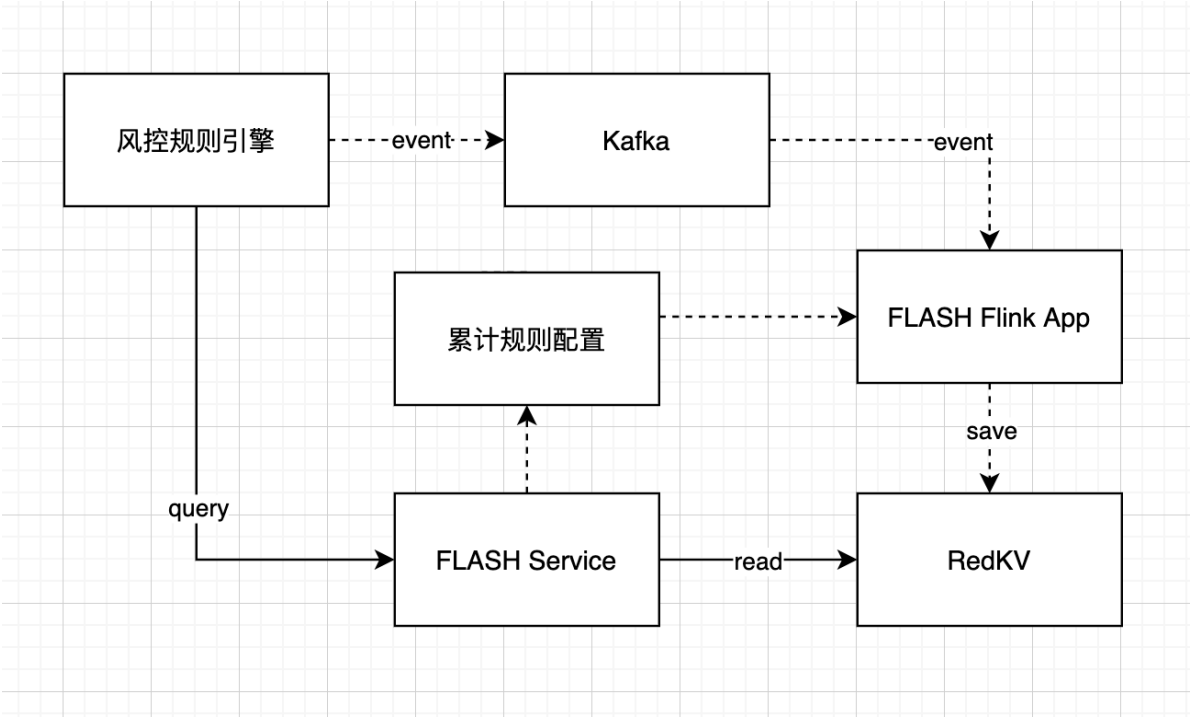
FLASH重构设计方案

背景

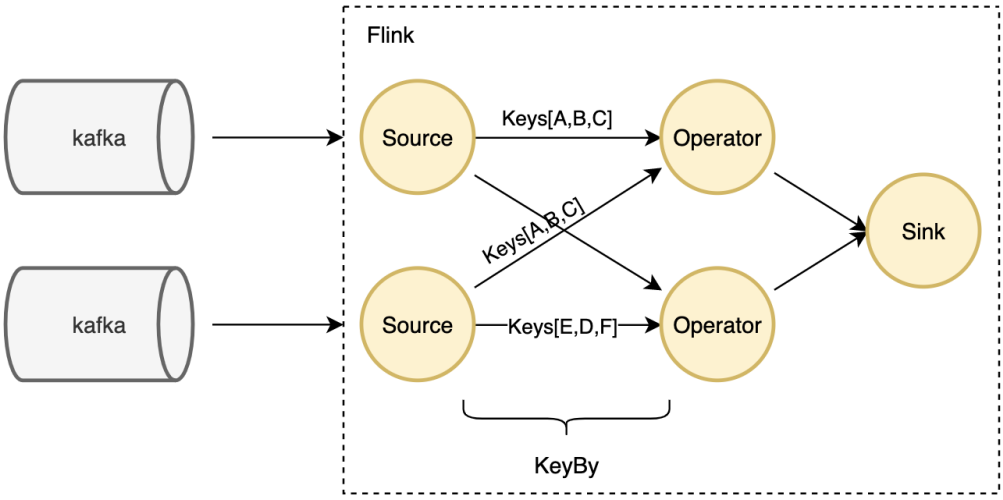
FLASH是一个基于事件驱动的实时规则聚合计算平台，目前主要用于风控规则引擎累计因子的计算和查询服务。



目前，FLASH的整体处理流程如下：

1. 风控规则引擎将事件数据发送到Kafka；
2. FLASH Flink App根据累计规则配置，消费Kafka的事件数据；
3. FLASH Flink App对事件进行累计后，将结果写入RedKV；
4. 风控规则引擎对FLASH Service发起累计因子查询；
5. FLASH Service提供RPC接口，查询RedKV中存储的累计数据，并将结果返回给规则引擎；

Flink App使用KeyBy将同一个事件主体的数据分配到同一个的算子任务，保证一个主体的事件只有一个线程在计算，避免多线程更新问题：



FLASH目前支持的计算类型如下：

计算类型	计算方法
SUM	对窗口期内的数据进行求和
COUNT	对窗口期内的数据进行计数统计
MAX	计算出窗口期内的最大值
MIN	计算出窗口期内的最小值
AVG	对窗口期内的数据进行均值计算
COUNT_DISTINCT	对窗口期内的指定字段去除重复量后计数统计

现状

当前FLASH是基于滑动时间窗口算法实现聚合计算。首先，每个累计因子会设定一个查询窗口，一个查询窗口会分为N个分片窗口，每个分片窗口包含查询窗口其中一段时间范围。计算时，根据事件的发生时间得出它属于哪个分片，并将结果计入到这个窗口中。

举个例子：

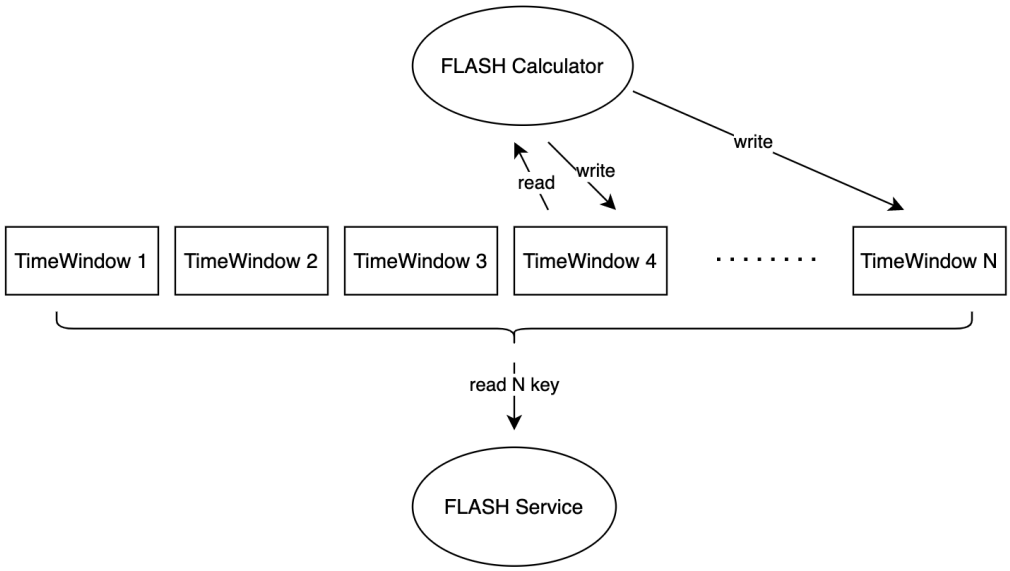
累计因子：用户在过去5分钟的点赞次数

查询窗口：5分钟

分片窗口：假设以1分钟一个窗口，共5个窗口

写入：假设用户的一次点赞时间发生在09:31:08，根据分窗规则，这个事件的累计结果将被计入(09:31:00-09:32:00]窗口，窗口对应的key为{累计因子id}:{累计维度}:{维度值}:{分片窗口的结束时间戳}，value为{累计结果}。

查询：如果风控规则引擎在09:33:20发起这个累计因子的查询，根据分窗规则，33:20对应(09:33:00-09:34:00]窗口，需要过去5分钟的数据，则此次查询对应的5个分片窗口为：(09:30] (09:31] (09:32] (09:33] (09:34]。将所有分片窗口全部查出并汇总累计，得出最后结果。



痛点

1. 当前每一个分片窗口对应1个key，每次查询时需要查询N个窗口，相当于1个RPC请求底层对应N个Key请求（从目前线上的统计来看平均10-14个），对数据库存在明显的读放大问题，数据库的压力随着请求量的上升而成倍的增长，消耗大量的服务器和网络资源。

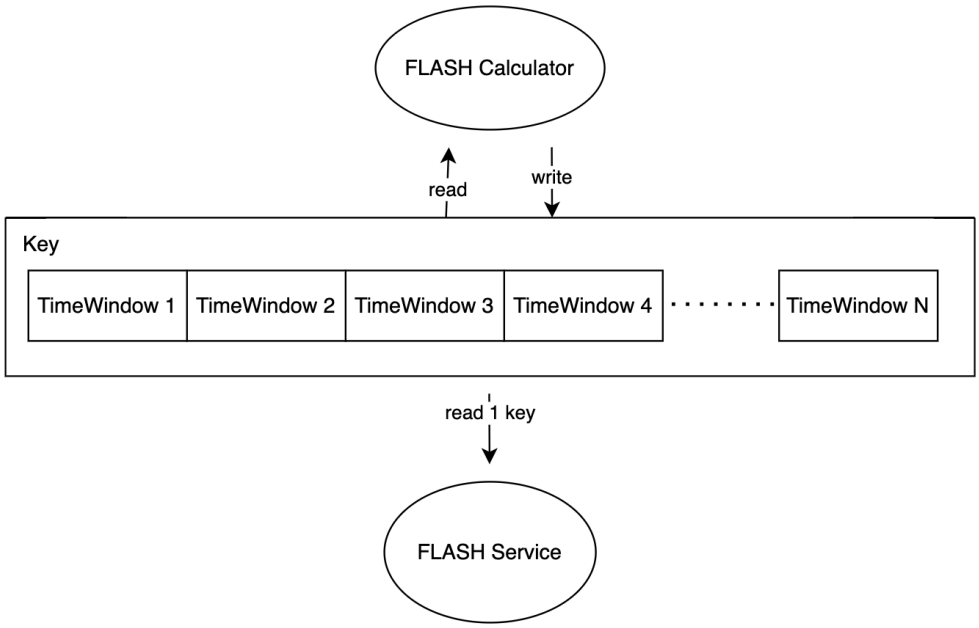
2. 受限于读放大的问题，在设置累计因子的分片规则时会尽量的减少窗口数。对于一个时间段，划分的窗口数就越小，每个窗口的时间范围就越大，导致计算误差就会越大。

3. 当前Distinct算法实现基于Hyperloglog，只能计算出一个估算值，样本量越大误差也会越大，且计算后存储的数据体积较大。

解决方案

概要设计

要解决读放大的问题，同时实现滑动时间窗口的累计效果，一个方法是将所有的分片窗口合并存储在一个key当中：



合并之后带来的好处：

1. 对于每次查询，只需要读1个key，解决了读放大的问题；
2. 解决了读放大的问题，分窗规则可以更加灵活，减小因为分窗过大导致的计算误差；

合并之后带来的问题：

1. value变大，会给数据库的IO带来压力，latency毛刺请求可能会增多，数据库方面建议value不超过32KB；
2. 计算复杂度变大，原来只需要计算对应的窗口然后查询出来聚合即可，合并之后查询之后还需要做额外的解析和计算；
3. 数据清理难度变大，原来只需要对每个窗口key设置过期时间，合并之后不能对整个key设置过期时间；
4. count distinct使用Hyperloglog计算后得到的数据较大，合并到一个key后会更容易出现大value；

详细设计

数据结构设计

累计主体的key是唯一的，且对应到一个累计主体。value存储所有的分片窗口，对象中包括窗口的时间及其累计结果。

```
key
{累计因子id}:{累计维度}:{维度值}
value
[{"ts": 1679275800, "result": 21},{"ts": 1679275860, "result": 31},{"ts":
1679275920, "result": 13},
{"ts": 1679276040, "result": 68},{"ts": 1679276100, "result": 75},{"ts":
1679276160, "result": 94}, ... ]
```

value是一个列表，列表包含多个窗口对象，每个窗口对象包含其对应的分片窗口结束时间戳和累计值，列表内的窗口对象根据窗口时间进行排序。

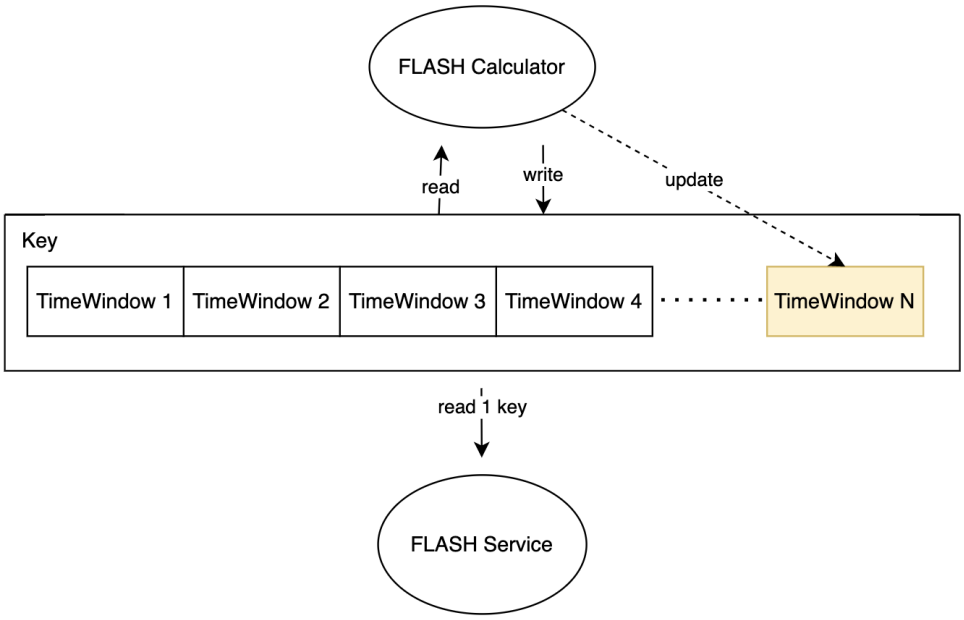
列表内的窗口并不一定是连续的，因为事件不一定是连续发生，我们只需要存储有事件发生的时间窗口。

以上数据结构使用JSON格式表示，实际实现时，可以采用Protobuf+Snappy对数据进行整理压缩，减少数据体积大小，提高效率。

逻辑设计

计算阶段，对于每一个累计事件：

1. 构建累计主体key，查询出对应的value，并反序列化为结构数据；
2. 根据事件的时间戳，从value中取出目标窗口对象；
3. 在目标窗口对象上进行累计计算；
4. 将累计结果更新到value对应的窗口对象中；
5. 将value序列化并写入数据库中；



读取阶段，对于每一个累计因子查询：

1. 构建累计主体key，查询出对应的value，并反序列化为结构数据；
2. 根据查询窗口条件，从value中选出符合条件的分片窗口；
3. 对选出的分片窗口做聚合计算，得出结果；

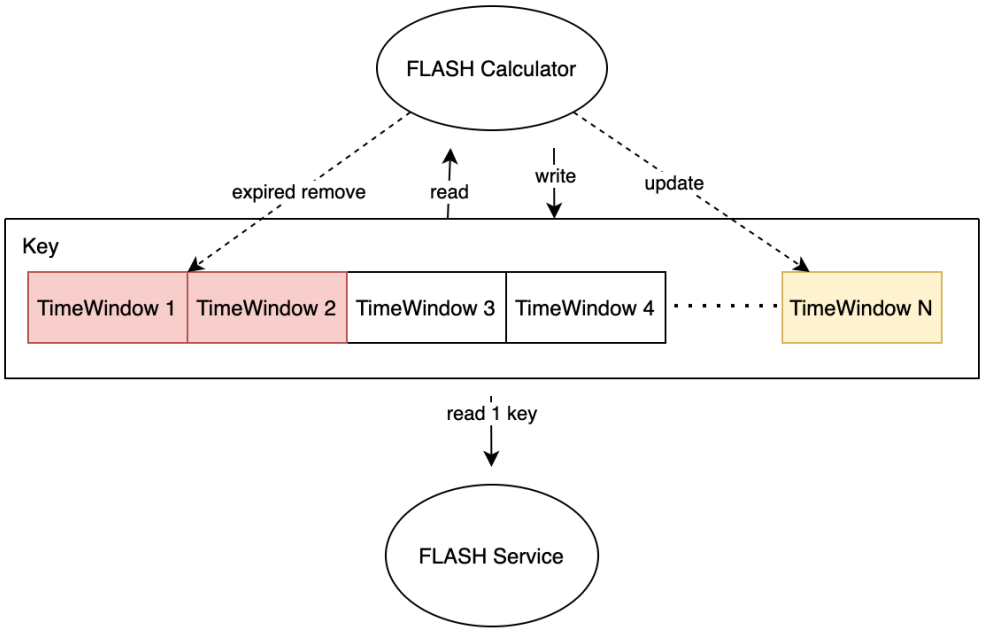
数据清理

由于合并之后不能对整个key设置TTL，随着时间流逝，value会不断增大。因此需要在计算阶段做一些额外的操作：

1. 根据提前设定好的过期时间，筛选并删除掉value已过期的分片窗口，再将value写入数据库。

2. 每次写入的同时给key也设置TTL，使key能自动失效，防止一个key在某一次写入后不再被更新。

key的TTL设置必须大于其对应的累计因子的查询窗口时间，如一个累计因子的查询窗口是1小时，则key的TTL必须要大于1小时。为了便于排查问题和满足回溯需求，可以适当加大，例如为查询窗口1小时的key设定TTL为1天。具体可以参考目前FLASH的TTL设置。



COUNT_DISTINCT的实现

方案一：

FLASH现有的方案是使用Hyperloglog计算后将结果序列化为一个字节数组后写入数据库。这个字节数组会随着样本量的增多越来越大，直到固定大小。

和上文的设计类似，将所有分片窗口的累计值都合并到一个value，累计值是Hyperloglog的计算结果。

计算阶段，只对目标分片窗口的累计值更新，同时淘汰过期窗口。

读取阶段，根据key查询value后，筛选出符合条件的窗口，做一次聚合计算。

优点	缺点
● 简单 ● value体积可控	● value体积较大 ● 计算不精确

根据目前FLASH使用的HLL配置，以ObjectId作为样本来测试，超过300个样本后序列化结果大小固定在515bytes。

对HLL要求的精度越高，占用体积就越大。

Guide to the HyperLogLog Algorithm in Java | Baeldung

方案二：

将样本值存储到另一个客体key中，每个客体key对应一个时间窗口，value存储窗口中所有的累计样本，累计时将其查出用于计算。

```
key
累计客体key标识:{累计因子id}:{累计维度}:{维度值}:{分片窗口时间戳}
value
["sample1", "sample2", "sample3", ..., "sampleN"]
```

计算阶段，将主体key和查询窗口内所有客体key一并查出，将当前事件与客体key中的现存样本做对比：

1. 如果当前事件样本不存在，则将主体key中的结果count+1，并将事件计入对应的客体key中

2. 如果当前事件样本已存在，则不增加count计数，并将样本从原客体key中删除，更新至当前事件对应的客体key中

读取阶段，只需要查询主体key就可以得出COUNT_DISTINCT结果。

优点	**缺点**
● 存储了所有样本，可以检索 ● 计算精确	● value体积受样本量的影响（已知在反爬场景有十万/百万数量级） ● 没有回溯的能力

方案三：

每一个样本都存一个客体key，value中存储样本最晚的事件时间戳：

```
key
{累计因子id}:{累计维度}:{维度值}:{样本值}
value
样本最晚的事件时间戳
```

计算阶段，将主体key和客体key一并查出：

1. 当客体key不存在时，则将主体key中对应分片的count+1，并新增客体key，value为当前事件时间

2. 客体key已存在，对比当前事件时间和客体key的value中的原事件时间：

a. 当前事件时间<原事件时间，不处理

b. 在同一个查询窗口内：

i. 在同一个分片窗口内，不修改分片值

ii. 不在同一个分片窗口内，将原事件对应的分片count-1，将当前事件对应的分片count+1

c. 不在同一个查询窗口内：

i. 不更新原事件对应分片，将当前事件对应的分片count+1

3. 若当前事件时间>原事件时间，更新客体key的value为当前事件的时间

4. 客体key TTL设置必须大于查询窗口时间

读取阶段，只需要查询主体key，选出对应的分片做一次聚合计算就可以得出COUNT_DISTINCT结果。

优点	缺点
● value体积小 ● 计算精确	● key数量受样本量的影响 ● 离散的数据，无法关联检索 ● 没有回溯的能力

【结论】：使用方案三

验证方案

1. 对于累计服务，新建一个flink任务消费相同的kafka topic，采用新的设计方案，同时保证新的数据和现有的数据隔离。

2. 对于查询服务，复制部分线上查询请求，由异步线程去执行新的计算逻辑，然后对比新老逻辑的结果。

3. 针对新老逻辑的对比结果，做打点监控和日志。

任务拆分&排期

编号	任务项	负责人	预计完成时间	状态
1	资源预估和申请	@流河(廖立标)	2023-03-24	DONE
2	schema设计开发	@流河(廖立标)	2023-03-24	DONE
3	查询逻辑开发	@流河(廖立标)		
4	计算逻辑开发			
5	数据验证			