在有详细约束的情况下，允许可写页面的多个副本存在是有可能的。一种方法是允许一个进程获得在部分虚拟地址空间上的一把锁，然后在被锁住的存储空间中进行多个读写操作。在该锁被释放时，产生的修改可以传播到其他副本上去。只要在一个给定的时刻只有一个CPU能锁住某个页面，这样的机制就能保持一致性。

另一种方法是，当一个潜在可写的页面被第一次真正写入时，制作一个“干净”的副本并保存在发出写操作的CPU上。然后可在该页上加锁，更新页面，并释放锁。稍后，当一个远程机器上的进程试图获得该页面上的锁时，先前进行写操作的CPU将该页面的当前状态与“干净”副本进行比较并构造一个有关所有已修改的字的列表，该列表接着被送往获得锁的CPU，这样它就可以更新其副本页面而不用废弃它（Keleher等人，1994）。

8.2.6 多计算机调度

在一台多处理机中，所有的进程都在同一个存储器中。当某个CPU完成其当前任务后，它选择一个进程并运行。理论上，所有的进程都是潜在的候选者。而在一台多计算机中，情形就大不相同了。每个节点有其自己的存储器和进程集合。CPU 1不能突然决定运行位于节点4上的一个进程，而不事先花费相当大的工作量去获得该进程。这种差别说明在多计算机上的调度较为容易，但是将进程分配到节点上的工作更为重要。下面我们将讨论这些问题。

多计算机调度与多处理机的调度有些类似，但是并不是后者的所有算法都能适用于前者。最简单的多处理机算法——维护就绪进程的一个中心链表——就不能工作，因为每个进程只能在其当前所在的CPU上运行。不过，当创建一个新进程时，存在着一个决定将其放在哪里的选择，例如，从平衡负载的考虑出发。

由于每个节点拥有自己的进程，因此可以应用任何本地调度算法。但是，仍有可能采用多处理机的群调度，因为惟一的要求是有一个初始的协议来决定哪个进程在哪个时间槽中运行，以及用于协调时间槽的起点的某种方法。

8.2.7 负载平衡

需要讨论的有关多计算机调度的内容相对较少。这是因为一旦一个进程被指定给了一个节点，就可以使用任何本地调度算法，除非正在使用群调度。不过，一旦一个进程被指定给了某个节点，就不再有什么可控制的，因此，哪个进程被指定给哪个节点的决策是很重要的。这同多处理机系统相反，在多处理机系统中所有的进程都在同一个存储器中，可以随意调度到任何CPU上运行。因此，值得考察怎样以有效的方式把进程分配到各个节点上。从事这种分配工作的算法和启发则是所谓的处理器分配算法（processor allocation algorithm）。

多年来已出现了大量的处理器（节点）分配算法。它们的差别是分别有各自的前提和目标。可知的进程属性包括CPU需求、存储器使用以及与每个其他进程的通信量等。可能的目标包括最小化由于缺少本地工作而浪费的CPU周期，最小化总的通信带宽，以及确保用户和进程公平性等。下面将讨论几个算法，以使读者了解各种可能的情况。

1. 图论确定算法

有一类被广泛研究的算法用于下面这样一个系统，该系统包含已知CPU和存储器需求的进程，以及给出每对进程之间平均流量的已知矩阵。如果进程的数量大于CPU的数量k，则必须把若干个进程分配给每个CPU。其想法是以最小的网络流量完成这个分配工作。

该系统可以用一个带权图表示，每个顶点是一个进程，而每个弧代表两个进程之间的消息流。在数学上，该问题就简化为在特定的限制条件下（如每个子图对整个CPU和存储器的需求低于某些限制），寻找一个将图分割（切割）为k个互不连接的子图的方法。对于每个满足限制条件的解决方案，完全在单个子图内的弧代表了机器内部的通信，可以忽略。从一个子图通向另一个子图的弧代表网络通信。目标是找出可以使网络流量最小同时满足所有的限制条件的分割方法。作为一个例子，图8-24给出了一个有9个进程的系统，这9个进程是进程A至I，每个弧上标有两个进程之间的平均通信负载（例如，以Mbps为单位）。

在图8-24a中，我们将有进程A、E和G的图划分到节点1上，进程B、F和H划分在节点 2上，而进程C、D和I划分在节点3上。整个网络流量是被切割（虚线）的弧上的流量之和，即30个单位。在图8-24b