## **vn.py社区精选12 - 策略参数优化，你需要懂得压榨CPU！**

原创 张国平 VNPY  2019-11-12

收录于话题

#vn.py社区精选

25个

**万恶的全局锁**

基于物理上的限制，各CPU厂商在核心频率上的比赛已经被多核所取代。为了更有效的利用多核处理器的性能，多线程的编程方式被越来越多地应用到了各类程序中，而随之带来的则是线程间数据一致性和状态同步的困难。

作为已经30岁的Python，自然早已支持多线程的功能，但坊间却始终存在着一种误解：Python的多线程是假的（或者虚拟机模拟的）。

Python虚拟机（或者叫解释器）使用了引用计数法的垃圾回收方式（另一种是以Java虚拟机为例的根搜索算法），这种垃圾回收方式使得Python虚拟机在运行多线程程序时，需要使用一把名为GIL（Global Interpreter Lock，全局解释器锁）的超级大锁，来保证每个对象上的引用计数正确。

从操作系统来看，每个CPU在同一时间都能够且只能执行一个线程。而在Python虚拟机上，任何一个线程的运行，都需要包含以下三个步骤：

获取GIL；

执行代码，直到sleep，或者被Python虚拟机挂起；

释放GIL；

因此，某个线程想要执行，必须先拿到GIL，我们可以把GIL看作是“通行证”，并且在一个Python进程中GIL也只有一个。所以哪怕硬件上CPU有再多的核心，任意时刻都只能有一个线程能拿到GIL来执行，这也是之前提到的误解来源。

Python多线程的痛点在于每次释放GIL锁，线程进行锁竞争、切换线程，会消耗资源。这导致很多时候，尤其是计算密集型任务为主的程序，多核多线程比单核多线程更差：

单核下多线程，每次释放GIL，唤醒的那个线程都能获取到GIL锁，所以能够无缝执行；

多核下，CPU0释放GIL后，其他CPU上的线程都会进行竞争，但GIL可能会马上又被CPU0拿到，导致其他几个CPU上被唤醒后的线程会醒着等待到切换时间后又进入待调度状态，这样会造成线程颠簸(thrashing)，导致效率更低。

因此，在Python中想要充分压榨多核CPU的性能，必须依赖多进程的模式。每个进程有各自独立的GIL，互不干扰，这样就可以真正意义上的并行执行。

**方便的多进程**

Python语言中内置了专门用于实现多进程的multiprocessing库，使用上相当傻瓜，通过multiprocessing.Process类来创建一个新的子进程对象，再启动这个对象，这样一个多进程任务就开始执行了。

等CPU分配一个独立核心去干活，func函数就在这个子进程中开始执行了，这里唯一要注意args是默认输入元组参数。

p = multiprocessing.Process(target=func, args=(a,))p.start()

除了一个一个的启动子进程外，也可以使用multiprocessing.Pool来创建进程池对象，把需要干的工作任务打包好，放在这个池子里面，这样一个任务执行完CPU核心空闲下来后，就能自动从进程池中去获取一个新的任务继续干活。

基本的使用步骤如下：

设置进程池中的进程数量，通常将其设置为小于或者等于cpu核心数量，避免多余的进程无法同时执行还要占用额外的内存；

然后使用pool.apply\_async方法，把打包好的任务插入池中；

调用pool.close把这个进程池关闭，不再接受新的任务；

若还有一些已有任务在跑，使用pool.join()函数，阻塞当前的主线程，直到进程池中的所有任务都执行完成才进入下一步。

**多进程参数优化**

学习多进程模块怎么用，最好的例子之一就是vn.py的CTA策略回测引擎中的参数优化功能，加载同样的历史数据基于不同的参数，执行历史数据回放和策略盈亏统计，属于典型的多进程应用场景。

多进程优化函数位于：

vnpy.app.cta\_strategy.backtesting.BacktestingEngine.run\_optimization

该函数中的执行步骤如下：

传入全局优化列表settings，传入的参数越多，所形成的全局组合越多；

传入优化目标，常见的有夏普比率，收益回撤比；

根据主机的CPU核数创建对应数量的进程池pool；

在for循环中从全局优化列表settings获取元素回测参数setting，和策略类，策略参数打包为任务内容，和任务方法optimize一起组合为一个工作任务，最后插入到进程池给CPU核心去跑；

每次优化结果为result字典，并且把回测结果放在results列表中；

基于优化目标，如夏普比率对results列表进行排序。

def run\_optimization(self, optimization\_setting: OptimizationSetting, output=True): """""" # Get optimization setting and target settings = optimization\_setting.generate\_setting() target\_name = optimization\_setting.target\_name  
 if not settings: self.output("优化参数组合为空，请检查") return  
 if not target\_name: self.output("优化目标未设置，请检查") return  
 # Use multiprocessing pool for running backtesting with different setting pool = multiprocessing.Pool(multiprocessing.cpu\_count())  
 results = [] for setting in settings: result = (pool.apply\_async(optimize, ( target\_name, self.strategy\_class, setting, self.vt\_symbol, self.interval, self.start, self.rate, self.slippage, self.size, self.pricetick, self.capital, self.end, self.mode ))) results.append(result)  
 pool.close() pool.join()  
 # Sort results and output result\_values = [result.get() for result in results] result\_values.sort(reverse=True, key=lambda result: result[1])  
 if output: for value in result\_values: msg = f"参数：{value[0]}, 目标：{value[1]}" self.output(msg)  
 return result\_values

启动多进程优化的任务后，打开Windows的任务管理器，可以看到此时CPU所有的8个核心都已经在满载运行了。

