State-change-driven. In this class of policy, nodes disseminate state information whenever their state changes by a certain degree [24]. A state-change-driven policy differs from a demand-driven policy in that it disseminates information about the state of a node, rather than collecting information about other nodes. Under centralized state-change-driven policies, nodes send state information to a centralized collection point. Under decentralized state-change-driven policies, nodes send information to peers [32].

11.5 STABILITY

We now describe two views of stability.

11.5.1 The Queuing-Theoretic Perspective

When the long term arrival rate of work to a system is greater than the rate at which the system can perform work, the CPU queues grow without bound. Such a system is termed unstable. For example, consider a load distributing algorithm performing excessive message exchanges to collect state information. The sum of the load due to the external work arriving and the load due to the overhead imposed by the algorithm can become higher than the service capacity of the system, causing system instability.

Alternatively, an algorithm can be stable but may still cause a system to perform worse than when it is not using the algorithm. Hence, a more restrictive criterion for evaluating algorithms is desirable, and we use the *effectiveness* of an algorithm as the evaluating criterion. A load distributing algorithm is said to be effective under a given set of conditions if it improves the performance relative to that of a system not using load distributing. Note that while an effective algorithm cannot be unstable, a stable algorithm can be ineffective.

11.5.2 The Algorithmic Perspective

If an algorithm can perform fruitless actions indefinitely with finite probability, the algorithm is said to be unstable [3]. For example, consider *processor thrashing*. The transfer of a task to a receiver may increase the receiver's queue length to the point of overload, necessitating the transfer of that task to yet another node. This process may repeat indefinitely [3]. In this case, a task is moved from one node to another in search of a lightly loaded node without ever receiving service. Discussions on various types of algorithmic instability are beyond the scope of this book and can be found in [6].

11.6 LOAD DISTRIBUTING ALGORITHMS

We now describe some load distributing algorithms that have appeared in the literature and discuss their performance.

11.6.1 Sender-Initiated Algorithms

In sender-initiated algorithms, load distributing activity is initiated by an overloaded node (sender) that attempts to send a task to an underloaded node (receiver). This section covers three simple yet effective sender-initiated algorithms studied by Eager, Lazowska, and Zohorjan [11].

Transfer policy. All three algorithms use the same transfer policy, a threshold policy based on CPU queue length. A node is identified as a sender if a new task originating at the node makes the queue length exceed a threshold T. A node identifies itself as a suitable receiver for a remote task if accepting the task will not cause the node's queue length to exceed T.

Selection policy. These sender-initiated algorithms consider only newly arrived tasks for transfer.

Location policy. These algorithms differ only in their location policy:

Random. Random is a simple dynamic location policy that uses no remote state information. A task is simply transferred to a node selected at random, with no information exchange between the nodes to aid in decision making. A problem with this approach is that useless task transfers can occur when a task is transferred to a node that is already heavily loaded (i.e., its queue length is above the threshold). An issue raised with this policy concerns the question of how a node should treat a transferred task. If it is treated as a new arrival, the transferred task can again be transferred to another node if the local queue length is above the threshold. Eager et al. [11] have shown that if such is the case, then irrespective of the average load of the system, the system will eventually enter a state in which the nodes are spending all their time transferring tasks and not executing them. A simple solution to this problem is to limit the number of times a task can be transferred. A sender-initiated algorithm using the random location policy provides substantial performance improvement over no load sharing at all [11].

Threshold. The problem of useless task transfers under random location policy can be avoided by polling a node (selected at random) to determine whether it is a receiver (see Fig. 11.3). If so, the task is transferred to the selected node, which must execute the task regardless of its state when the task actually arrives. Otherwise, another node is selected at random and polled. The number of polls is limited by a parameter called *PollLimit* to keep the overhead low. Note that while nodes are randomly selected, a sender node will not poll any node more than once during one searching session of PollLimit polls. If no suitable receiver node is found within the PollLimit polls, then the node at which the task originated must execute the task. By avoiding useless task transfers, the threshold policy provides substantial performance improvement over the random location policy [11].

Shortest. The two previous approaches make no effort to choose the best receiver for a task. Under the *shortest* location policy, a number of nodes (= PollLimit) are

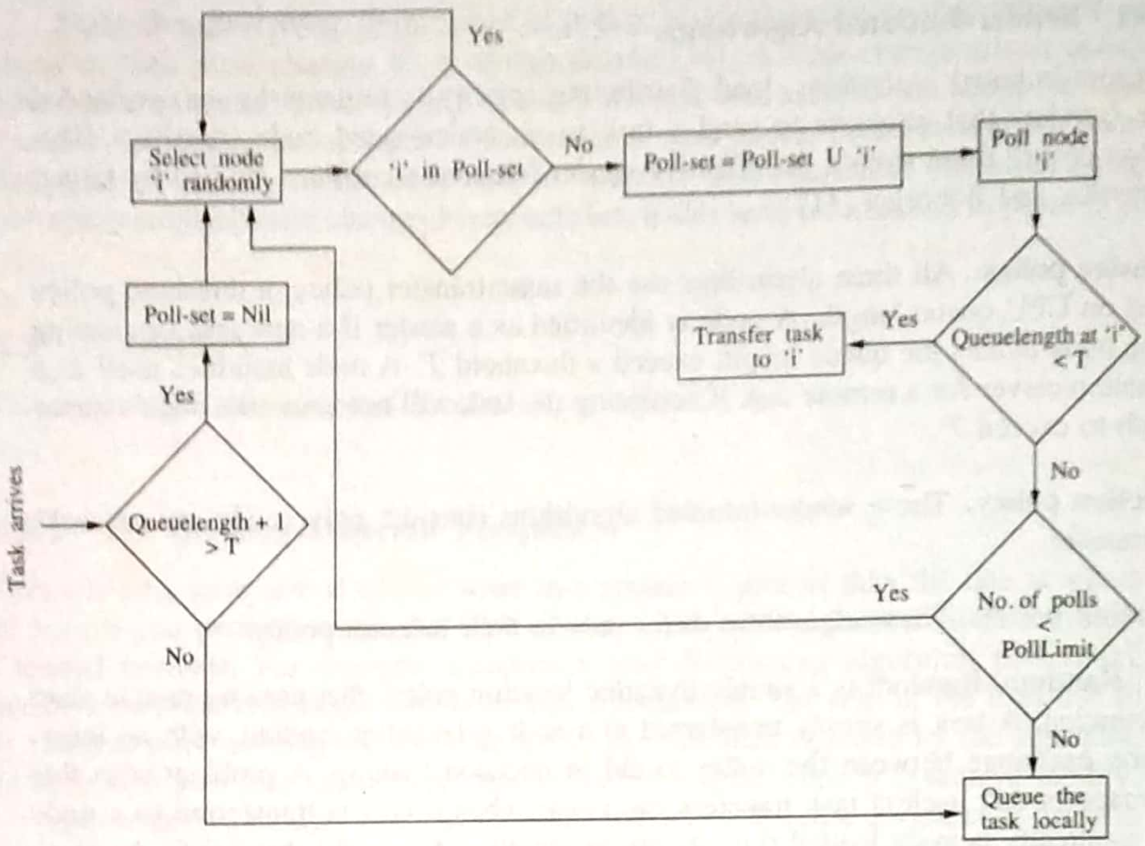

FIGURE 11.3
Sender-initiated load sharing with threshold location policy.

selected at random and are polled to determine their queue length [11]. The node with the shortest queue length is selected as the destination for task transfer unless its queue length $\geq T$. The destination node will execute the task regardless of its queue length at the time of arrival of the transferred task. The performance improvement obtained by using the shortest location policy over the threshold policy was found to be marginal [11], indicating that using more detailed state information does not necessarily result in significant improvement in system performance.

Information policy. When either the shortest or the threshold location policy is used, polling activity commences when the transfer policy identifies a node as the sender of a task. Hence, the information policy can be considered to be of the demand-driven type.

Stability. These three approaches for location policy used in sender-initiated algorithms cause system instability at high system loads, where no node is likely to be lightly loaded, and hence the probability that a sender will succeed in finding a receiver node is very low. However, the polling activity in sender-initiated algorithms increases as the rate at which work arrives in the system increases, eventually reaching a point where the cost of load sharing is greater than the benefit. At this point, most of the available CPU cycles are wasted in unsuccessful polls and in responding to these polls. When the load due to work arriving and due to the load sharing activity exceeds the system's serving capacity, instability occurs. Thus, the actions of sender-initiated algorithms are not effective at high system loads and cause system instability by failing to adapt to the system state.

11.6.2 Receiver-Initiated Algorithms

In receiver-initiated algorithms, the load distributing activity is initiated from an underloaded node (receiver) that is trying to obtain a task from an overloaded node (sender). In this section, we describe the policies of an algorithm [31] that is a variant of the algorithm proposed in [10] (see Fig. 11.4).

Transfer policy. Transfer policy is a threshold policy where the decision is based on CPU queue length. The transfer policy is triggered when a task departs. If the local queue length falls below the threshold *T*, the node is identified as a receiver for obtaining a task from a node (sender) to be determined by the location policy. A node is identified to be a sender if its queue length exceeds the threshold *T*.

Selection policy. This algorithm can make use of any of the approaches discussed under the selection policy in Sec. 11.4.2.

Location policy. In this policy, a node selected at random is polled to determine if transferring a task from it would place its queue length below the threshold level. If

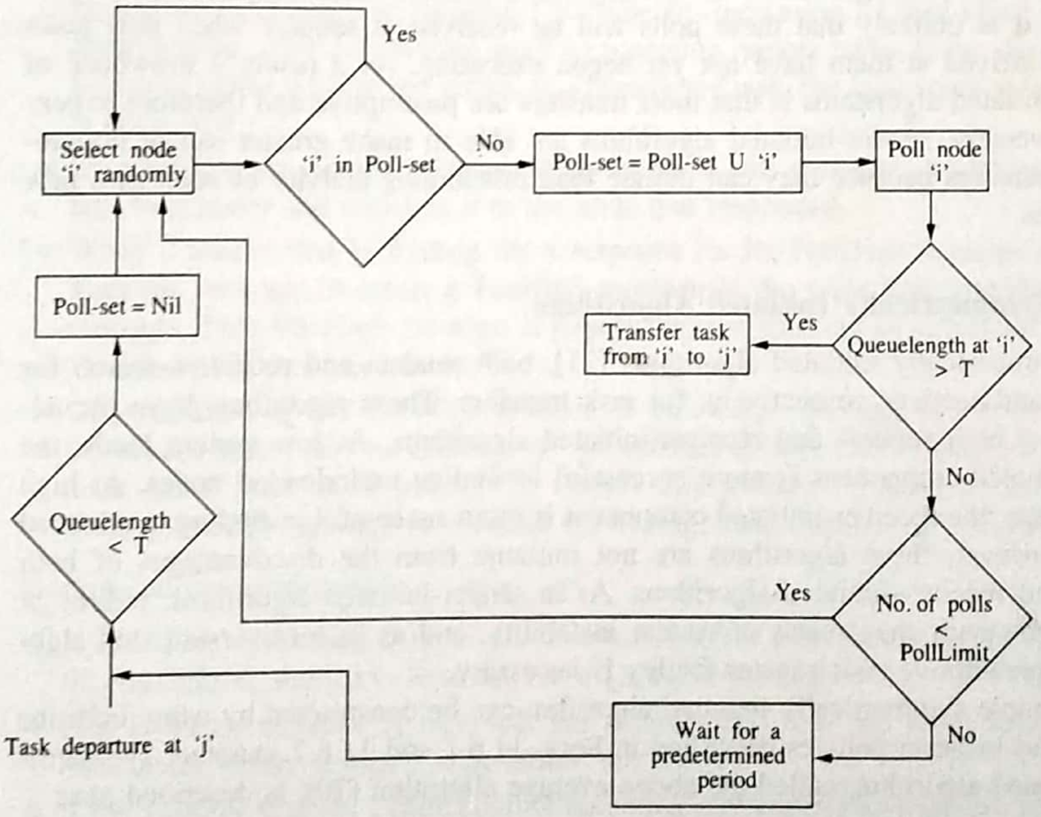

FIGURE 11.4
Receiver-initiated load sharing.