股票市场的主力资金流向计算

谢子成 徐锦焜

{12212609,12210613}@mail.sustech.edu.cn

1 问题分析

在本项目中，我们旨在通过深交所 Level-2 数据（逐笔委托和逐笔成交数据）计算某支股票在给定时间窗口内的主力流入、主力流出和主力净流入等关键指标。然而，**问题的复杂性主要体现在数据特征、时间窗口划分、主动成交的识别，以及数据聚合与指标计算上**。以下是针对问题的详细分析：

1.1 数据特征

深交所 Level-2 数据由两部分组成：逐笔委托数据和逐笔成交数据，每种数据具有不同的特征和用途。

1.1.1 逐笔委托数据

逐笔委托数据记录了每笔订单的委托价格、 数量、 时间以及买卖方向等详细信息。 主要字段包括：SecurityID（股票代码）、Price（委托价格）、OrderQty（委托数量）、TransactTime（委托时间）、 ApplSeqNum（委托索引） 等。 在本项目中， 其实可以直接使用逐笔成交数据完成所有的计算。

在本项目中，逐笔委托数据主要用于关联分析，但实际上可以直接使用逐笔成交数据完成所有的计算。

1.1.2 逐笔成交数据

记录了每笔交易的成交价格、 成交数量、 买卖双方的委托索引以及成交时间。 主要字段包括：SecurityID（股票代码）、Price（成交价格）、TradeQty（成交数量）、BidApplSeqNum（买方委托索引）、OfferApplSeqNum（卖方委托索引）、ExecType（成交类型）等。

挑战：需要分析买卖双方数据以确定主动方，并基于成交量和成交额分别计算买卖方的主力流动，确保分类和计算的准确性。此外，逐笔成交数据中包含大量与本项目无关的冗余条目，处理过程中需要进行筛选和清理。

1.2 时间窗口划分

1.2.1 时间窗口的定义

时间窗口是项目计算的基本单位，基于逐笔成交记录的tradetime字段，将数据按用户输入的时间间隔（如10分钟或20分钟）划分为固定区间。

1.2.2 时间窗口的实现

每条逐笔成交记录通过tradetime字段与时间窗口范围进行匹配，从而将其归类到对应的时间窗口中。

挑战：收盘集合竞价时间段的所有成交数据都会被标记为在最后时刻成交，因此需要对最后一个时间窗口的右端点取闭，以避免其对连续竞价结果的影响。此外，如何并行处理多个时间窗口以实现效率最大化也是一个重要的技术难点。

1.3 主动成交识别

主动成交是由新提交的委托单触发的成交， 具体表现为与订单簿中已有的对手方委托单匹配成功并立即成交。 **在本项目中， 可以直接比较买卖双方索引（BidApplSeqNum 和 OfferApplSeqNum）的大小来判断主动成交方，索引较大的为主动方**。

1.4 数据的聚合与指标计算

1.4.1 数据去重与合并

由于一个委托单可能对应多笔成交单，因此需要根据委托索引（ApplSeqNum）对同一时间窗口内的主动成交记录进行去重合并。

合并逻辑：对相同索引的成交记录，累计其成交量（TradeQty）和成交额（TradeQty * Price），从而得到统一的聚合结果。

1.4.2 分类成交量级

根据每笔合并后的成交量、成交额以及流通盘占比，划分为**超大单、大单、中单、小单**四类。分类规则基于行业标准（我们采用华泰金工 Insight 的分类标准）。

1.4.3 主力资金指标计算

基于分类结果，计算主力资金相关指标，包括主力流入、主力流出和主力净流入：

- **主力流入**：时间窗口内所有超大单和大单的主动买入金额之和。
- **主力流出**：时间窗口内所有超大单和大单的主动卖出金额之和。
- **主力净流入**：主力流入减去主力流出。

2 方案介绍

本项目基于 **深交所 Level-2 数据**，通过 MapReduce 框架实现对股票主力资金流向的计算，目标是输出每个时间窗口内的主力流入、主力流出和主力净流入等指标。以下为实现方案的详细描述：

2.1 总体设计思路

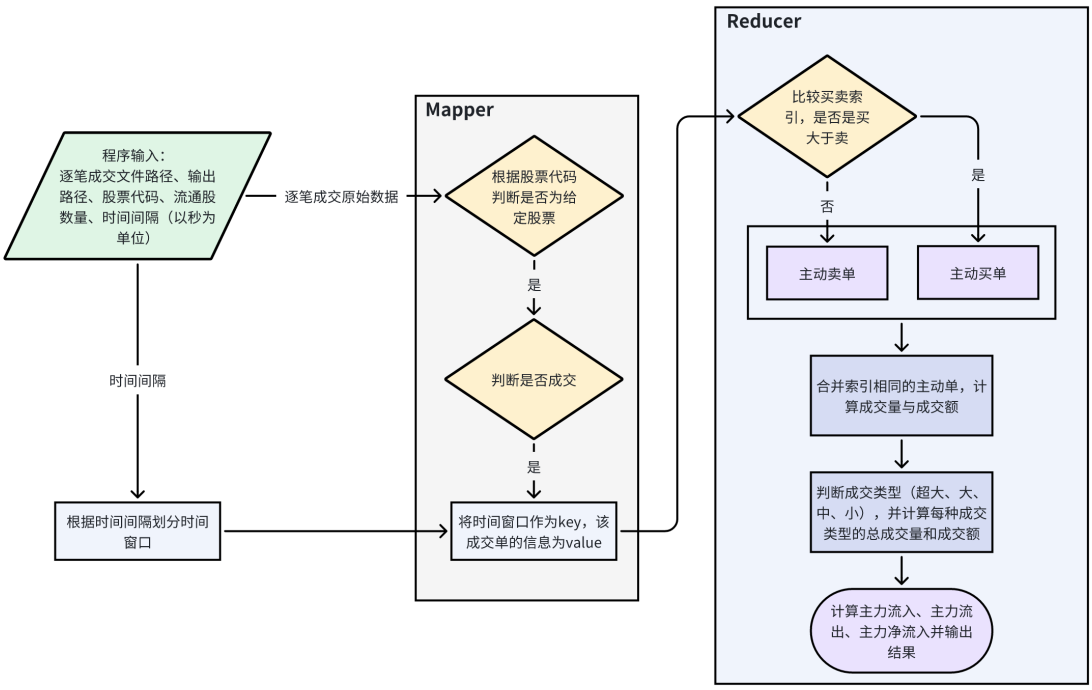

图 1. 总体设计思路

项目的总体设计思路如图1所示，整体实现采用**MapReduce框架**进行分布式计算，主要步骤包括：

1. 时间窗口处理。

2. Mapper 阶段：过滤符合条件的交易记录，并分配到对应的时间窗口，以便后续聚合处理。

3. Reducer 阶段：对每个时间窗口内的交易数据进行聚合分析，最终生成交易统计结果。

通过以上步骤，实现对每个时间窗口内股票主力资金的计算与统计。

2.2 时间窗口处理

2.2.1 时间窗口的动态生成

我们创建了一个通用的时间窗口生成工具类 TimeWindowUtils，支持根据指定的时间间隔动态生成不同时段的时间窗口。

- **输入参数**：用户可通过输入时间间隔（如秒为单位的字符串）定义窗口大小。
- **处理时段**：交易时间分为上午时段（09:30:00.000 - 11:30:00.000）和下午时段（13:00:00.000 - 15:00:00.000）。代码中对每个时段分别生成时间窗口。
- **时间窗口划分逻辑**：通过遍历的方式，将每个时间段按间隔划分为多个时间窗口，窗口结束时间严格小于等于指定的时段结束时间。
- **特殊时间窗口的处理：**在集合竞价阶段，所有交易信息都会集中在最后时刻写入，因此，为了确保上午和下午最后一个时间窗口的交易记录不被遗漏，这些时间窗口的区间需要设置为左闭右闭（包括结束时间）。为此，在Mapper中对特殊时间窗口进行了调整：**针对上午时段的结束时间 20190102113000000，将其修正为 20190102113000001；针对下午时段的结束时间 20190102150000000，将其修正为 20190102150000001**。通过这种方式，确保了边界时间点的数据也能准确匹配到对应的时间窗口。

图2展示了时间窗口生成的核心代码。

```
private static void generateTimeWindowsForPeriod(SimpleDateFormat sdf, String start,
String end, int interval, List<String> timeWindows) throws Exception{
    Date startTime = sdf.parse(start); //将字符串转换为时间
    Date endTime = sdf.parse(end); //将字符串转换为时间

    while(startTime.before(endTime)){ //严格小于
        Date nextTime = new Date(startTime.getTime() + interval); //计算下一个时间
        if(nextTime.after(endTime)){ //严格大于
            timeWindows.add(sdf.format(startTime)+"-"+sdf.format(endTime)); //将时间窗口添加
到列表
            break;
        }
        timeWindows.add(sdf.format(startTime)+"-"+sdf.format(nextTime)); //将时间窗口添加到列
表
        startTime = nextTime; //更新开始时间
    }
}
```

图 2. 时间窗口生成

2.3 Mapper 阶段：主动成交的判断与输出

在Mapper阶段，任务的主要功能是预处理逐笔成交数据，将其按时间窗口进行划分并标记，输出为 (时间窗口, 原始记录) 的键值对，便于Reducer阶段进行进一步聚合。具体逻辑如下：

2.3.1 时间窗口的读取与解析

从配置中读取securityID（目标股票代码）和时间窗口列表。时间窗口列表由程序根据输入的时间间隔动态生成，并在运行时传入Mapper进行处理。

2.3.2 数据过滤

解析输入记录，检查其是否匹配目标股票代码，以及是否为有效成交记录（执行类型为"F"）。如果股票代码不匹配或记录类型不符合要求，直接过滤掉。

2.3.3 时间窗口分类

将每条记录的交易时间与时间窗口范围进行匹配，找到其所属的时间窗口。例如，对于输入的记录，如果其时间戳在窗口 [20220101093000, 20220101094500) 之间，则输出键值对：`context.write(new Text(window), value);`

2.4 Reducer 阶段：数据聚合与指标计算

Reducer阶段的主要任务是对Mapper阶段的输出数据进行聚合，计算各时间窗口内目标股票的主力流入、主力流出、不同成交量级（如超大单、大单、中单、小单）的买卖数据，并生成最终的统计结果。逻辑如下：

2.4.1 数据解析与分类

遍历每条输入的交易记录，根据买卖索引的大小判断交易的主动方是买单还是卖单。如果买方索引大于卖方索引则判断为主动买单，否则为主动卖单。

2.4.2 成交量级标记

根据每笔交易的成交量和流通股本的比例，对交易量级进行标记（超大、大、中、小单）。

2.4.3 数据聚合

根据成交类型（买或卖）和量级标记，对各量级的成交量、成交金额进行累计。同时，主力买单金额计入mainInflow，主力卖单金额计入mainOutflow。具体的判断逻辑如图3所示。

```
private double mainInflow = 0; // 主力流入
private double mainOutflow = 0; // 主力流出
private long ultraBuyQty = 0, ultraSellQty = 0; // 超大单买入/卖出数量
private double ultraBuyAmount = 0, ultraSellAmount = 0; // 超大单买入/卖出金额
private long largeBuyQty = 0, largeSellQty = 0; // 大单买入/卖出数量
private double largeBuyAmount = 0, largeSellAmount = 0; // 大单买入/卖出金额
private long midBuyQty = 0, midSellQty = 0; // 中单买入/卖出数量
private double midBuyAmount = 0, midSellAmount = 0; // 中单买入/卖出金额
private long smallBuyQty = 0, smallSellQty = 0; // 小单买入/卖出数量
private double smallBuyAmount = 0, smallSellAmount = 0; // 小单买入/卖出金额

if (ad.type == 1) { // 买单
    if (label.equals("超大")) {
        ultraBuyQty += ad.totalQty; // 累加超大单买入数量
        ultraBuyAmount += ad.totalAmount; // 累加超大单买入金额
        mainInflow += ad.totalAmount; // 累加主力流入
    } else if (label.equals("大")) {
        largeBuyQty += ad.totalQty; // 累加大单买入数量
        largeBuyAmount += ad.totalAmount;  // 累加大单买入金额
        mainInflow += ad.totalAmount; // 累加主力流入
    } else if (label.equals("中")) {
        midBuyQty += ad.totalQty; // 累加中单买入数量
        midBuyAmount += ad.totalAmount; // 累加中单买入金额
    } else {
        smallBuyQty += ad.totalQty; // 累加小单买入数量
        smallBuyAmount += ad.totalAmount; // 累加小单买入金额
    }
```

图 3. 数据聚合逻辑

2.4.4 结果输出

计算主力净流入（mainInflow - mainOutflow）等指标，并将结果以CSV格式输出。

2.5 优化策略

2.5.1 Job数量优化

在分布式计算中，作业（Job）的数量对任务的整体性能和资源消耗有直接影响。每个Job都会引入额外的开销，包括任务初始化、数据传输、任务调度、容错管理等。因此，减少Job的数量是提升MapReduce任务效率的重要优化手段之一。

在本任务的设计中，为了实现高效的计算，**我们采用了单Job的方式完成整个数据处理流程**，将数据的预处理、分组计算和最终结果聚合整合到一个MapReduce作业中。这种设计最大限度地减少了作业切换和中间数据存储的开销，同时简化了任务的设计与管理。

2.5.2 字段解析优化

在本项目的Mapper代码中，为了提高数据处理的效率，我们对传统的字段解析方式进行了优化。常见的字段解析方法是使用 String.split()，但这种方法在处理大规模文本数据时会产生较高的性能开销。这是因为 String.split() 在执行时需要解析正则表达式，并为每个字段生成新的字符串对象，增加了CPU和内存的使用。在高性能计算任务中，这种开销可能会成为瓶颈。

为此，**我们采用了基于索引的字段解析方法，通过手动查找分隔符位置直接提取子字符串，避免了正则表达式解析的性能开销**。当输入数据格式规整且字段数固定时，这种方法不仅减少了内存分配和垃圾回收的压力，还提高了字段解析的速度。

在代码中，我们预先定义了字段数量（fieldCount）并使用一个预分配的数组来存储字段值。通过遍历字符串寻找制表符（\t）的位置，逐步提取每个字段。具体的代码实现如图4所示。

```
String line = value.toString();
int fieldCount = 16; // 预计字段数量
String[] fields = new String[fieldCount]; // 用于存储输入字段的预分配数组
int startAt = 0, fieldIndex = 0; // 初始化开始位置和字段索引

for (int i = 0; i < line.length() && fieldIndex < fieldCount; i++) {
    if (line.charAt(i) == '\t') {
        fields[fieldIndex++] = line.substring(startAt, i);
        startAt = i + 1;
    }
} // 按制表符分割字段
if (fieldIndex < fieldCount) {
    fields[fieldIndex++] = line.substring(startAt); // 最后一列
}
```

图 4. 字段解析的代码实现

3 实验结果

我们对股票代码从000001到000005的股票数据进行了程序运行时间的测试，结果如表格1所示。可以看出，对于每只股票，从原始数据到计算主力流入、主力流出以及主力净流入的全过程，平均耗时约为51秒，说明程序在大规模数据处理下具有较高的效率。

表格 1. 速度测试结果

股票代码	用时（秒）
000001	51
000002	51
000003	50
000004	51
000005	51

此外，我们使用图形组件对000001的数据进行了深入分析和可视化展示，结果如图5和图6所示。

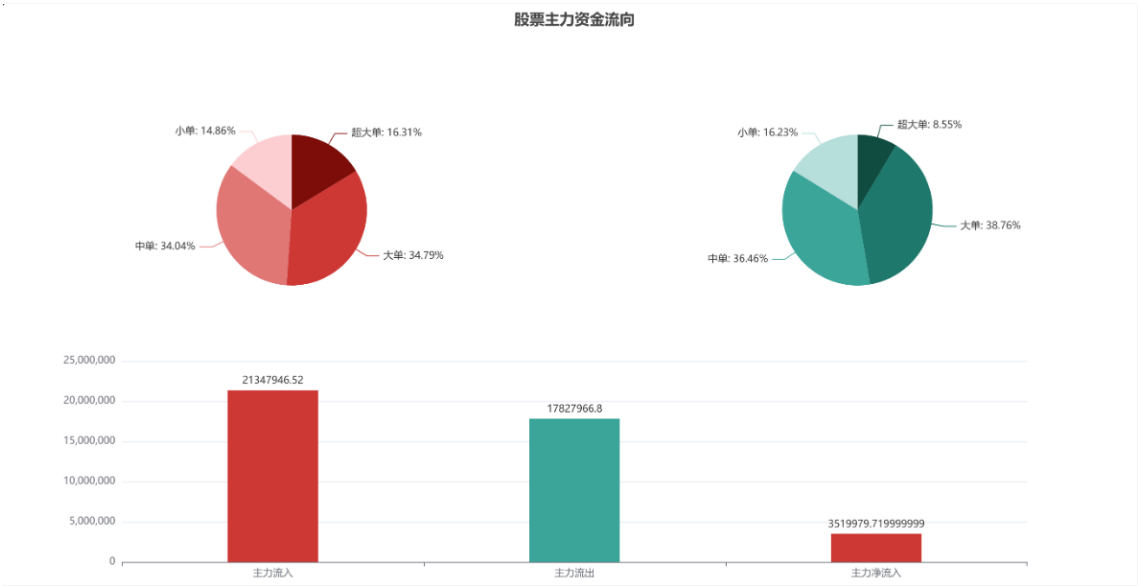

图 5. 股票主力资金流向

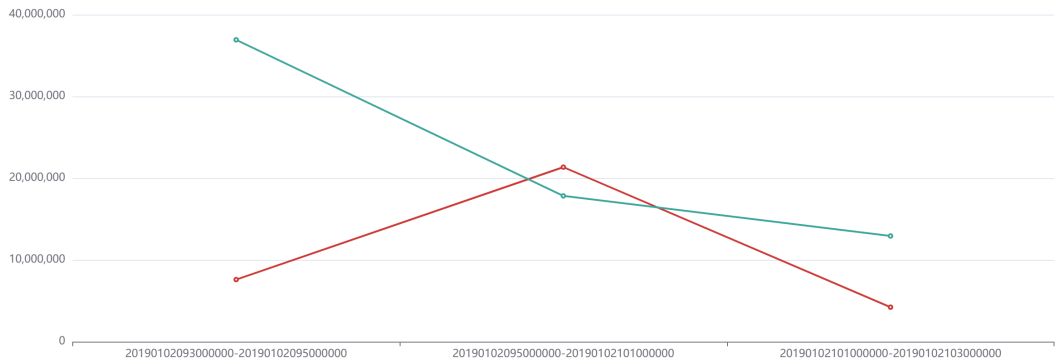

图 6. 主力流入与主力流出岁时间的变化关系