Chapter 5: Distributed Process Scheduling

CMSC 602
Advanced Operating Systems

- Static Process Scheduling
- Dynamic Load Sharing and Balancing
- Real-Time Scheduling
- Section 5.2, 5.3, and 5.5
- Additional reading: paper # 4, 5, 6 and 7.

Ju Wang, 2003 Fall Virginia Commonwealth University
Static Process Scheduling

- Given a set of partially ordered tasks, define a mapping of processes to processors before the execution of the processes.

- Cost model: CPU cost and communication cost, both should be specified in prior.

- Minimize the overall finish time (makespan) on a non-preemptive multiprocessor system (of identical processors)
  - Except for some very restricted cases, scheduling to optimize the makespan are NP-Complete
  - Heuristic solution are usually proposed
• This model is used to describe scheduling for ‘program’ which consists of several sub-tasks. The schedulable unit is sub-tasks.

• Program is represented by a DAG.

• Precedence constraints among tasks in a program are explicitly specified.

• critical path: the longest execution path in the DAG, often used to compare the performance of a heuristic algorithm.
• Scheduling goal: minimize the makespan time.

Algorithms:

• List Scheduling (LS): Communication overhead is not considered. Using a simple greedy heuristic: No processor remains idle if there are some tasks available that it could process.

• Extended List Scheduling (ELS): the actual scheduling results of LS with communication consideration.

• Earliest Task First scheduling (ETF): the earliest schedulable task (with communication delay considered) is scheduled first.

• what is the scheduling results of the above example when there are two processors? how about four processors?
Communicating Process Model

- There are no precedence constrains among processes

- Modeled by a undirected graph $G$, node represent processes and weight on the edge is the amount of communication messages between two connected processes.

- Process execution cost might be specified some times to handle more general cases.

- Scheduling goal: maximize the resource utilization.

- The problem is to find an optimal assignment of $m$ process to $P$ processors with respect to the target function: $\text{Cost}(G, P) = \sum_{j \in V(G)} e_j(p_i) + \sum_{i,j \in E(G)} c_{i,j}(p_i, p_j)$

  - $P$: a set of processors. $e_j(p_i)$: computation cost of execution process $p_i$ in processor $P_j$. 
- $c_{i,j}(p_i, p_j)$: communication overhead between processes $p_i$ and $p_j$.
- Assume a uniform communicating speed between processors.

- This is referred as Module Allocation problem. It is NP-complete except for a few cases:
  - For $P=2$, Stone suggested an polynomial time solution using Ford-Fulkerson’s maximum flow algorithm.
  - For some special graph topologies such as trees, Bokhari’s algorithm can be used.

Known results: The mapping problem for an arbitrary number of processors is NP-complete.

<table>
<thead>
<tr>
<th>Problem</th>
<th>optimal polynomial time algorithm</th>
<th>suboptimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 processor</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>2 proc. with varying load</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>tree-structured graph</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>series parallel graph</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>3 and more processor systems</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>
Maximum Flow Algorithm in Solving the Scheduling Problem

<table>
<thead>
<tr>
<th>Process</th>
<th>Cost on A</th>
<th>Cost on B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>inf</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>inf</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1: Computation cost
Figure 1: Convert to maximum flow problem, and find the minimum cut

- Generalized solution for more than two processor:
  1. Stone uses a repetitive approach based on two-processor algorithm to solve n-processor problems.
  2. Treat (n-1) processors as one super processor.
  3. The processors in the super-processor is further break down based on the results from previous step.
• Other heuristic: separate the optimization of computation and communication.
  – Assume communication delay is more significant cost
  – merge processes with higher interprocess interaction into cluster of processes
  – clusters of processes are then assigned to the processor that minimizes the computation cost
    * With reduced problem size, the optimal is relatively easier to solve (exhaust search)
  – A simple heuristic: merge processes if communication costs is higher than a threshold \( C \)
  – Also can put constrains on the total computation for the cluster, to prevent over clustering. Extreme case: the size of cluster become 1, meaning...?
Dual Processor Scheduling with Dynamic Reassignment

- Extended Stone’s method to dynamic assignment
- Dynamic process execution model:
  - Phase Residence cost Relocation cost
- Reduction of the cost graph when residence cost are all zero—reduce the size of the problem.
- Example: Fig 7 is reduced to Fig 9 in the paper.

- If a node has degree one and the only edge it has represents relocation cost, remove this node and the edge.
- If some nodes in the graph have a chain topology, it can shrunked by two nodes.
Dynamic Load Sharing and Balancing

- No prior knowledge is assumed
  - Scheduling need to be dynamic
  - Assignment decision made locally

- Based on disjoint process model

- Performance goal for scheduling is the high utilization of the system and equal fairness of user processes
Sender-Initiated Algorithm

- Activated by a sender process that wishes to offload some of its computation

- Transfer policy: When does a node become the sender?
  - queue size, ...

- Selection policy: How does a sender choose a process for transfer?
  - Newly-arrived processes, ...

- Location policy: which node should be the target receiver?
  - Randomly-chosen, polling for minimal load,...
Sender-Initiated Algorithm: cont.

Figure 2: Sender-initiated load balancing
Receiver-Initiated Algorithms

- A receiver can pull a process from others into its site for execution
  - Activates the pull operation when its queue length is shorter than RT
  - Require preemption capability
  - Which process to remove is not obvious

- Can be effective at highly-loaded systems

- Can be combined with sender-initiated algo.
A process migration facility need to:

- locate and negotiate a remote host
- Transfer the code image
- Initialize the remote operation
- Transfer its state information
- Transfer its communication information
  - Communication links
  - Messages in transit
Figure 3: Process Migration