# 第三届中间件性能挑战赛比赛攻略

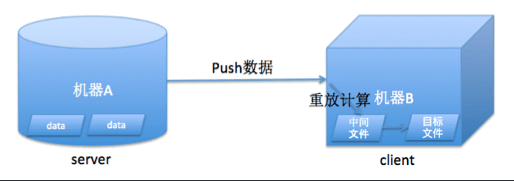
### 团队：一颗赛艇 队员：谭钧升 张明 俞飞樾

#### 第二赛季：《模拟阿里双十一分布式数据同步》

##### 赛题背景分析及理解

题目内容是：题目主要解决的是数据同步领域范畴：实时增量同步，主要的技术挑战为模拟数据库的主备复制，提供"高效"的实时同步能力。即给定一批固定的增量数据变更信息，程序需要收集增量变更信息，并进行一定的数据重放计算，然后将最终结果输出到给定的目标文件中。增量数据的变更信息为了简化处理，会给出明文的数据，主要包含数据库的insert/update/delete三种类型的数据。具体的增量数据变更信息的数据格式见环境描述部分。数据重放主要是指模拟数据库的insert/update/delete语义，允许使用一些中间过程的存储。

其中，程序会使用两台机器，一台作为Server机器，存放着需要重放的数据文件，一台作为Client机器，需要保存重放之后的结果文件。官方给出的例图如下：



可以看到，主办方的出题背景是基于阿里在实际业务中遇到的跨机房数据库同步问题，即将在机器A上的数据库数据同步到机器B上。

而此赛题一些比较重要的点是：

1. 因为要模拟数据库的实时增量同步，所以对于给定的多个数据文件，只允许单线程顺序读每个文件；
2. 在测评程序进行查询的时候，查询的只是部分区间的数据记录，而且更重要的是，这个查询区间在程序一开始就给出！
3. 可能会存在主键变更。

针对这些特点，特别是特点(2)，我们可以很自然的想到，程序可以只重放查询区间内的数据即可！而这确实也是整个第二赛季所有选手的基本思路。

程序的运行环境是：两台配置一样的机器，每台16核CPU，32GB内存，JVM限制为3GB内存，磁盘为SSD，读写性能为110MB/S。数据文件有10个，总共有10GB左右，存放在内存文件系统，读性能是4GB/S。。

##### 核心思路

从预热赛开始算起，第二赛季持续了近二十天，而我们的程序也由于各种原因经历了多次的推倒重来。接下来，我们将按照时间顺序概要介绍一下所编写过的三个不同版本的程序，其中，成绩最优的是版本3的代码。

##### 版本1

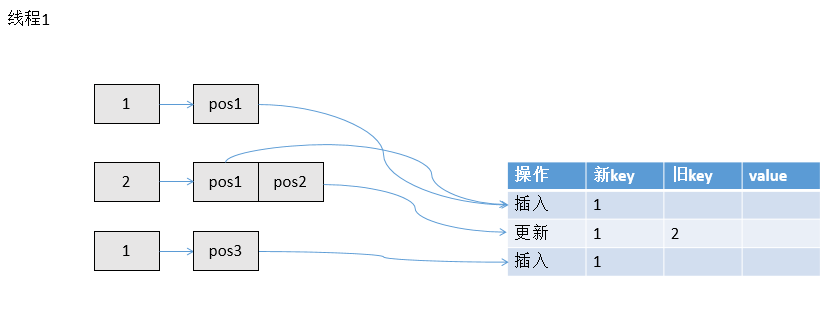
**时间**：热身赛

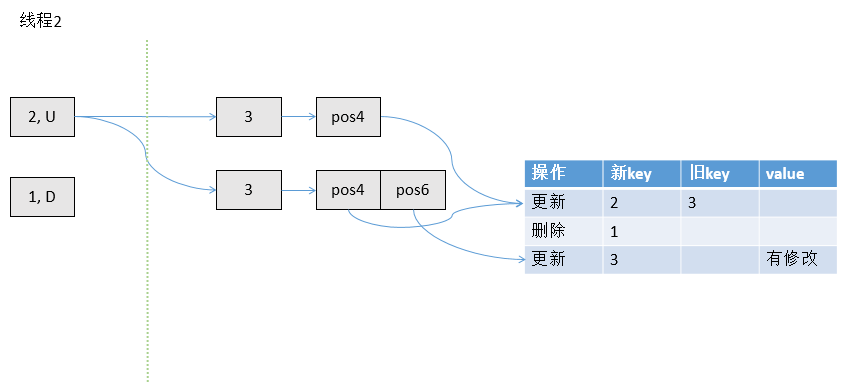
**评测环境**：24核CPU，96GB内存，读写性能较好（400MB）SSD。

**备注：**彼时尚未明确对于10GB原数据文件，只允许单线程顺序流式读一遍。

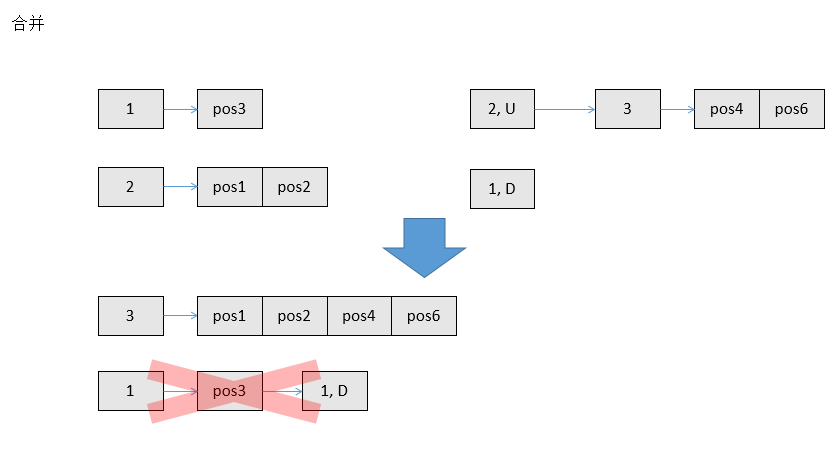
**算法设计思路**：当时，官方尚未明确提出流式读数据文件的要求，因此，当时的设计重点关注的是：如何充分利用多核性能。因此，版本1的设计如下：

1. 大文件分片处理。由于有24核，10个文件，因此每个文件平均分成两部分，每部分交给一条线程处理，总共有20条线程同时处理文件。
2. 每条线程在解析数据时，保存某个主键某条变更记录在哪个文件和在文件中的位置(pos)，也就是构建索引。其中，为了减少内存消耗，这些信息用4byte的int来编码。如下图所示：





1. 把20个线程的key变更记录合并，保证同一条数据的key变更记录在一起。如下图所示：



1. 使用24条线程根据前面构造的变更记录的索引来进行数据重放。在重放时跳过不在查询范围内的主键。
2. 对结果根据主键排序，输出。

**创新点：**将文件标识和文件偏移压缩编码到一个4byte的int里，最大限度减少内存消耗； 设计可扩展性好，能充分利用多核。

**通用性**：好，没有针对数据集特征进行任何过滤，可适用于任何表结构任何数据集。

##### 版本2

**时间**：热身赛末期~正式赛初期

**评测环境**：16核CPU，32GB内存，JVM内存3GB，数据文件保存在RAMFS，读性能超好；中间文件需要写到SSD，写性能只有100MB/S左右。

**备注：**已经明确对于10GB原数据文件，只允许单线程顺序流式读一遍。

**算法设计思路**：由于数据文件只能单线程顺序读一遍，而能用的JVM内存只有3GB，所以只能写相关数据到SSD暂存了：

1. 单线程顺序读源文件，并将某主键的变更记录的有效内容哈希到某个文件。例如，假设有13个中间文件，pk=14，那么保存到的文件是：fileNo = pk%13 = 1， 也就是第一个文件。这样hash的目的是保证所有相同pk的变更记录都保存到同一个中间文件，这样在后面进行重放的时候，每条线程只需读一个文件即可，文件之间无依赖性。然而，考虑到存在主键变更，所以需要保存主键变更的映射，以保证所有有依赖（产生过主键变更）的主键的变更记录都写到同一个文件。例如，pk原本是14，那么该pk的所有有效变更内容全部保存在中间文件1。而当发生了主键变更14 -> 16时，记录此次主键变更，然后对于从此往后遇到的pk=16的变更记录，都认为是pk=14，都写到中间文件1里面去。特别的，不管主键发生多少次变更，都以第一次遇到的pk为准，如pk=14->16->17，则在计算应该hash到哪个中间文件时，认为16和17等价，都是14。
2. 在第一步读文件写中间文件的同时，构建映射：pk -> pk最终的值。例如，如果某个pk发生了如下变更：16 -> 66 -> 5 -> 10， 则保存的映射为：16->10, 66->10, 5->10。这样的目的是，在后面重放数据时，对于遇到的pk，先查表找到其对应的最终值，如果最终值不在查询范围，则不进行重放，忽略该pk的变更记录。
3. 经过步骤1和2后，产生了13个中间文件，因此使用13条线程并行重放构造结果。

**创新点：**通过hash的方法消除原10个文件里面变更记录的依赖性，在最后并行读中间文件重放结果时完全无锁，并行性很好。

**通用性**：很好，没有针对数据集特征进行任何过滤，可适用于任何表结构任何数据集。

##### 版本3 （成绩最优版本）

（在编写优化版本2的过程中，我偶然发现，线上测评的数据里面，居然没有“查询范围外的主键变更到查询范围内”的情况！而且，最后结果集里面的主键，也没有发生过一次主键变更！查询范围内唯一的主键变更情况只有：范围内主键变更到范围外！例如，如果查询范围是1~20，不存在pk变更：2->23->4，也不存在25->3，甚至连:5->7都不存在！只存在这样一种很弱的情况：8->28。在发现数据集的这些略bug的特征以后，我联系了比赛官方人员，确认是否能利用这些特征来进行针对性优化，在得到肯定的答复后，我便在寻找数据集特征的路上越跑越远。。。。）

**时间**：正式赛中后期

**评测环境**：16核CPU，32GB内存，JVM内存3GB，数据文件保存在RAMFS，读性能超好；中间文件需要写到SSD，写性能只有100MB/S左右。

**备注：**已经明确对于10GB原数据文件，只允许单线程顺序流式读一遍。

**算法设计思路**：版本3可以用“探索发现之旅”来概括。主要探索并利用到的数据集特征有：

1. 数据集里，表的各列值的取值范围是有限的，因此可以用64bit的原始类型long来编码表示表的一行记录。具体来说，数据集里的一行记录的5个属性以及其取值范围如下：

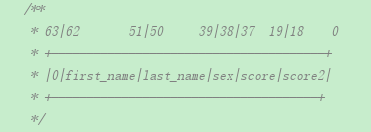
（1），first\_name，字符串类型；

（2），last\_name，字符串类型。其中first\_name和last\_name都是中文，而first\_name和last\_name的不同的取值加起来只有1921个！

(3),sex，性别，字符串类型，只有’男’,’女’两种取值;

(4), score, score2，数字类型，大小范围是0~30多万。

因此，对于first\_name和last\_name，我们可以在第一次碰到某个字符串时保存其到一个ArrayList里面，然后用该字符串在ArrayList的下标来表示该字符串。因为总共只有1900多个不同的取值，所以用12bit（2^12=4096）来表示first\_name和last\_name在ArrayList的下标；而对于score和score2，由于范围是30多万，因此可以用19bit （能表示50多万的范围）来表示其值。而对于性别，只用1bit即可编码表示。因此，一行记录可以如下编码：



这样，所有结果便可用long数组保存下，而且非常节省内存，即使最终结果有1千万个，理论上也只需76MB左右的内存。更重要的是，所有的变更操作都可以用位操作来进行，速度也较快。

1. 数据集里，不存在“范围外主键变更到范围内”的情况！也不存在主键多次变更的情况！最后结果集里面的主键一次主键变更都没有发生过！这几点让比赛的难度大大降低了，版本1，版本2里面考虑到的很多地方都没必要再做。因此，对于原数据文件，只需要读一遍即可构造出最终结果。而且，在读的时候可以直接过滤掉不在查询范围内的主键的变更记录。
2. 最终结果集的主键都是奇数！在发现这一点之后，在解析文件的时候，便能过滤更多数据了：如果pk为偶数，忽略跳过该变更记录。
3. 最终结果集的主键最大值只有5240295。虽然查询范围的主键大小是一百万到八百万，但发现这个特征后，查询范围可以缩小三百万。
4. 对于变更，只需要应用最后一次变更即可！例如，如果score字段发生如下UPDATE：66->77->100->99，那么66->77，77->100这两次变更记录都可忽略，直接应用77->100这次变更即可。因此，我们只需找到结果集所有的主键里面，每个字段的最后一次变更，发生在哪个文件即可。例如，经过统计，我发现结果集里的所有主键，first\_name字段的最后一次更新都发生在文件10.txt，而last\_name的字段的最后一次更新，有的主键发生在文件7.txt，有的发生在10.txt，而Score字段的最后一次更新也都是发生在文件10.txt，性别和score2则根本没有发生过变更。因此，由于字段最后一次发生变更的最早文件是7.txt，那么我们可以完全忽略掉文件1.txt到文件6.txt的UPDATE变更记录！
5. 接着5的思路，我们探索出了INSERT变更的分布特征：虽然结果集的主键的INSERT操作分布到了所有的10个文件，但是不同的文件里面的具体位置分布也是不一样的！而且范围较小！例如，在文件2.txt，INSERT变更的位置的范围是： [871613619，871763998]，整个INSERT长度只有1MB左右！那么对于文件2.txt，我们就可以忽略掉不在范围内的所有变更记录了！也就是虽然文件2.txt大小有979MB，但是需要我们解析的其实只有1MB多的数据，其它近978MB数据都被我们过滤掉了！因此，应用了这个特征后，程序运行时间有了大幅度的减少。

**创新点：**用long数组来表示记录，最大限度的减少了内存的使用；而对于first\_name，last\_name取值的字符串，也只在第一次遇到时需要new String，后面都可以直接用下标来表示，避免了在解析数据过程中不断新建对象。通过这两点优化，整个程序运行过程的GC次数减少到了0次。

**通用性**：极差，属于为了追求极致的性能而针对数据集针对表结果做的极端优化，换一套数据集换一套表结构而不修改程序的话，程序不可能运行正确。

##### 赛后感想

官方出题组基于实际业务需求出题，是希望能够看到选手的设计有创新性，能够借鉴到实际业务中。然而，在发现输入数据的一些漏洞之后，我们开始走向了利用数据特征之路，虽然性能越来越好，但健壮性通用性也越来越差，最后产出了工程价值比较低的代码，可谓此次第二赛季比赛的一个遗憾。