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

Task Distribution

In lightly loaded systems there is not much opportunity for task distribution because most servers are underutilized

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 an 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**

```
upper threshold (UT)
This node’s estimate of the systems average load
lower threshold (LT)

*thresholds equidistant from average
```

**Basic Step**

```
sender
  if (> UT)
  if (> UT)  
  if (still sender)
    Task
receiver
  TooLow
  TooHigh
  TooLow
  TooHigh
  TooHigh

if (receiver)  
  { increment load; send accept; }
```

**Timers**

```
sender
  if (> UT)  
  start timer
  (timer expires)
  RaiseAverage
receiver
  TooHigh
  broadcast
```

---
Timers

A Stable, Symmetrically Initiated Algorithm

Transmit 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.

\[ \text{on node } i \quad \text{SV:} \]

sender/receiver/OK

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

Sender Protocol

\[ \text{sender} \]

\[ \text{receiver} \]

 poll node at head of receiver list; set SV[j] = sender;

if (state j == receiver) {
  send task;
  done;
} else {
  put j on head of sender or OK list depending on state
}

reply current state; set SV[j] = state j

Sender continues polling until receiver list empty or task is transferred.

Receive Protocol

Receiver Protocol

\[ \text{sender} \]

\[ \text{receiver} \]

 when load < LT then:

for all i:
  if (SV[i] != receiver) {
    send update;
    set SV[i] = receiver;
  }

put j at head of receiver list;

1 is receiver

Note: receiver only informs selected nodes of its status change.

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

<table>
<thead>
<tr>
<th></th>
<th>sender-initiated</th>
</tr>
</thead>
<tbody>
<tr>
<td>no high loads</td>
<td></td>
</tr>
<tr>
<td>has high loads</td>
<td>stable algorithm</td>
</tr>
<tr>
<td>wide fluctuations</td>
<td>stable symmetric</td>
</tr>
<tr>
<td>wide fluctuations</td>
<td>stable sender-initiated</td>
</tr>
<tr>
<td>and high migration</td>
<td></td>
</tr>
<tr>
<td>cost</td>
<td></td>
</tr>
</tbody>
</table>