

# 第三届阿里中间件性能挑战赛

挑战双十一万亿级消息引擎



# 团队介绍 -- 也许放弃才能靠近你

2/27



## ➤ 胡建洪（队长）：负责维护版本1

南京理工大学硕士（研二），获得过国家奖学金，热爱跑步，正在努力进步的菜鸟程序员。



## ➤ 闵大为：负责维护版本2

浙江大学硕士（研二），获得过国家奖学金，喜欢打球看书，正在努力进步的菜鸟程序员。



## ➤ 胡传铃：负责 review 代码及部分模块

浙江大学硕士（研一），喜欢看书和关注时事政治，正在努力进步的菜鸟程序员。

# 目录

1

问题回顾

2

流水化处理

3

数据结构

4

解析处理

5

其它优化

6

总结

1

## 问题回顾

# 问题回顾

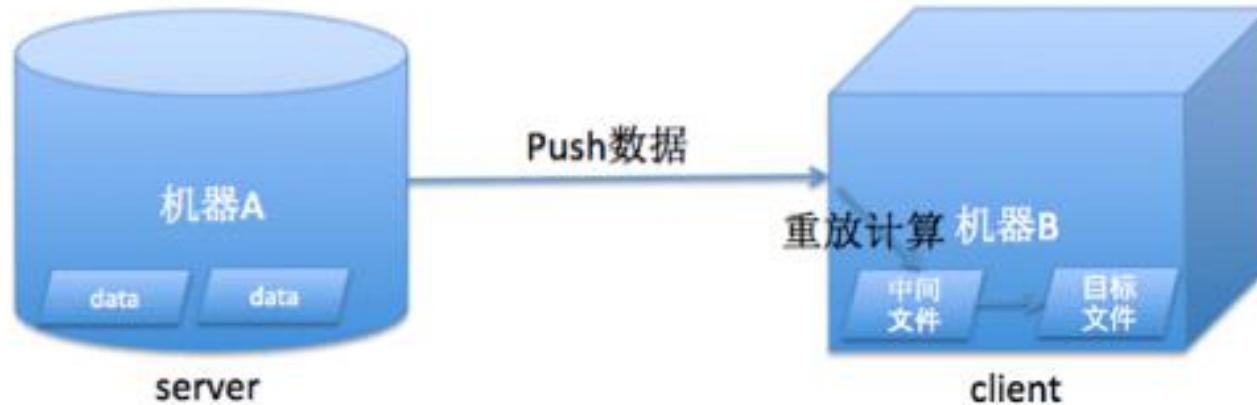
对10G的增量数据变更信息进行重放计算并储存

服务端：

- 10G增量数据的读取。
- 10G增量数据的解析。
- 10G增量数据的计算。

客户端：

- 从Server端获取中间结果。
- 解析为目标结果。
- 将目标结果写入目标文件。



2

## 流水化处理

# 流水化处理 -- 细分为3个部分

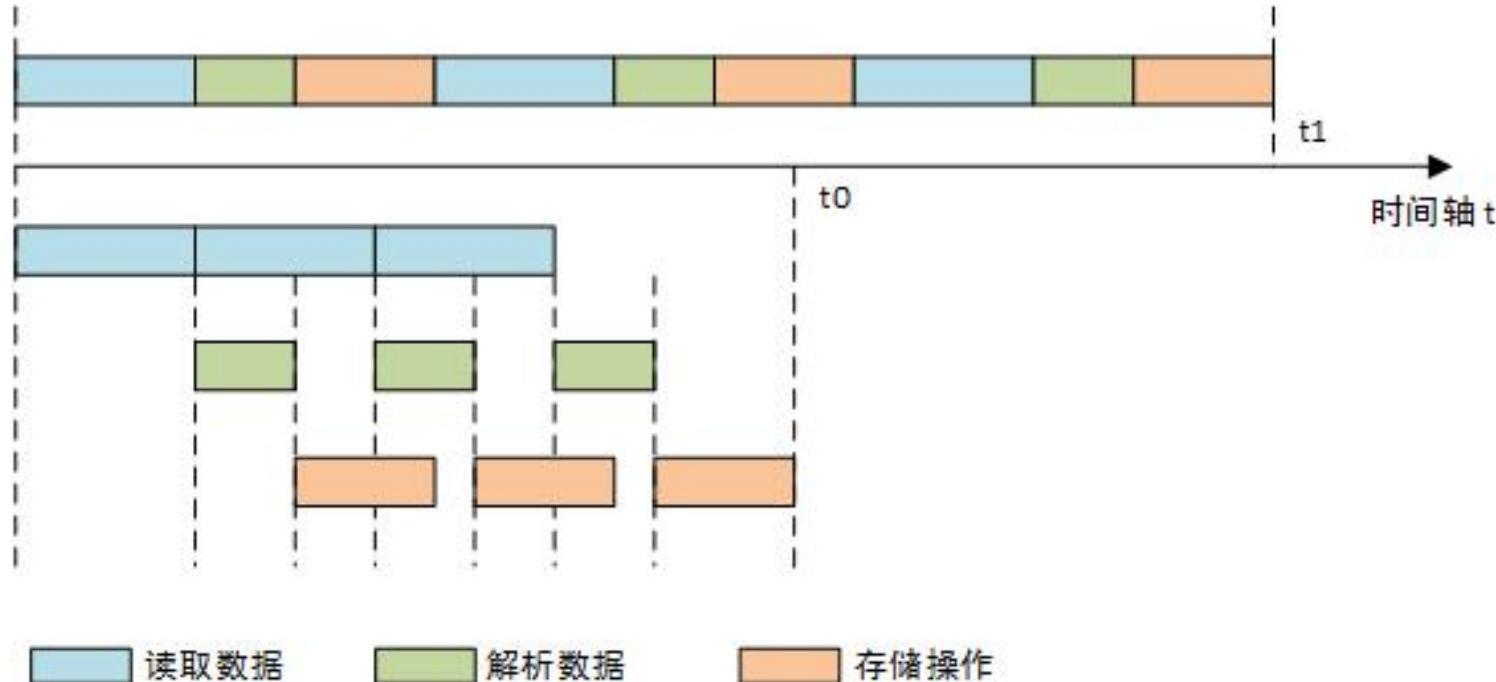


图1 流水化图

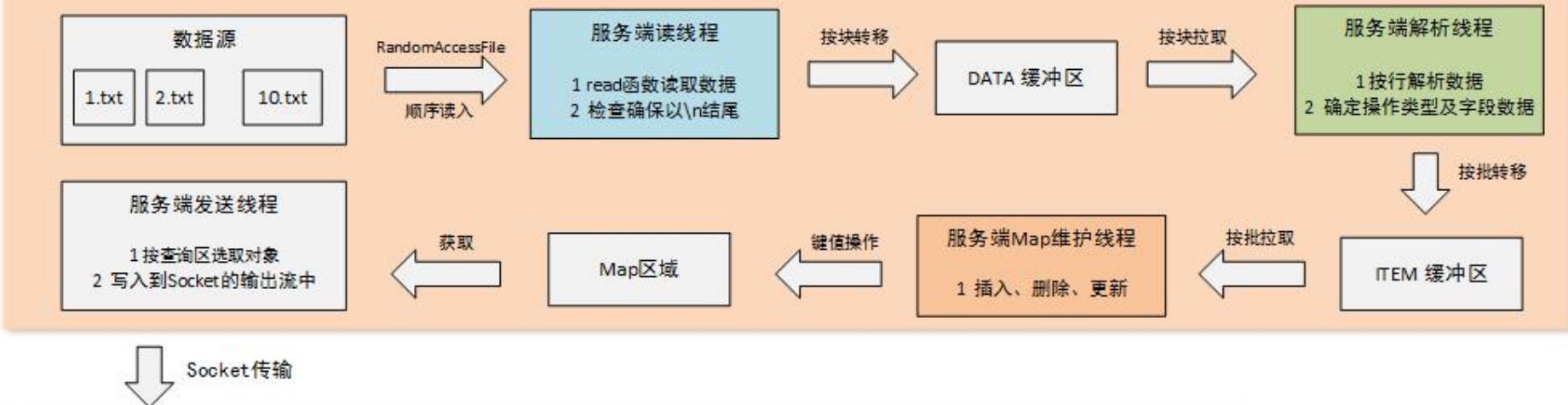
## 为什么不再细化流水线？

在测试中，当继续细化下去时，线程间数据转移的开销 > 细分带来的收益。

# 流水化处理 -- 整体流程

读取    解析    储存

## 服务端



## 客户端

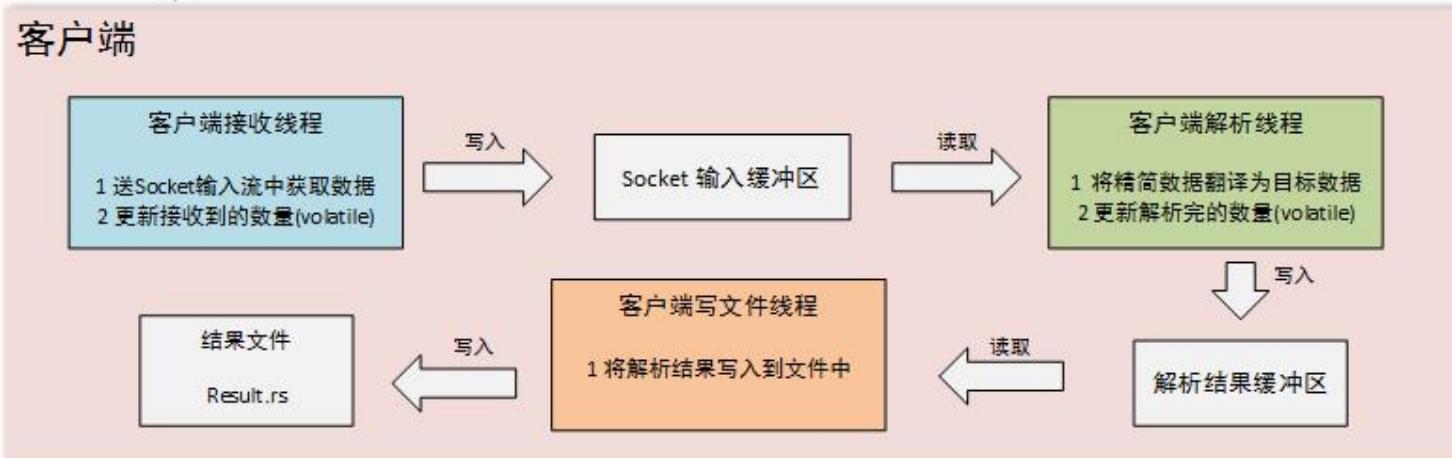


图2 整体流程图

3

# 数据结构

# 数据结构 -- 记录表示

表1 记录表示

| 字段 | score2   | sex | lastName2 | lastName1 | firstName | score   |
|----|----------|-----|-----------|-----------|-----------|---------|
| 说明 | 分数2      | 性别  | 名字部分2     | 名字部分1     | 姓         | 分数      |
| 范围 | 40-30039 | 男、女 | 1个中文      | 1个中文      | 1个中文      | 33-8414 |
| 位数 | 19       | 1   | 10        | 10        | 10        | 14      |

64位，可用long类型表示

## 优点：

- 1) 减少对象创建的开销。
- 2) 节约服务端发送数据到客户端的时间（一半以上）。

## 缺点：

- 1) 位运算使用不当容易出bug。
- 2) 增加了解析时间。（流水化设计可以抵消）

# 数据结构 -- 记录表示

表2 记录表示

| 字段 | score2   | sex | lastName2 | lastNam1 | firstName | score   |
|----|----------|-----|-----------|----------|-----------|---------|
| 范围 | 40-30039 | 男、女 | 1个中文      | 1个中文     | 1个中文      | 33-8414 |
| 位数 | 19       | 1   | 10        | 10       | 10        | 14      |

一个中文 *UTF-8* 表示需要3个字节24位, *Unicode*表示  
需要2个字节16位。如何用10位表示?

前提条件: 比赛中中文字符只有72个。

建立一个Map, 将中文映射到0-71序号, 7位就够了。

进一步: 访问Map有开销, 能否直接得到唯一标识?

即找这样一个单射函数:

$$f(x_i) = y_i, \forall x_i, x_j \in \{72个中文\}, x_i \neq x_j \Rightarrow y_i \neq y_j$$

我们采用的函数:  $f(x_i) = ((x_i[1] \ll 5) \wedge x_i[2]) \& 1023$

# 数据结构 -- 记录表示

为了减少位移次数，对字段位置进行了调整，调整原则是：频繁更新的放低位；字段位数少的放低位。

表3 不调整位置（红色为会更新字段）

| 字段 | firstName | lastName2 | lastName1 | sex | score | score2 |
|----|-----------|-----------|-----------|-----|-------|--------|
| 位数 | 10        | 10        | 10        | 1   | 14    | 19     |

表4 调整位置（红色为会更新字段）

| 字段 | score2 | sex | lastName2 | lastName1 | firstName | score |
|----|--------|-----|-----------|-----------|-----------|-------|
| 位数 | 19     | 1   | 10        | 10        | 10        | 14    |

表5 调整位置前后位移数变化

|        | 插入  | 更新 score | 更新 firstName | 更新 lastName |
|--------|-----|----------|--------------|-------------|
| 调整前位移数 | 184 | 19       | 54           | 34          |
| 调整后位移数 | 161 | 0        | 14           | 24          |

# 数据结构 -- Map结构

- 容器：1个数组 + n个IntLongHashMap。

```
long[] map1 = new long[SIZE1];
```

```
IntLongHashMap[] map2 = new IntLongHashMap[SIZE2];
```

- IntLongHashMap改写自从Netty中的IntObjectHashMap：1) 减少自动装拆箱；2) 内联并精简代码。
- map2 采用开放定址的方式解决hash冲突，访问设计如下表所示：

表6 map2 访问设计

| 主键位数    | 作用                       | 例子  |
|---------|--------------------------|---|
| 10 ~ 31 | 定位IntLongHashMap         | pk1 = 90,000 pk2 = 123,000<br>会进入不同的Map, 不会冲突         |
| 0 ~ 9   | 作为IntLongHashMap 中的hash值 | pk1 = 90,000 pk2 = 90,001<br>进入相同的Map,但有不同的hash, 不会冲突 |

缓冲区 = 循环队列（数组） + 信号量

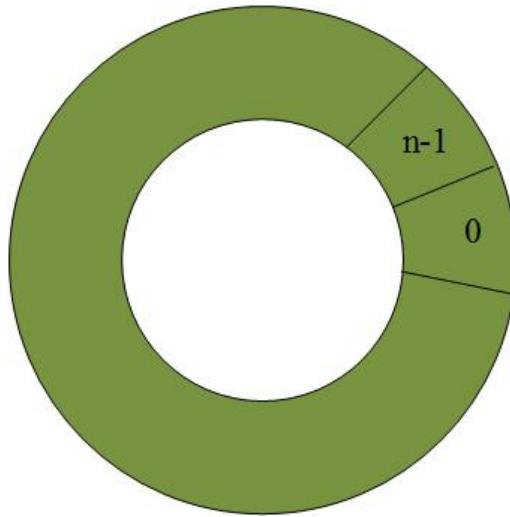


图3 循环队列示意

- 1) Semaphore FULL = new Semaphore(0);
- 2) Semaphore EMPTY = new Semaphore(n);

为什么不用 ArrayBlockingQueue?  
减少函数调用和泛型的开销。

4

## 解析处理

# 解析处理 -- 变更类型偏移预测

表7 更新记录格式

| binaryId | timestamp | schema | table | 变更类型               | ... |
|----------|-----------|--------|-------|--------------------|-----|
| 变长       |           | 固定长度   |       | I、U、D唯一，<br>可作为标志位 |     |

偏移 (offset) ?

- 1) 使用上次偏移预测:  $\text{predict\_offset} = \text{last\_offset}$
- 2) 预测失败重算并更新  $\text{last\_offset}$
- 3) 初始值 (不超出记录长度即可) :  $\text{last\_offset} = 19$

好处: 可减少binaryId处的解析处理。

# 解析处理 -- 是否更新主键预测

表8 更新主键

| ... | 主键1  | 主键2  | 分隔符 | 换行符 |
|-----|------|------|-----|-----|
|     | len1 | len2 |     |     |

表9 更新其它字段

| ... | 主键1  | 主键1  | 列信息 | 变更前列值 | 变更后列值 | 分隔符 | 换行符 |
|-----|------|------|-----|-------|-------|-----|-----|
|     | len1 | len1 |     |       |       |     |     |

表10 预测大致思想

| ... | 主键1  | ...  |
|-----|------|------|
|     | len1 | len1 |



直接偏移一定长度, 寻找标志位

好处: 预测正确可以少算一次主键。

# 解析处理 -- 是否更新主键预测

表11 预测问题

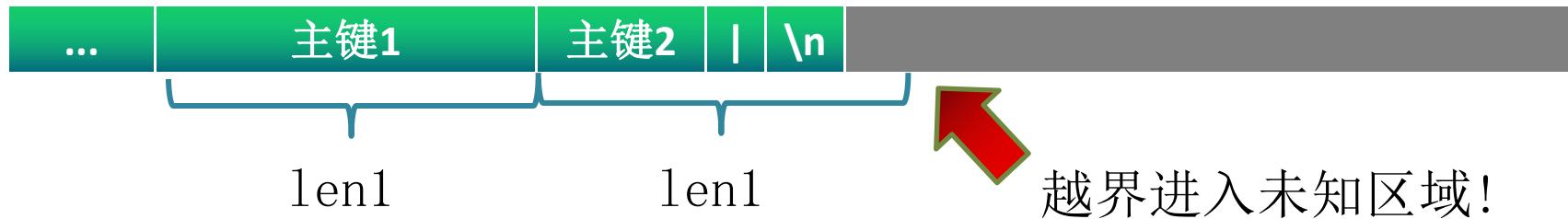


表12 预测中的一些问题

| 问题描述                               | 解决   |
|------------------------------------|--|
| 如果len1越界进入下一条记录，“ mysql-bin.”会造成误判 | 再移动4位<br>score -> e<br>first_name -> t<br>last_name -> _ |
| 由于缓冲区是复用的，len1可能越界后会受到旧数据的影响       | 超出数据块大小后放弃预测   |

# 解析处理 -- 转10进制优化

基本思路：循环展开并合并。

例：“123”转换成 123

$$sum = '1' - '0'; \quad (1)$$

$$sum = (sum << 3) + (sum << 1) + ('2' - '0'); \quad (2)$$

$$sum = (sum << 3) + (sum << 1) + ('3' - '0');$$

总共：4次左移操作 + 4次加运算 + 3次减法运算

将（1）式带入（2）式得：

$$sum = ('1' << 3) + ('1' << 1) + '2' - base; \quad (\text{其中 } base = 11 * '0')$$

$$sum = (sum << 3) + (sum << 1) + ('3' - '0');$$

省一次减法

总共：4次左移操作 + 4次加运算 + 2次减法运算

# 5

## 其它优化

# 其它优化 -- 网络传输优化

表13 网络传输优化过程

| 方案说明  | 耗时 (毫秒)   |
|---|-----------|
| 1) 服务端：使用Snappy压缩算法压缩最终结果，通过Netty传输给客户端<br>2) 客户端：启动一个线程接收数据、解压、写入文件  | 2000~3000 |
| 1) 服务端：通过Socket传输给客户端最终结果<br>2) 客户端：启动一个线程接收数据、写入文件   | 600~1000  |
| 1) 服务端：通过Socket传输中间结果 (int主键 + long记录)<br>2) 客户端：启动三个线程处理 <ul style="list-style-type: none"> <li>➤ 读取线程通过Socket接收中间结果</li> <li>➤ 解析线程解析中间结果</li> <li>➤ 存储线程将解析后的结果写入文件</li> </ul> | 300~350   |

# 其它优化 -- 参数调优

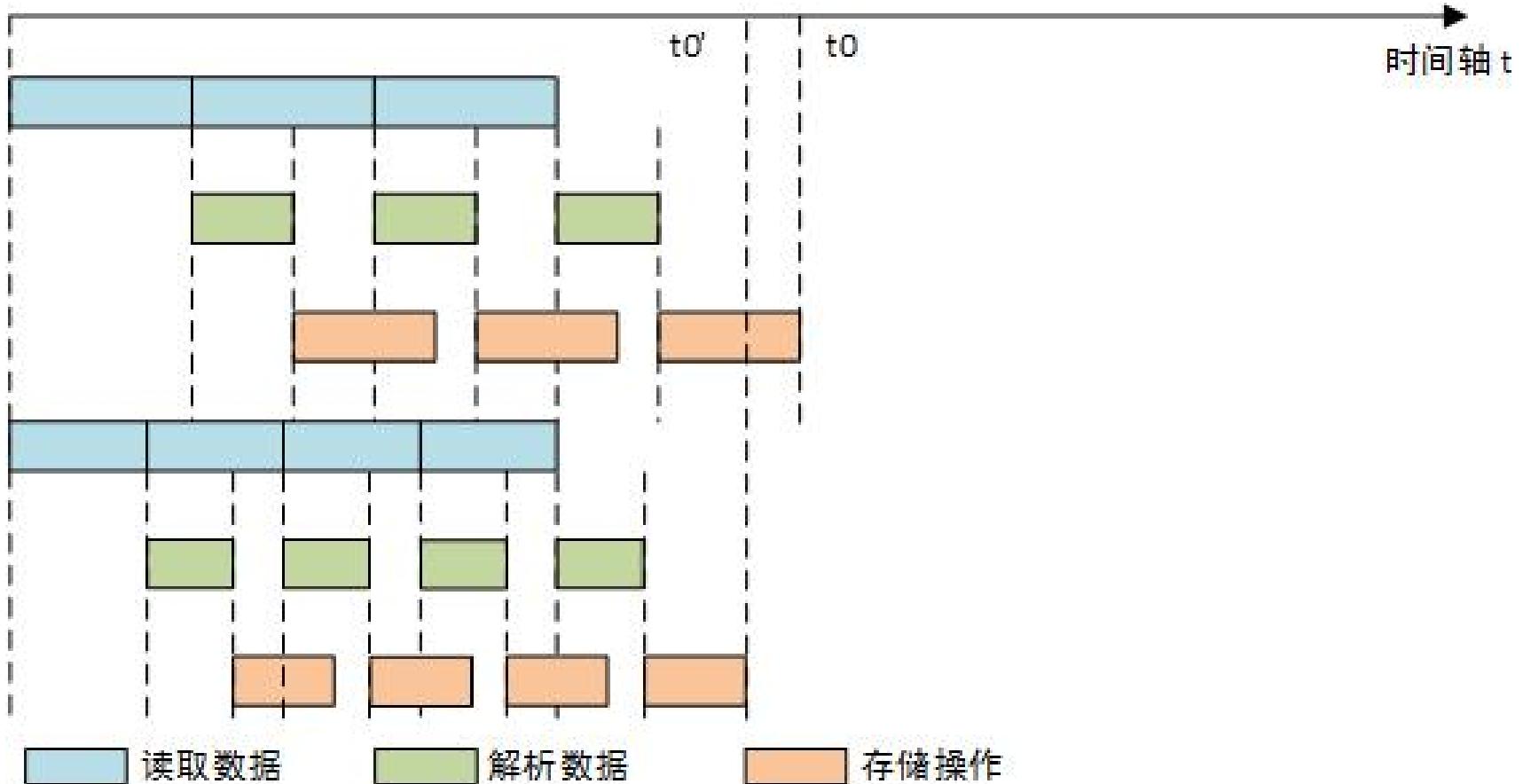


图4 流水化块大小影响

块大小影响流水线的时间，所以可对块大小进行调参。

# 其它优化 -- 参数调优

表14 读取块大小对处理耗时的影响

| 读取块大小 (M) | 2   | 4   | 5   | 8   | 16  | 32  |
|-----------|-----|-----|-----|-----|-----|-----|
| 处理耗时(S)   | 4.5 | 4.3 | 4.2 | 4.5 | 4.8 | 5.5 |

读取块大小对处理耗时影响

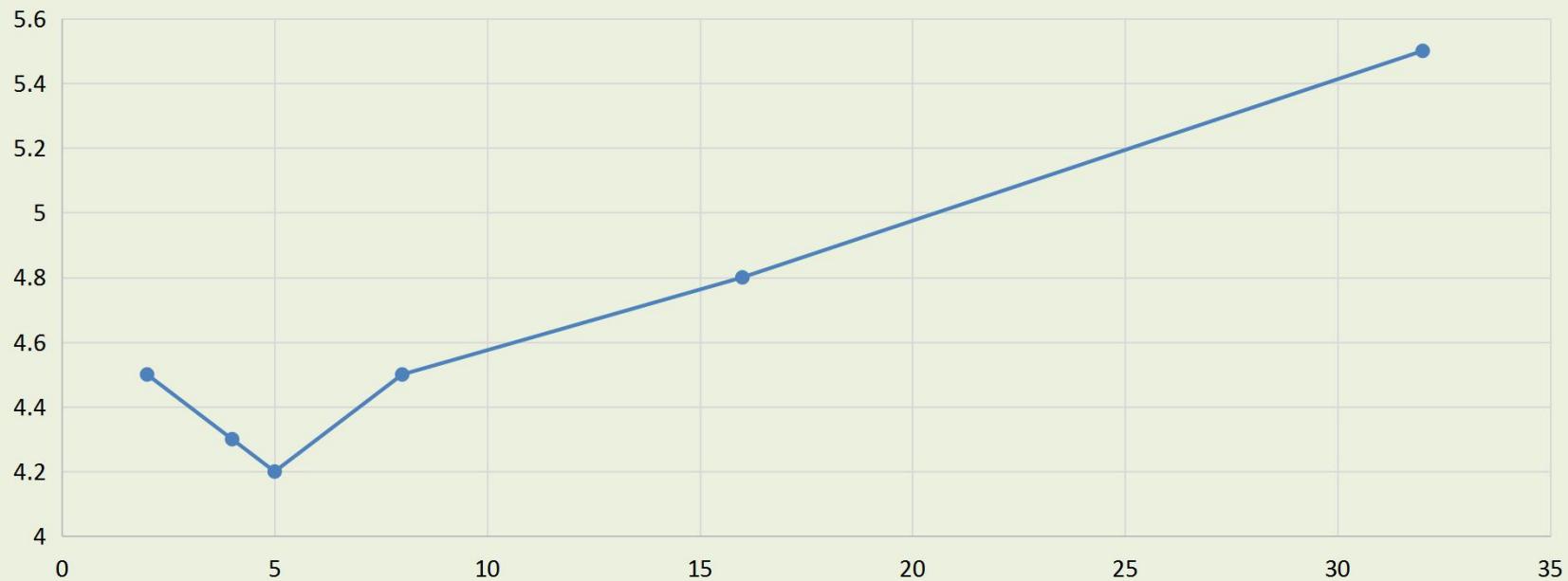


图5 读取块大小对处理耗时的影响

6

## 总结

# 总结 -- 优化总结

表15 优化总结

| 优化方法                | 有用思想              |
|---------------------|-------------------|
| 客户端、服务端都采用了多线程流水线处理 | 合理细分任务            |
| 记录用long表示记录         | 数据表示可以细化到位        |
| 解析过程中的预测            | 预测技术              |
| 缓冲区的设计              | 不一定要用jdk中的容器      |
| map的优化              | 要关注hash冲突         |
| 网络传输优化              | 可以约定数据的表示，减少数据传输量 |
| 块大小参数调整             | 控制任务的大小           |
| 转10进制优化             | 循环展开，合并计算         |

# 总结 -- 数据特性依赖

表16 最终版本的数据特性依赖

| 最佳成绩    | 依赖的数据特性                 | 受限制原因                  | 解决方案                    |
|---------|-------------------------|------------------------|-------------------------|
| 5283 ms | 插入主键递增                  | 直接使用一个递增<br>int变量      | 增加插入主键解析代码              |
|         | 中文个数、score、<br>score2范围 | long位数有限，增加<br>后不能表示完全 | 增加long的个数或者使<br>用其它表示方案 |
|         | 每次只更新一个字<br>段           | 会影响更新类型预<br>测          | 去掉预测，正常解析               |

# 总结 -- 感想

首先，感谢天池和中间件团队举办这次比赛！

其次，在参赛过程中，我们不仅提高了技术能力，也积累了经验心得，更对所学到的知识有了一个更加深刻的认识！这真是一次难忘的编程体验！

Thanks !