

## Distributed Scheduling

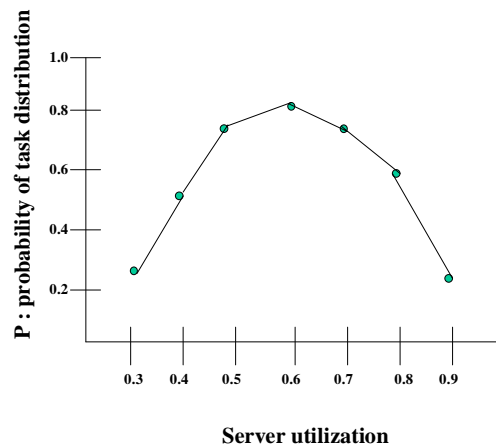
Goal: enable transparent execution of programs on networked computing systems

Motivations: reduce response time of program execution through load balancing

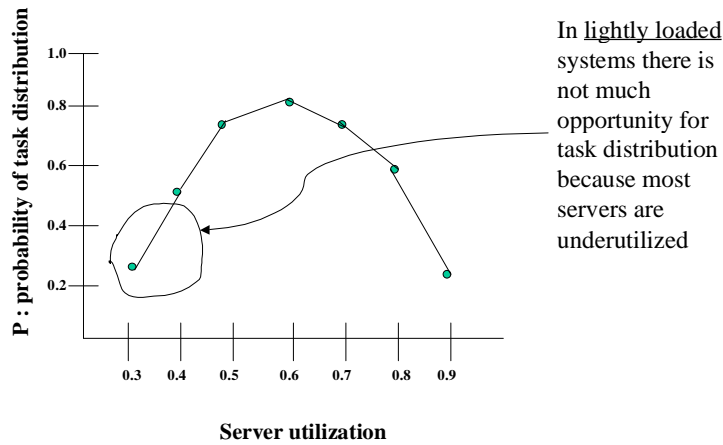
An aspect of current interest in “metacomputing” systems

- globus
- legion

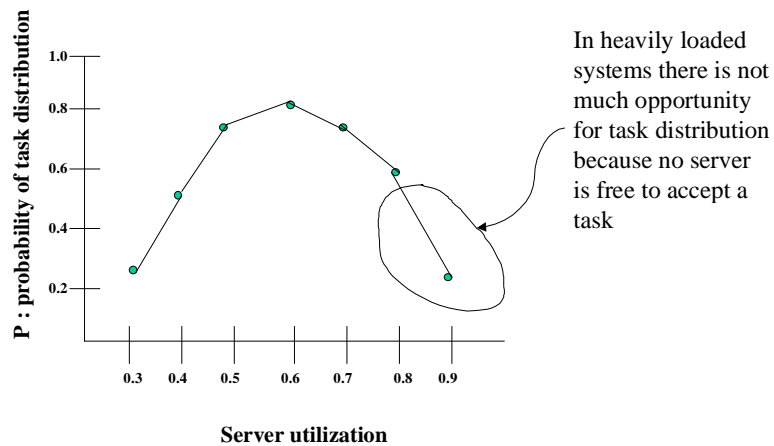
## Opportunities for Task Distribution



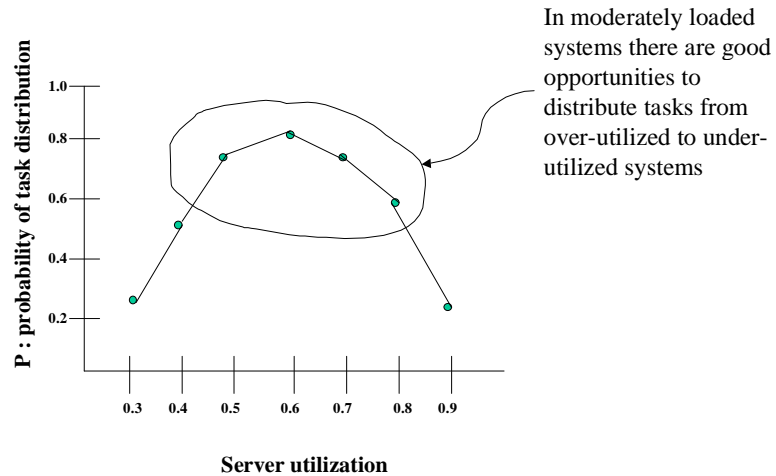
## Task Distribution



## Task Distribution



## Task Distribution



## Characteristics of Approaches

### Goals:

- load sharing (distribute load) vs.
- load balancing (equalize load)

### Information:

- static (invariant of system state)
- dynamic (uses system state)
- adaptive (changes actions with system state)

### Transfers:

- preemptive (interrupts task for transfer) vs.
- non-preemptive (transfers only new tasks)

## Component Policies

- Transfer determines whether a node is in a state to participate in load transfers and in what role
- Selection determines which local task is involved in the transfer
- Location determines a pair of nodes to participate in task transfer
- Information determines what information is collected and how
  - demand-driven (obtained when needed)
  - periodic (at regular intervals)
  - state-change-driven (obtained when nodes change state)

## Kinds of Algorithms

sender-initiated : an overloaded node searches for a underloaded node to take one of its tasks

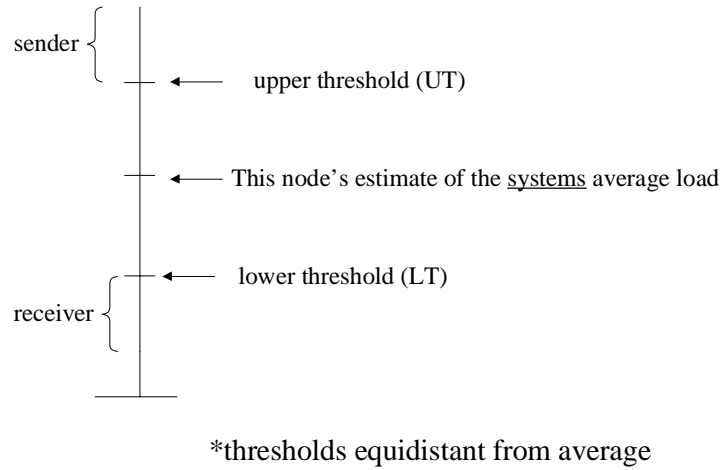
location policies: random, polling-first found, polling-least loaded  
stability: unstable/ineffective at high system loads

receiver-initiated : an underloaded node searches for a task to take from an overloaded node

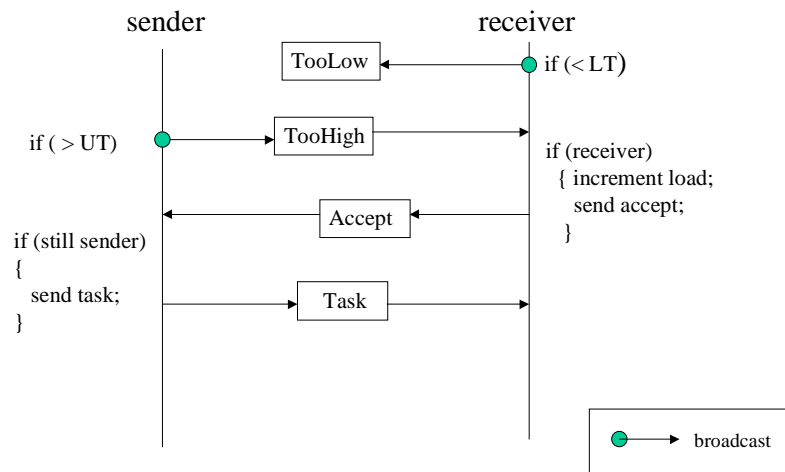
location policies: random, polling  
stability: stable at high system loads  
drawback: uses preemptive transfers in many cases

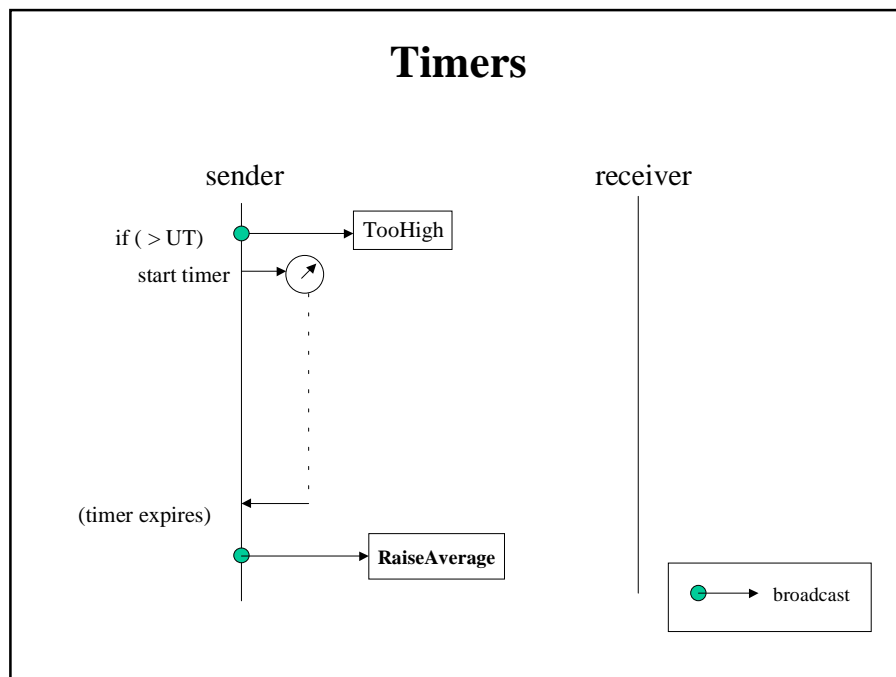
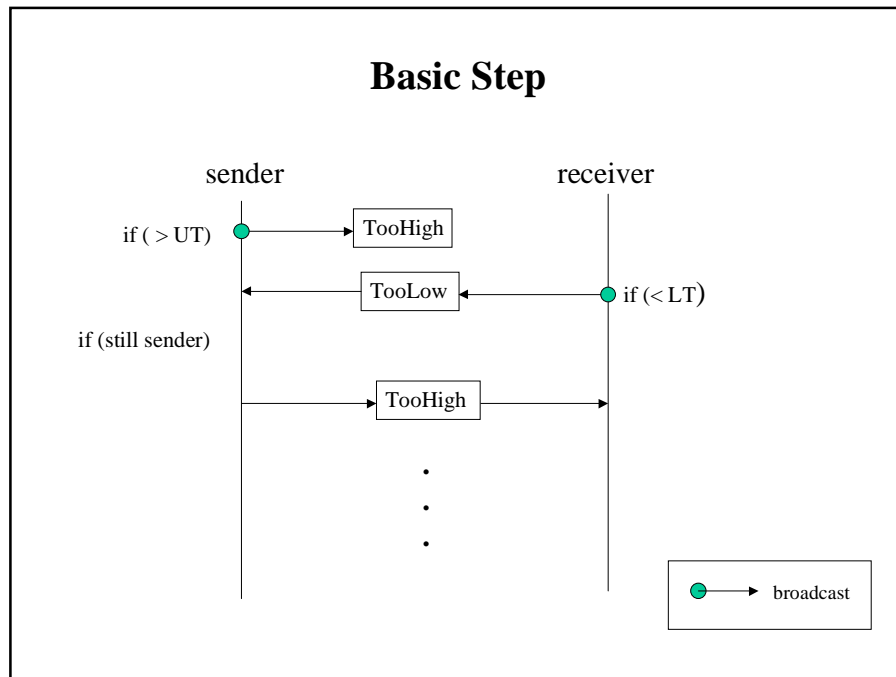
symmetrically-initiated : senders and receivers search for each other

## Above-Average Algorithm

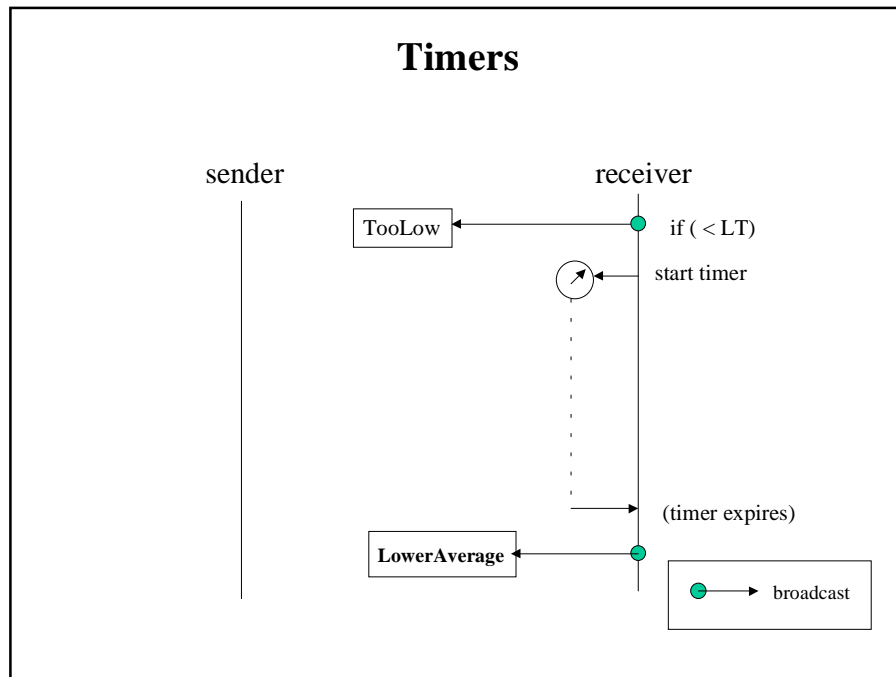


## Basic Step





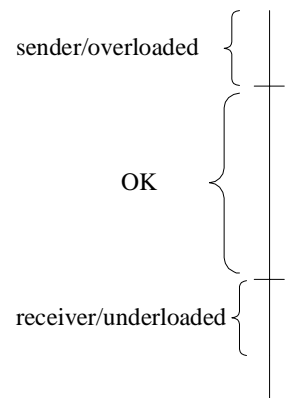
## Timers



## A Stable, Symmetrically Initiated Algorithm

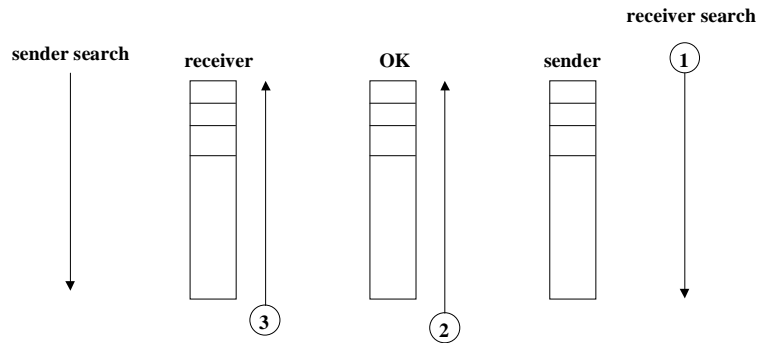
Transfer Policy:

Load is measured by  
CPU queue length

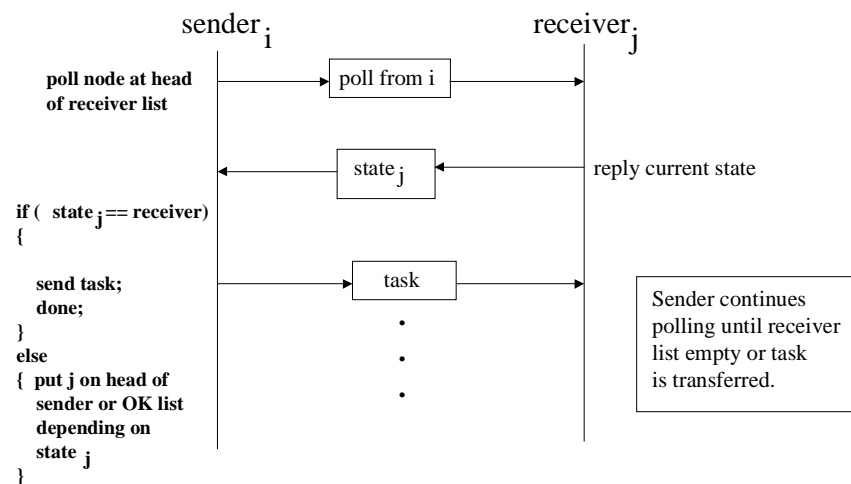


## Stable, Symmetrically Initiated Algorithm

Each node maintains three lists that are searched in the following orders:

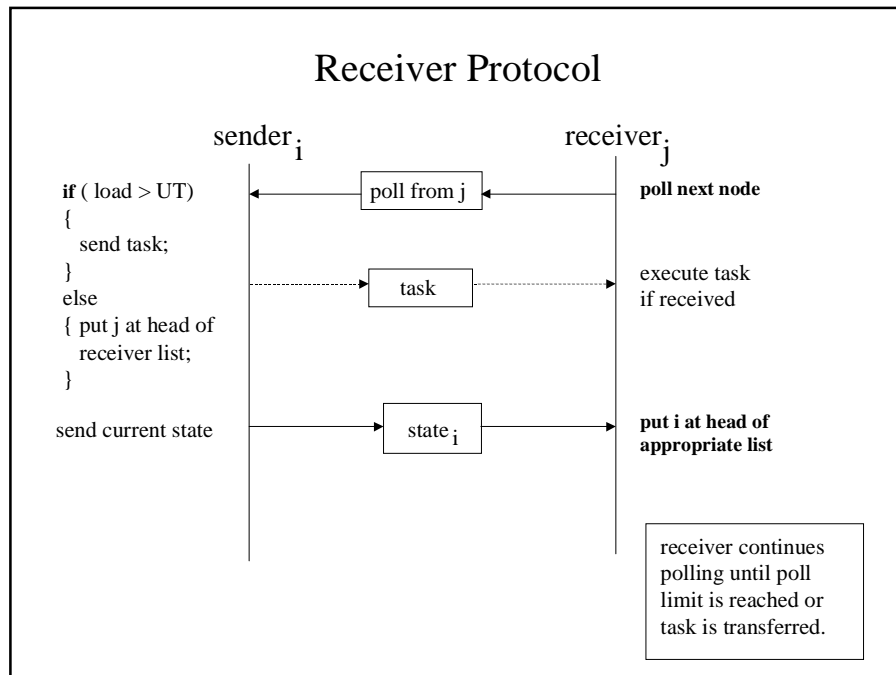


## Sender Protocol





## Receiver Protocol



## Stability

### At high loads:

- sender-initiated polling stops because receiver list becomes empty
- receiver-initiated polling has low overhead because it will quickly find a task to transfer

### At low loads:

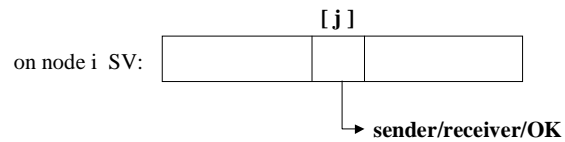
- receiver-initiated polling will usually fail but overhead is acceptable and other nodes are updated
- sender initiated polling will quickly succeed

### At intermediate loads:

- receiver-initiated and sender-initiated both work

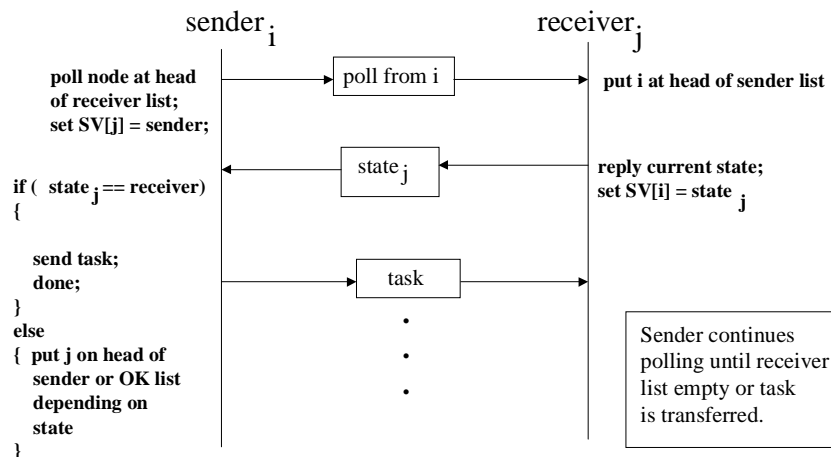
## A Stable Sender-Initiated Algorithm

Similar to previous algorithm except that it has a modified receiver protocol. Each node maintains a state vector,  $SV$ , indicating on which list the node is on at all other nodes.

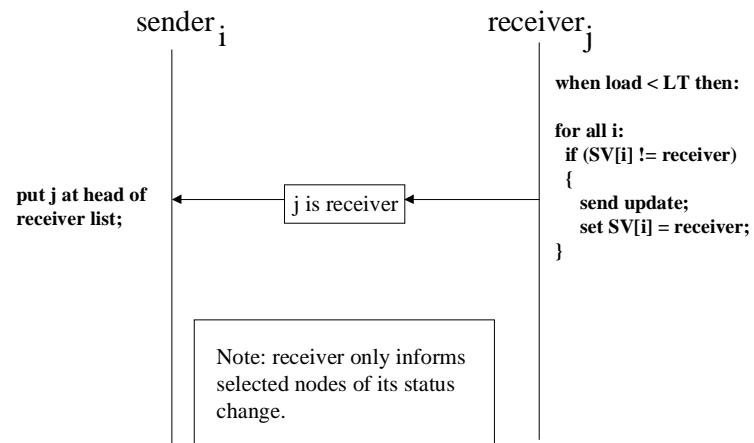


Note: the movement of node  $i$  to a different list on node  $j$  can only occur as a result of an interaction between nodes  $i$  and  $j$ . Thus, it is possible for node  $i$  to keep its information current.

## Sender Protocol



## Receiver Protocol



## Advantages

The sender-initiated algorithm:

- avoids broadcasting of receiver state
- does not transfer preempted tasks (because it is sender-initiated)
- is stable (as for previous algorithm)

## Selecting a Scheduling Algorithm

no high loads	sender-initiated
has high loads	stable algorithm
wide fluctuations	stable symmetric
wide fluctuations and high migration cost	stable sender-initiated