中，有一种不同的划分方法，只有28个单位的网络流量。假设该方法满足所有的存储器和CPU的限制条件，那么这个方法就是一个更好的选择，因为它需要较少的通信流量。

直观地看，我们所做的是寻找紧耦合（簇内高流量）的簇（cluster），并且与其他的簇有较少的交互（簇外低流量）。讨论这些问题的最早的论文是（Chow和Abraham，1982；Lo，1984；Stone和Bokhari，1978）等。

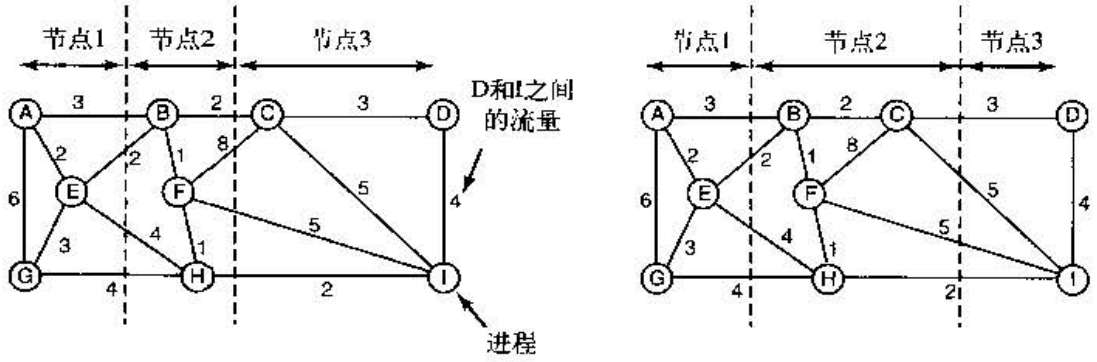

图8-24 将9个进程分配到3个节点上的两种方法

2. 发送者发起的分布式启发算法

现在看一些分布式算法。有一个算法是这样的，当进程创建时，它就运行在创建它的节点上，除非该节点过载了。过载节点的度量可能涉及太多的进程，过大的工作集，或者其他度量。如果过载了，该节点随机选择另一个节点并询问它的负载情况（使用同样的度量）。如果被探查的节点负载低于某个阈值，就将新的进程送到该节点上（Eager等人，1986）。如果不是，则选择另一个机器探查。探查工作并不会永远进行下去。在N次探查之内，如果没有找到合适的主机，算法就终止，且进程继续在原有的机器上运行。整个算法的思想是负载较重的节点试图甩掉超额的工作，如图8-25a所示。该图描述了发送者发起的负载平衡。

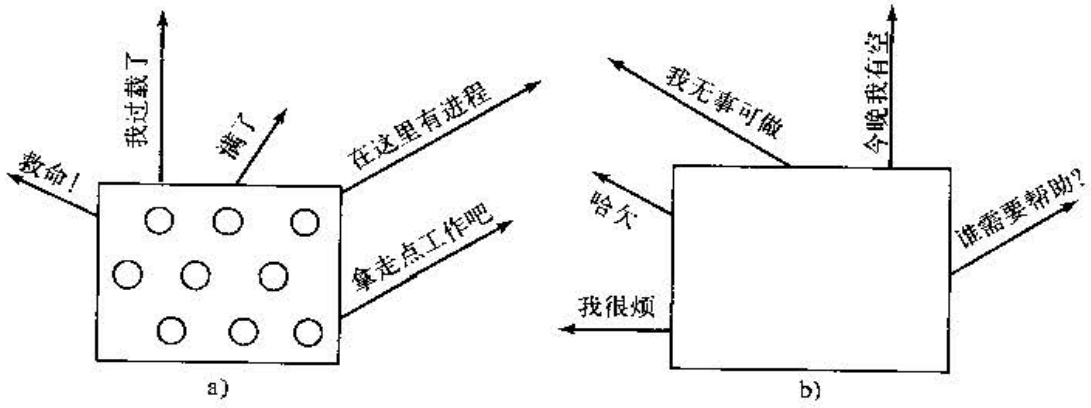

图8-25 a) 过载的节点寻找可以接收进程的轻载节点；b) 一个空节点寻找工作做

Eager等人（1986）构造了一个该算法的分析排队模型（queueing model）。使用这个模型，所建立的算法表现良好而且在包括不同的阈值、传输成本以及探查限定等大范围的参数内工作稳定。

但是，应该看到在负载重的条件下，所有的机器都会持续地对其他机器进行探查，徒劳地试图找到一台愿意接收更多工作的机器。几乎没有进程能够被卸载，可是这样的尝试会带来巨大的开销。

3. 接收者发起的分布式启发算法

上面所给出的算法是由一个过载的发送者发起的，它的一个互补算法是由一个轻载的接收者发起的，如图8-25b所示。在这个算法中，只要有一个进程结束，系统就检查是否有足够的工作可做。如果不是，它随机选择某台机器并要求它提供工作。如果该台机器没有可提供的工作，会接着询问第二台，然后是第三台机器。如果在N次探查之后，还是没有找到工作，该节点暂时停止询问，去做任何已经安排好的