

股票市场的主力 资金流向计算

分布式与并行计算project的汇报

组别3：
欧炜娟 李怡萱



目录

CONTENTS

01 项目介绍 Introduction

02 方案变革 Solution Evolution

03 最终方案 Final Solution

04 结果分析 Result Analysis

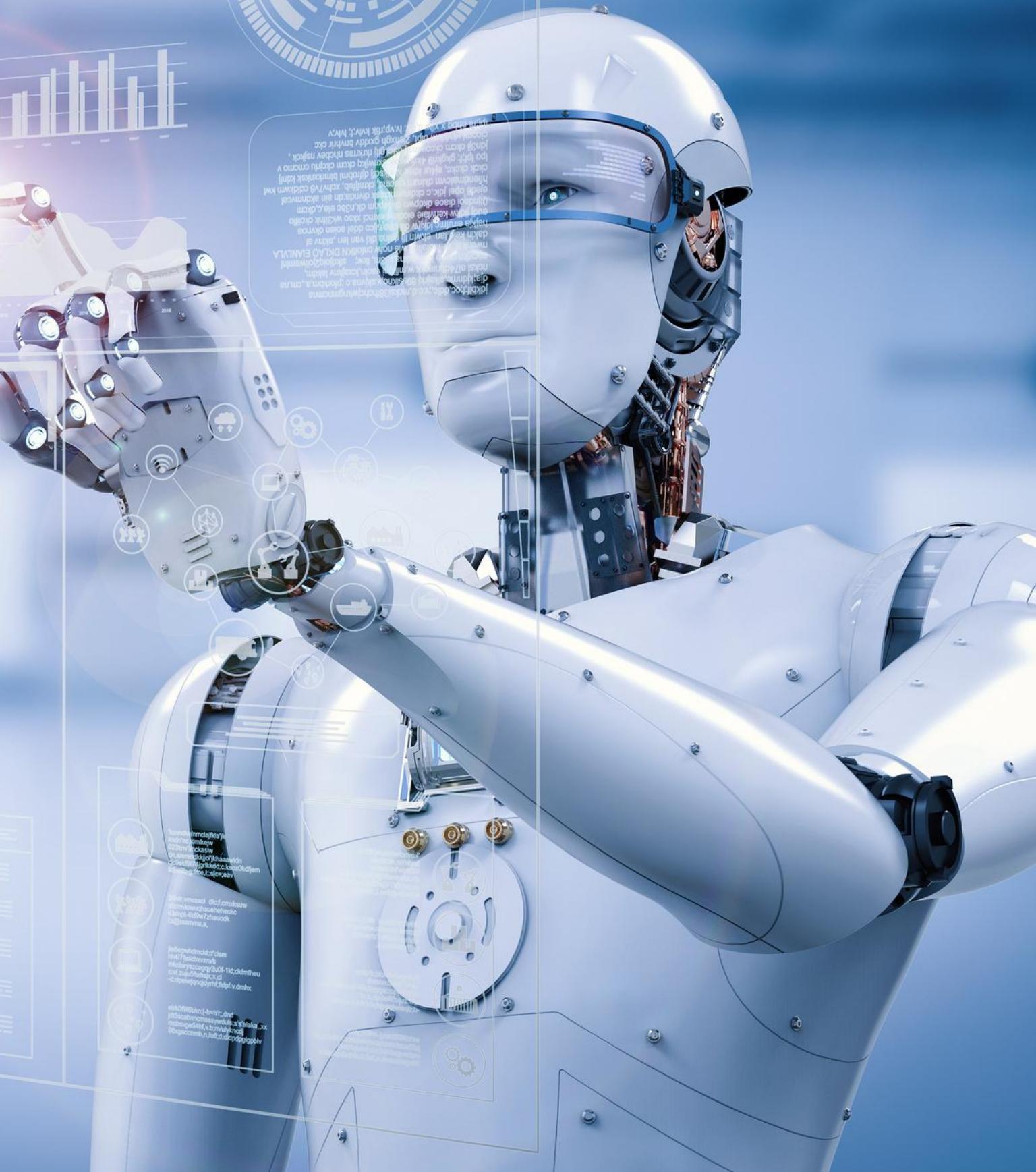
05 感想总结 Thinkings



PART 01

项目介绍

股票市场的主力资金流向计算



背景介绍

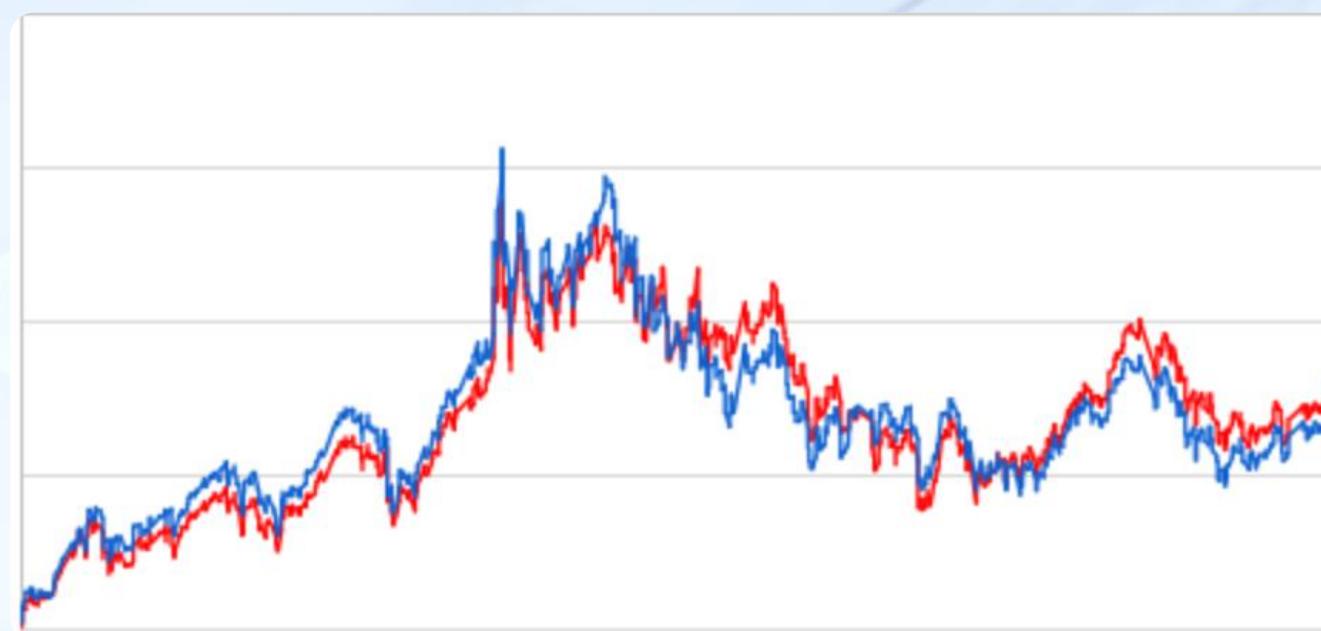
主力资金的介绍



- **定义：**
主力资金：大资金交易方（如机构投资者），影响股票价格走势。
- **重要性：**
主力资金流向：判断市场趋势、辅助投资决策。
- **困难性：**
难以追踪：缺乏逐笔订单投资者身份信息，需间接分析成交特征。



背景介绍



本项目简单介绍

- 数据来源：
 - 深交所 Level-2 数据：逐笔委托与逐笔成交
- 筛选有效成交数据和判断主动单子
- 计算相同索引的总成交量和成交额
- 判断分类成交单类型：超大单、大单、中单、小单
- 判断主力资金流向：
 - 统计大单和超大单：主力流入与主力流出
 - 计算每个类型单子的买卖情况
- 时间窗口划分：
 - 按时间切分数据，计算每个窗口内资金流向
- 输出主力资金流向等数据，供分析和决策使用

PART 02

方案的变革

方案一：五个独立Job

方案二：三个Job的

方案三：一个Job——显著提升数据处理效率和资源利用率

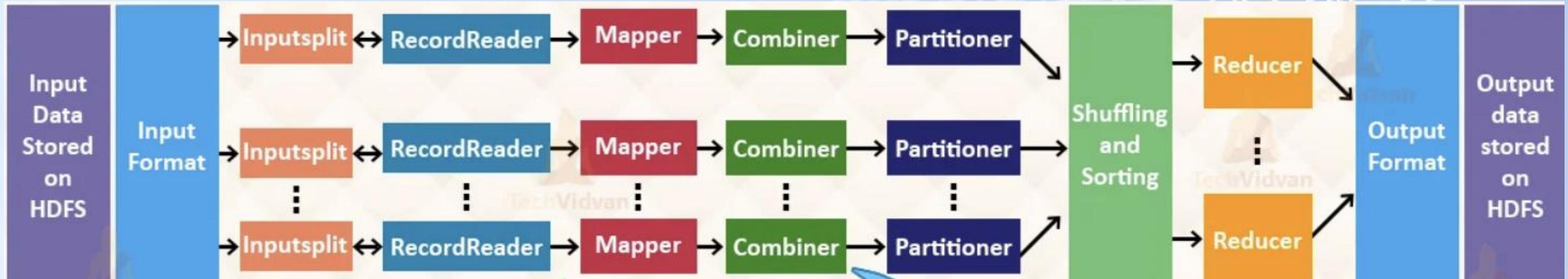


方案一

直觉型出方案



“试水方案”



job1 & 2

- Order表预处理：
提取有效的委托数据信息
- Trade表预处理：
提取有效交易数据信息；
计算时间窗口id

job 3

- 数据关联：将委托与交易
数据匹配
- 主动单识别：区分主动买
单与卖单
- 输出结果：主动单集合

job 4

- 聚合相同索引的订单：计算每
个时间窗口，同一个索引的总
成交量与成交额
- 输出结果：每个时间窗口同一
个索引的相对应信息

job 5

- 判断单子类型：根据总成交额，
总成交量和总流通盘定义每个索
引的对应的单子类型
- 主力流向分析：计算流入、流出、
净流入和每个类型单子买卖数额
等指标

方案一的问题

耗时太久啦！！！

- 数据预处理：
 - 只处理上午数据耗时约 1 分半
- 表格合并：
 - 两张表格合并耗时超 5 分钟

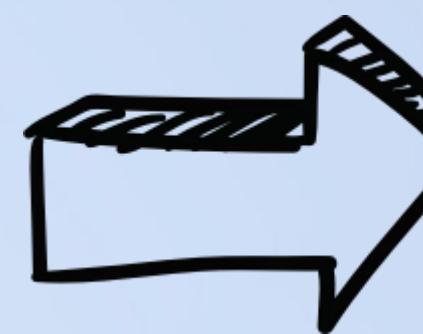


我哭死！！！时间怎么这么多



改进？

方案二的诞生 order (不要啦~)



“索引直通车计划” trade (还要的~)

pm_hq_order_spot.txt

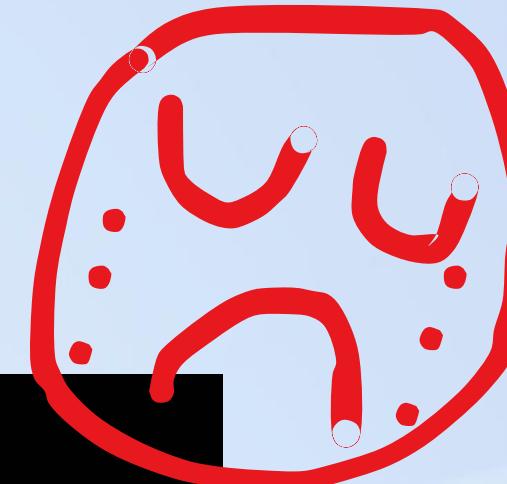
20190102	201901021300000000	201901021300000000	201901021300000000	201901021300000000
2019010213000067	2012 011	4588753 002454 102	6.070000	28000
2019010213000000	1 2	0.0 NULL NULL	0 0	
20190102	201901021300000000	2019010213000059	2019010213000066	
2019010213000072	2012 011	4588754 300537 102	10.110000	100
2019010213000000	1 2	0.0 NULL NULL	0 0	
20190102	201901021300000000	2019010213000059	2019010213000066	
2019010213000072	2022 011	49213 150201 102	0.635000	33300
2019010213000000	1 2	0.0 NULL NULL	0 0	
20190102	201901021300000000	2019010213000059	2019010213000067	
2019010213000073	2012 011	4588756 300548 102	38.490000	7500
2019010213000000	1 2	0.0 NULL NULL	0 0	
20190102	201901021300000000	2019010213000059	2019010213000067	
2019010213000073	2012 011	4588764 300098 102	7.120000	100
2019010213000000	1 2	0.0 NULL NULL	0 0	
20190102	201901021300000000	2019010213000059	2019010213000067	
2019010213000073	2012 011	4588765 002899 102	14.630000	20000
2019010213000000	1 2	0.0 NULL NULL	0 0	
20190102	201901021300000000	2019010213000060	2019010213000068	
2019010213000003	2012 011	4588786 002899 102	14.610000	20000
2019010213000000	1 2	0.0 NULL NULL	0 0	
20190102	201901021300000000	2019010213000060	2019010213000068	
2019010213000073	2012 011	4588787 002217 102	4.640000	1000
2019010213000000	1 2	0.0 NULL NULL	0 0	
20190102	201901021300000000	2019010213000060	2019010213000068	
2019010213000073	2012 011	4588788 002450 102	7.290000	5700
2019010213000000	1 2	0.0 NULL NULL	0 0	
20190102	201901021300000000	2019010213000060	2019010213000068	
2019010213000073	2012 011	4588789 300098 102	7.240000	60000
2019010213000000	2 2	0.0 NULL NULL	0 0	

pm_hq_trade_spot.txt

20190102	201901021300000000	201901021300000000	201901021300000000	201901021300000000
2019010213000067	2012 011	4588755 300537 102	4588754 4584371 9.300000	100
F	201901021300000000	201901021300000000	201901021300000000	201901021300000000
20190102	201901021300000000	2019010213000059	2019010213000067	300
F	201901021300000000	201901021300000000	201901021300000000	201901021300000000
20190102	201901021300000000	2019010213000059	2019010213000067	800
F	201901021300000000	201901021300000000	201901021300000000	201901021300000000
20190102	201901021300000000	2019010213000059	2019010213000067	1000
F	201901021300000000	201901021300000000	201901021300000000	201901021300000000
20190102	201901021300000000	2019010213000059	2019010213000067	1400
F	201901021300000000	201901021300000000	201901021300000000	201901021300000000
20190102	201901021300000000	2019010213000059	2019010213000067	300
F	201901021300000000	201901021300000000	201901021300000000	201901021300000000
20190102	201901021300000000	2019010213000059	2019010213000067	3600
F	201901021300000000	201901021300000000	201901021300000000	201901021300000000
20190102	201901021300000000	2019010213000059	2019010213000067	500
F	201901021300000000	201901021300000000	201901021300000000	201901021300000000
20190102	201901021300000000	2019010213000059	2019010213000067	1100
F	201901021300000000	201901021300000000	201901021300000000	201901021300000000

- 索引递增：避免额外关联查询，提高效率
- 直接比较索引来判断主动单：BidApplSeqNum 和 OfferApplSeqNum
- 无需查找时间戳

方案二的问题



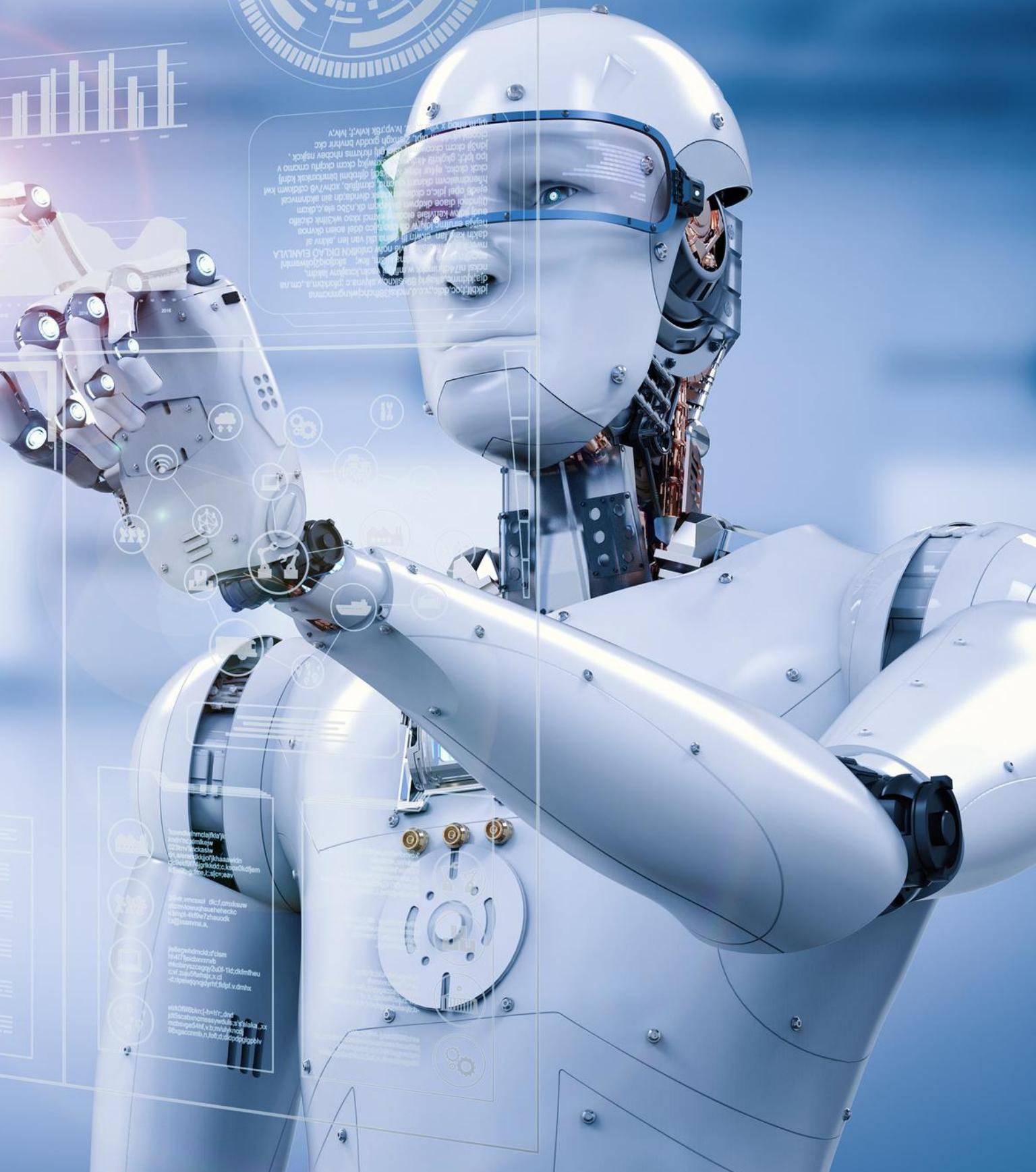
还要两分钟呀！怎么办！！

- 问题：耗时过多
 - 启动 3 个 job，用时较长
 - 筛选有效信息：1 分半钟
 - 每个 job 启动时间：十几秒
- 改进建议（方案三）
 - 合并 Job：减少启动次数，提高效率
- 关键点分析
 - Key 的选择：合并时的核心优化点。
 - 流程优化：将后续流程合并至一个 Reducer 完成

PART 03

最后的方案

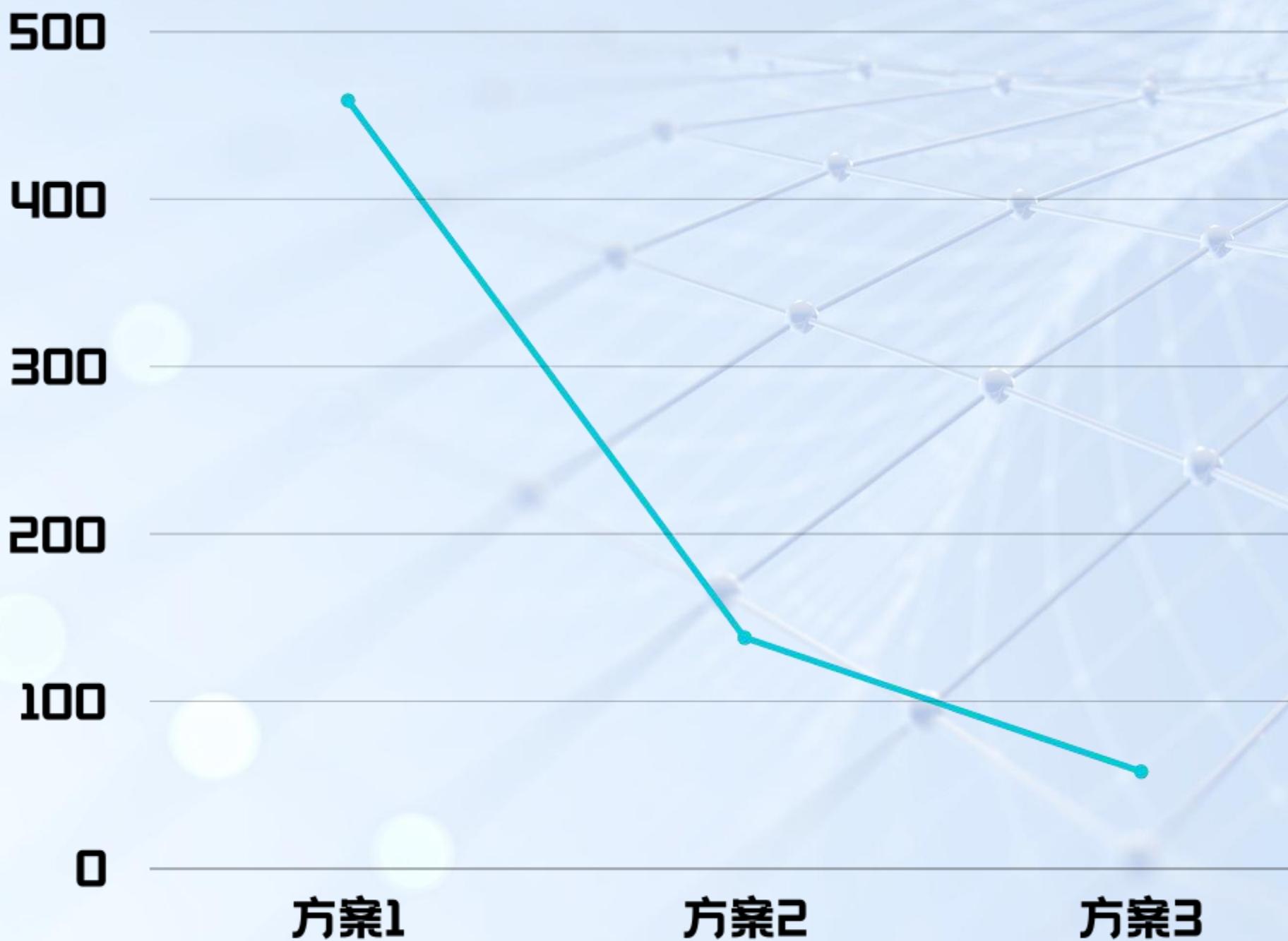
方案的亮点 + 流程图 + 代码模块的讲解



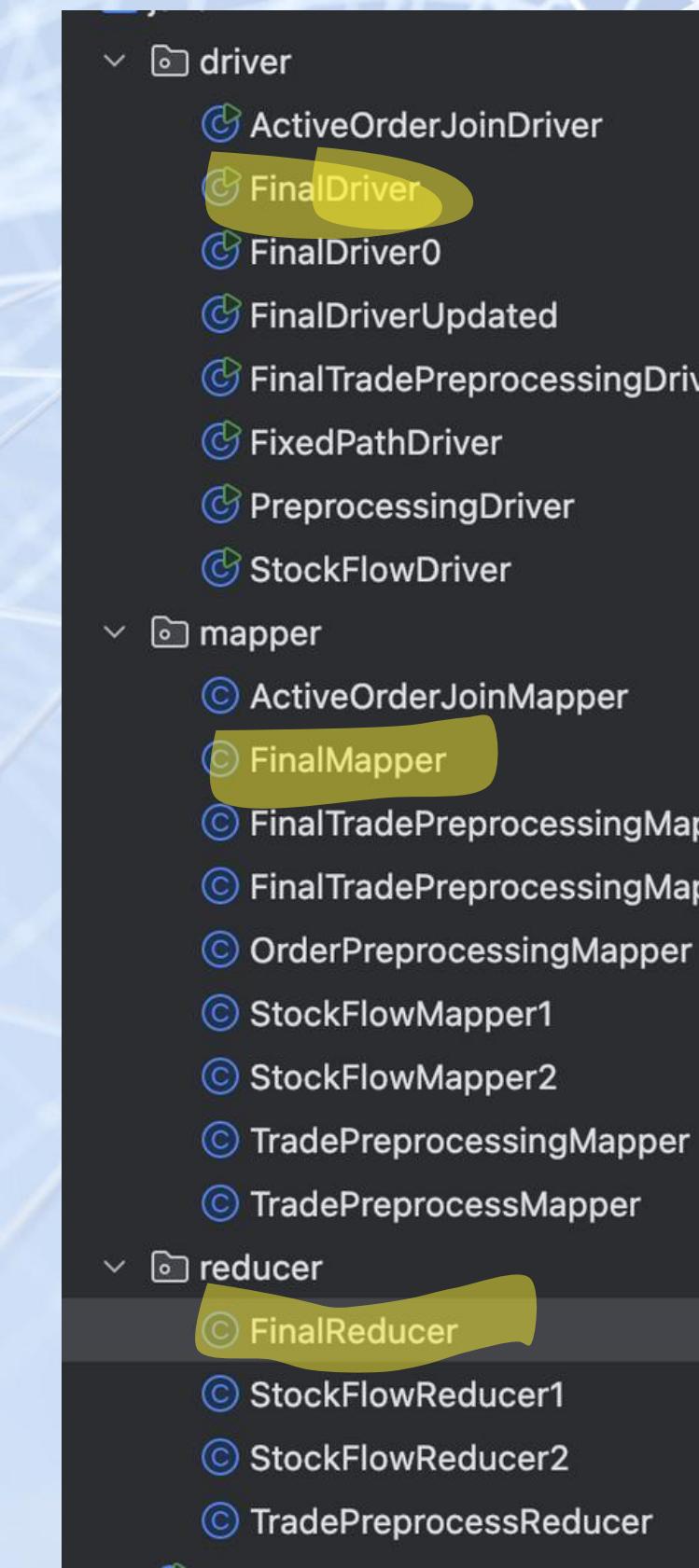
聪明又勤快的最终方案

时间的加速！！！

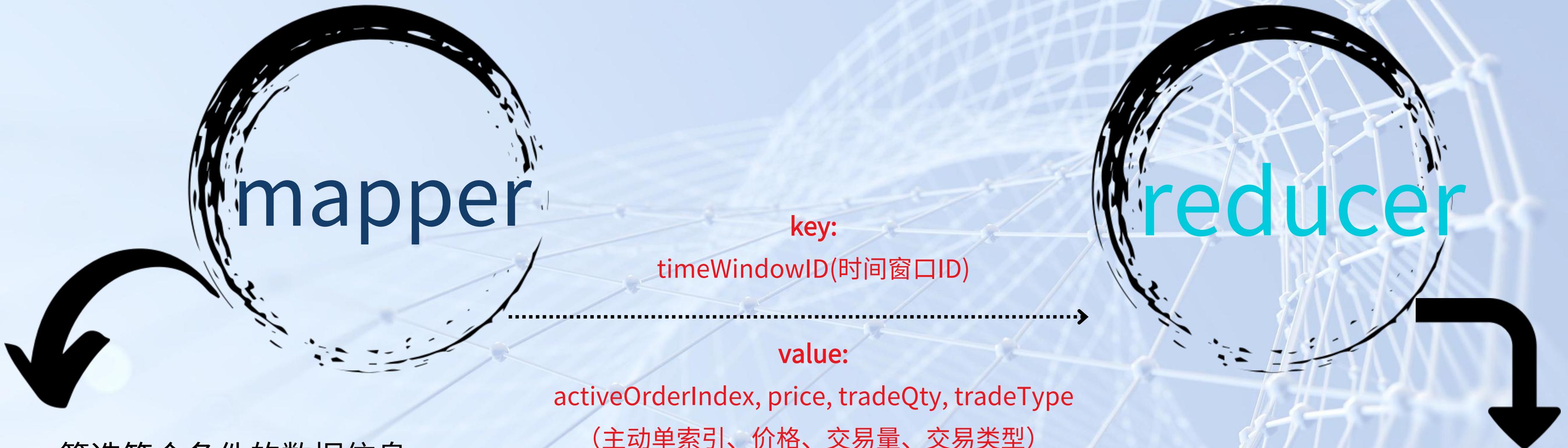
(几乎8分钟到55秒)



整体框架的不断优化
(6个job 到 1个job)

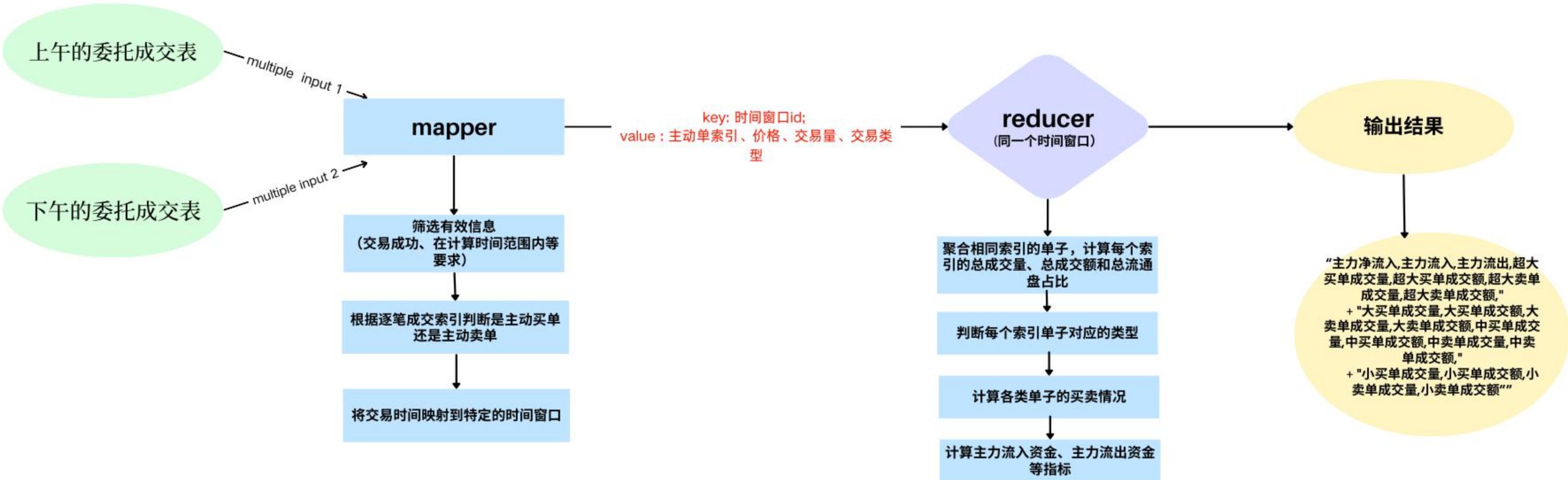


方案三的结构



- 筛选符合条件的数据信息
- 基于时间窗口对交易记录进行分组
- 判断主动单合集

- 聚合相同索引的单子
- 数据分类统计
- 主力资金流向分析
- 时间窗口补全



方案三mapper

筛选符合条件的数据 & 获取主动单的索引

筛选：

```
1 // 处理符合条件的记录：成交类型为F，股票代码为000001。  
2 if ("F".equals(execType) && "000001".equals(securityID)) {  
3     if (isWithinTradingTime(tradeTime)) {  
4         // 进行后续的时间窗口计算、交易类型判断，并写入结果  
5     }  
6 }
```

- ExecType = F: 筛选执行类型为 "F" 的有效交易记录
- SecurityID = 000001: 筛选证券代码为 "000001" 的特定股票
- 交易时间：
 - 早上有效时段：9:30 到 11:30
 - 下午有效时段：13:00 到 15:00

判断主动买卖单：

```
1 long bidApplSeqNum = Long.parseLong(records[10]);  
2 long offerApplSeqNum = Long.parseLong(records[11]);  
3  
4 // 确定交易类型并获取主动单索引  
5 String activeOrderIndex = (bidApplSeqNum > offerApplSeqNum) ?  
6     String.valueOf(bidApplSeqNum) : String.valueOf(offerApplSeqNum);  
7 int tradeType = (bidApplSeqNum > offerApplSeqNum) ? 1 : 2;
```

流程优化：简化主动单判断，大幅提升技术方案效率

方案三mapper 如何计算时间窗口

在 FinalMapper 中，我们将每个交易时间（ tradeTime ）映射到一个特定的时间窗口ID，并将其作为 key 传递给 reducer

```
// 将时间字符串 (yyyyMMddHHmmssSSS) 转化为分钟
public static int getTimeInMinutes(long tradetime) {
    // 转换 tradetime
    String timeStr = String.valueOf(tradetime).substring(8, 12); // 提取 "HHmm"
    int hour = Integer.parseInt(timeStr.substring(0, 2));
    int minute = Integer.parseInt(timeStr.substring(2, 4));

    // 将时间转换为从午夜开始的分钟数
    return hour * 60 + minute;
}
```

```
1 // 计算时间窗口ID
2 public static long calculateTimeWindowID(long tradetime) {
3     ...
4     if (currentTimeInMinutes >= morningStartInMinutes &&
5         currentTimeInMinutes <= morningEndInMinutes) {
6         // 早上 9:30 - 11:30 的时间段，计算属于哪个窗口
7         timeWindowID = (currentTimeInMinutes - morningStartInMinutes) /
8             TIME_WINDOW + 1;
8     } else if (currentTimeInMinutes >= afternoonStartInMinutes &&
9         currentTimeInMinutes < afternoonEndInMinutes) {
10        long interval = (morningEndInMinutes - morningStartInMinutes) /
11            TIME_WINDOW;
11        // 下午 13:00 - 15:00 的时间段，计算属于哪个窗口
12    }
13}
```

getTimeInMinutes() 方法：

- 转换为 从午夜开始的分钟数

calculateTimeWindowID 方法：

- 将交易时间映射到早上和下午两个交易时段内的
特定时间窗口。

具体计算公式为：

$$\text{早上: 时间窗口 ID} = \frac{\text{目前分钟时间} - \text{早上开始分钟时间}}{\text{时间窗口参数}} + 1$$

$$\text{下午: 时间窗口 ID} = \frac{\text{目前分钟时间} - \text{下午开始分钟时间}}{\text{时间窗口参数}} + \text{早上的窗口数目} + 1$$

方案三 reducer

使用 HashMap 存储每个主动委托索引聚合后的数据

- 成交量与成交额：按时间窗口聚合计算
- 流通盘占比：根据 CIRCULATION_STOCK 计算为 成交量 / 流通股总量
- 数据聚合：使用 HashMap 存储每个 activeOrderIndex 的交易信息（成交量、成交额、买卖类型）

```
// 存储每个主动委托索引的累计成交量、成交额和买卖类型
Map<String, Object[]> activeOrderData = new HashMap<>();

// 遍历所有值，累加每个主动委托索引的成交量和成交额，并记录买卖类型
for (Text value : values) {
    String[] fields = value.toString().split("\t");

    try {
        String activeOrderIndex = fields[0].trim(); // 主动委托索引
        ...
        activeOrderData.putIfAbsent(activeOrderIndex, new Object[]{0.0, 0.0,
tradeType}); // 根据相同的主动委托索引，累加该主动委托索引对应成交单数据
        Object[] data = activeOrderData.get(activeOrderIndex);
    }
}
```

```
// 遍历所有值，累加每个主动委托索引的成交量和成交额，并记录买卖类型
for (Text value : values) {
    String[] fields = value.toString().split("\t");

    try {
        String activeOrderIndex = fields[0].trim(); // 主动委托索引
        double price = Double.parseDouble(fields[1]); // 成交价格
        double tradeQty = Double.parseDouble(fields[2]); // 成交量
        double amount = price * tradeQty; // 成交金额
        ...
        Object[] data = activeOrderData.get(activeOrderIndex); // 遍历该索引对应的单
        data[0] = (double) data[0] + tradeQty; // 累加成交量
        data[1] = (double) data[1] + amount; // 累加成交额
        data[2] = tradeType; // 更新买卖类型（确保一致）
    } catch (NumberFormatException e) {
    }
}
```

方案三 reducer

单子类型分类规则

成交单类型（超大单、大单、中单、小单）对买单和卖单进行分类

判断方式	超大单	大单	中单	小单
成交量	$\geq 200,000$	$\geq 60,000$	$\geq 10,000$	其他
成交额	$\geq 1,000,000$	$\geq 300,000$	$\geq 50,000$	其他
流通盘占比	≥ 0.003	≥ 0.001	≥ 0.00017	其他

```
1 // 确定单子类型索引
2 int orderTypeIndex;
3 if (totalTradeQty >= 200000 || totalAmount >= 1000000 || circulationRatio >=
4 {
    orderTypeIndex = 0; // 超大单
5 } else if (totalTradeQty >= 60000 || totalAmount >= 300000 || circulationRatio >=
6 0.001) {
    orderTypeIndex = 1; // 大单
7 } else if (totalTradeQty >= 10000 || totalAmount >= 50000 || circulationRatio >=
8 0.00017) {
    orderTypeIndex = 2; // 中单
9 } else {
    orderTypeIndex = 3; // 小单
10 }
```

方案三 reducer

主力资金流动计算公式

- 单类型数据合并：根据成交单判断结果和买卖类型累加相同类型的数据
- 主力资金计算：按照公式计算 主力流入、主力流出 和 主力净流入

```
1 // 根据买卖类型累加数据
2 if (tradeType == 1) { // 买单
3     buyQty[orderTypeIndex] += totalTradeQty;
4     buyAmount[orderTypeIndex] += totalAmount;
5     if (orderTypeIndex == 0 || orderTypeIndex == 1) {
6         mainFlowIn += totalAmount; // 计算主力流入
7     }
8 } else if (tradeType == 2) { // 卖单
9     sellQty[orderTypeIndex] += totalTradeQty;
10    sellAmount[orderTypeIndex] += totalAmount;
11    if (orderTypeIndex == 0 || orderTypeIndex == 1) {
12        mainFlowOut += totalAmount; // 计算主力流出
13    }
14 }
15 // 计算主力净流入
16 netMainFlow = mainFlowIn - mainFlowOut;
```

主力流入 = 超大买单金额 + 大买单金额

主力流出 = 超大卖单金额 + 大卖单金额

主力净流入 = 主力流入 - 主力流出

方案三 reducer

小细节：判断时间窗口是否有缺漏

- 检测指标 index：跟随时间窗口处理自增
- 跳过检测：识别时间窗口数据缺失
- 补全输出：填补缺失时间窗口的数据，确保时间数据完整性

```
1 if (index < timeWindowID){  
2     long initial = index;  
3     for (long i = initial; i < key.get(); i++){  
4         String timeInterval = calculateTimeInterval(index);  
5         context.write(new Text("0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0"), new  
6 Text(timeInterval)); // 补足缺失的时间区间  
7         index++;  
8     }  
9 }
```

PART 04

结果分析

准确性 + 时间提升 + 可视化



可视化的展示

PART 05

感想总结

准确性 + 时间提升 + 可视化



感想总结

1. 时间优化：

- 分析瓶颈
- 挖掘数据处理规律
- 深刻理解计算逻辑

2. 优化成果：

- 提升程序性能
- 数据驱动决策的验证
- 技术应用成就感

3. 未来启示：

- 全局视角与细节优化
- 业务需求与技术结合
- 理解业务逻辑是开发成功的关键

感谢您的观看

分布式与并行计算报告的汇报

欧炜娟 李怡萱

2024年12月

