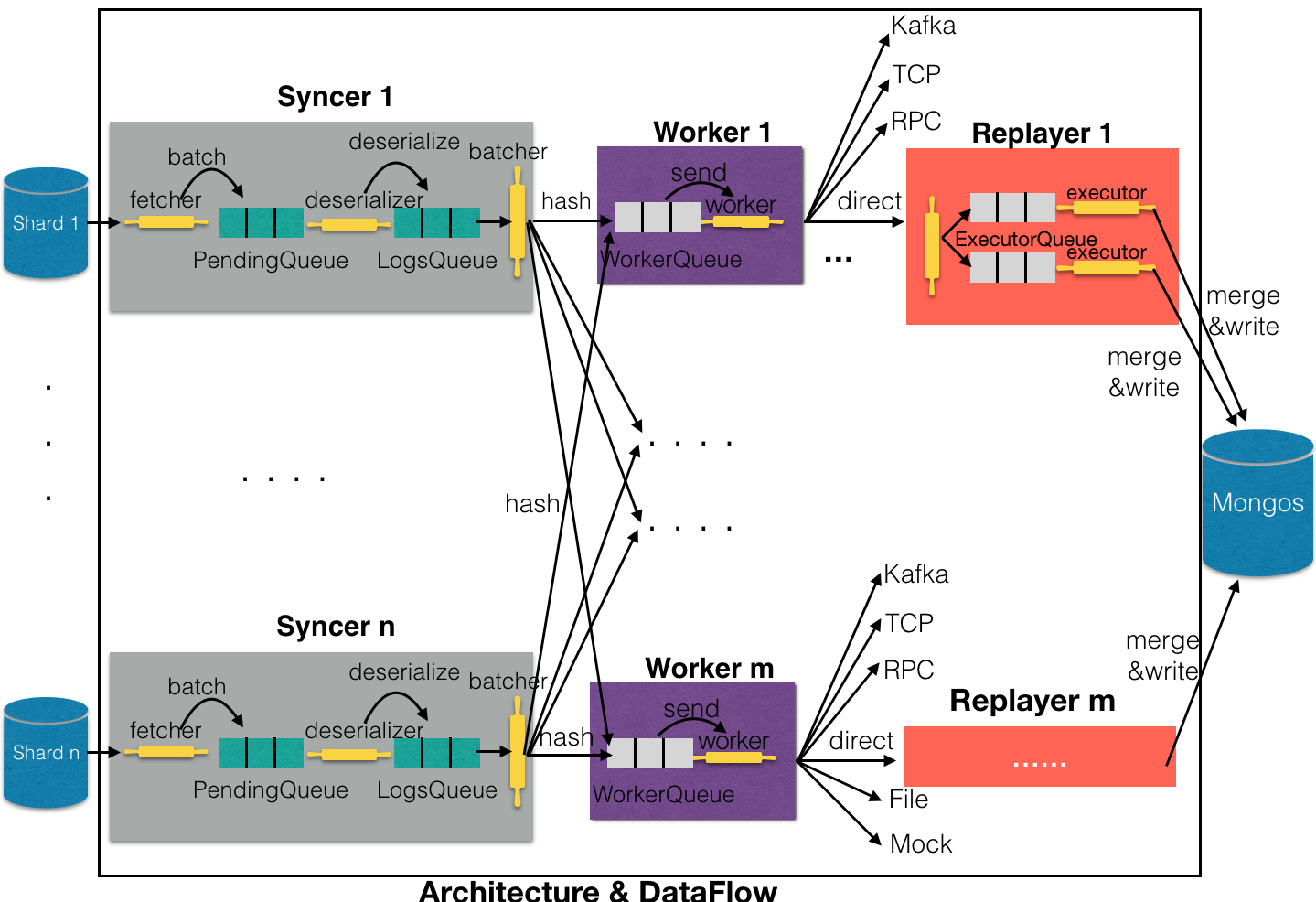
|  |
| --- |
| 1. 建图 2. 找到所有入度为0的结点，加入集合 3. 集合内进行并发写入 4. 将所有起始在集合的边进行删除，同时对后继儿子节点入度减一。 5. 清空集合 6. 跳转到2，直到集合为空则退出。 |

下图给出了一个例子：一共有10个oplog结点，一个横线表示文档ID相同，右图箭头方向表示存在唯一键冲突的依赖关系。那么，该图一共分为4次执行：并发处理写入1，2，4，5，然后是3，6，8，其次是7，10，最后是9。



说明：由于MongoDB中冲突检测uk部分的修改尚未开源，所以**开源版本下此功能受限，**但在阿里云MongoDB版本已支持（<https://www.aliyun.com/product/mongodb?spm=5176.8142029.388261.351.a7236d3evDQVW1>）。

# 架构和数据流



上图展示了MongoShake内部架构和数据流细节。总体来说，整个MongoShake可以大体分为3大部分：Syncer、Worker和Replayer，其中Replayer只用于tunnel类型为direct的情况。

Syncer负责从源数据库拉取数据，如果源是Mongod或者ReplicaSet，那么Syncer只有1个，如果是Sharding模式，那么需要有多个Syncer与Shard一一对应。在Syncer内部，首先fetcher用mgo.v2库从源库中抓取数据然后batch打包后放入PendingQueue队列，deserializer线程从PendingQueue中抓取数据进行解序列化处理。Batcher将从LogsQueue中抓取的数据进行重新组织，将前往同一个Worker的数据聚集在一起，然后hash发送到对应Worker队列。

Worker主要功能就是从WorkerQueue中抓取数据，然后进行发送，由于采用ack机制，所以会内部维持几个队列，分别为未发送队列和已发送队列，前者存储未发送的数据，后者存储发送但是没有收到ack确认的数据。发送后，未发送队列的数据会转移到已发送队列；收到了对端的ack回复，已发送队列中seq小于ack的数据将会被删除，从而保证了可靠性。

Worker可以对接不同的Tunnel通道，满足用户不同的需求。如果通道类型是direct，那么将会对接Replayer进行直接写入目的MongoDB操作，Worker与Replayer一一对应。首先，Replayer将收到的数据根据冲突检测规则分发到不同的ExecutorQueue，然后executor从队列中抓取进行并发写入。为了保证写入的高效性，MongoShake在写入前还会对相邻的相同Operation和相同Namespace的Oplog进行合并。

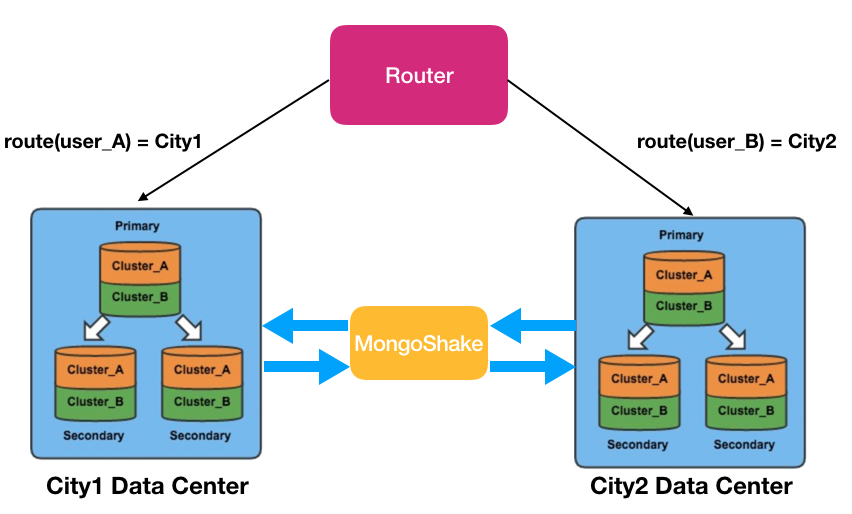
# 用户使用案例

高德地图 App是国内首屈一指的地图及导航应用，阿里云MongoDB数据库服务为该应用提供了部分功能的存储支撑，存储亿级别数据。现在高德地图使用国内双中心的策略，通过地理位置等信息路由最近中心提升服务质量，业务方(高德地图)通过用户路由到三个城市数据中心，如下图所示，机房数据之间无依赖计算。

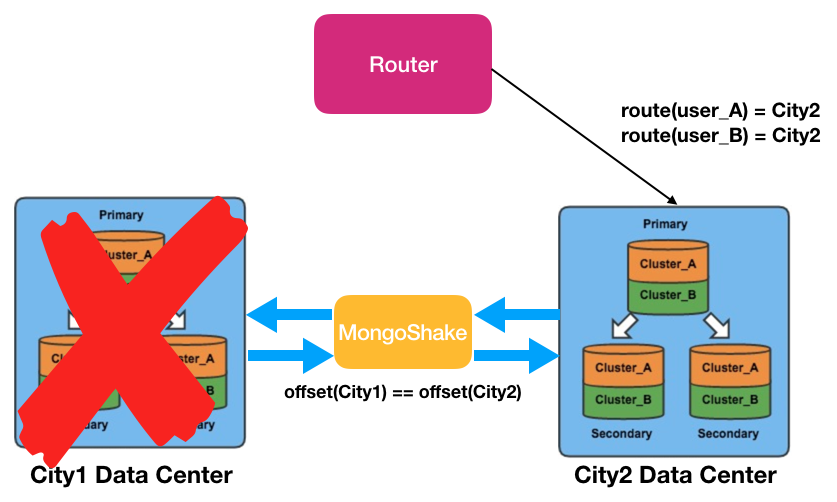


这三个城市地理上从北到南横跨了整个中国 ，这对多数据中心如何做好复制、容灾提出了挑战，如果某个地域的机房、网络出现问题，可以平滑的将流量切换到另一个地方，做到用户几乎无感知？

目前我们的策略是，拓扑采用机房两两互联方式，每个机房的数据都将同步到另外两个机房。然后通过高德的路由层，将用户请求路由到不同的数据中心，读写均发送在同一个数据中心，保证一定的事务性。然后再通过MongoShake，双向异步复制两个数据中心的数据，这样保证每个数据中心都有全量的数据(保证最终一致性) 。任意机房出现问题，另两个机房中的一个可以通过切换后提供读写服务。下图展示了城市1和城市2机房的同步情况。



遇到某个单元不能访问的问题，通过MongoShake对外开放的Restful管理接口，可以获得各个机房的同步偏移量和时间戳，通过判断采集和写入值即可判断异步复制是否在某个时间点已经完成。再配合业务方的DNS切流，切走单元流量并保证原有单元的请求在新单元是可以读写的，如下图所示。



# 测试数据

具体测试数据请参考测试文档

# 后续

MongoShake将会长期维护，大版本和小版本将会进行持续迭代。欢迎提问留言以及加入一起进行开源开发。

另外，阿里云MongoDB团队招人，欢迎投递简历以及来信咨询：[zhuzhao.cx@alibaba-inc.com](mailto:zhuzhao.cx@alibaba-inc.com)