Security Matching System Speech

Shen Guanzhi

2024 年 11 月 24 日

第一章　问题分析

1.1　市场周期

证券交易所的一个交易日内会呈现 4 种不同的状态，这里简单地将其称作睡眠状态、准备状态、集合竞价状态和连续竞价状态。其中，从集合竞价状态切换到睡眠状态时需要计算出开盘或收盘交易。图中做了一定的简化，忽略了连续竞价中场休息等状态切换过程。

1.2　单次撮合过程

集合竞价和连续竞价的基础都是单次撮合算法——即比较并修改最优买单和最优卖单。

1.3　集合竞价与连续竞价对比

尽管都是对于单次撮合算法的循环，集合竞价算法与连续竞价算法却需要完全不同的实现。

计算开盘价与收盘价的任务是“不可抢占的”。这里的“不可抢占”不是说它们对于操作系统来说是原子操作，而是强调它们内部独占所需的数据资源，不必考虑任何任务间数据竞争与同步的问题。由于参与集合竞价的订单可能很多，计算结盘价的任务应该视为 CPU 密集型的任务。为了公平性，不同证券的结盘价计算任务需要尽量并发进行。本系统中直接采用了第三方库来进行此调度工作。

连续竞价任务则相反。过程中用户随时可以有查询行情或撤销订单的操作。如果竞价任务长时间占据数据资源，会使得查询或撤单任务陷入饥饿，违反了公平性原则。

第二章　程序设计

2.1　买卖盘

每支证券对应一个买卖盘。一个买卖盘包含一个买单队列和一个卖单队列。

系统的核心任务是“撮合”，因此订单队列需要是优先队列。优先队列的实现方式可以是二叉堆——就是用完全二叉树实现的那种数据结构；也可以是平衡搜索树——包括 AVL 树、红黑树这些。

比较可知，二叉堆的优势在于查询但不删除最优元素的效率更高。但是，由于买单和卖单在统计学上总体是平衡的，这种“只查询但不删除”的操作并没有那么频繁。

平衡树还有其他的优势——比如能够直接获取排好序的即时行情。

尽管最终版本没能实现，本系统设计之初曾试图考虑过“订单时刻完全相同”的问题。在这种情况下，平衡树将是唯一正确的选择。因此，由于更加优秀的综合性能，平衡树是比二叉堆更为合适的订单队列数据结构。

2.2　撮合器

2.2.1　组成

单独一个买卖盘是无法撮合订单的，需要一些其他数据结构辅助。等待队列用于缓存那些已生成订单号但尚未进入买卖盘的订单——接收订单并生成订单号是 $O(1)$ 的，但插入买卖盘并竞价则可能遇到先前订单占用的问题，需要等待更多时间。“竞价”任务是一个单独的协程，用于完成等待中订单的入盘。Watcher 之后再介绍。

2.2.2　难点

1 个撮合器会受到 5 种不同协程的读写，各自之间有着较为复杂的竞争与同步关系。这些关系是整个撮合算法的核心与难点。“接收”任务需要唤醒“竞价”任务并挂起“计算结盘价”任务；“竞价”任务需要在清空等待队列后唤醒“计算”任务并挂起自身，“撤销”任务与“获取行情”任务都需要与其他任务竞争买卖盘的读写权。

2.2.3　同步方案

整个撮合算法力求兼顾正确性与公平性。买卖盘和等待队列使用读写锁保护。Watcher 是一个特殊管道，其中有且只有一个布尔值。

下面这个图是为了示意撮合过程中状态变化的复杂性的，不需要细看。编程实现其实并不复杂——“竞价”任务等待 Watcher 值为 true 时试图清空等待队列，队列空后将 Watcher 置 false；“计算”任务等待 Watcher 值为 false 时试图锁定买卖盘并结算。

第三章　数据库设计

A/C 库存储账户数据。Req(uest) 库记录买单、卖单、撤单请求，其主键兼作订单号，随时间递增。Order 库用于备份买卖盘，以防宕机。Rec(ord) 库记录成交记录。Msg 库用于备份未发出的用户通知。

数据库选用了 PostgreSQL ，程序中利用了它的事务机制以保证了交易的一致性。