Task Assignment and Scheduling

Scheduling real time tasks on distributed and multiprocessor systems consists of two subproblems :

1. Task allocation to the processor
   a. The task assignment problem is concerned with how to partition a set of tasks and then how to assign these tasks to processors -- task assignment can be : **1) Static or 2) Dynamic.**
   b. In the static allocation scheme, the allocation of tasks to nodes is permanent and does not change with time.
   c. In the dynamic task assignment, tasks are assigned to the nodes as they arise, different instances of tasks may be allocated to different nodes.
2. Scheduling of tasks on the individual processors : Uniprocessor scheduling algorithms can be used for the task set allocated to a particular processor.

Static allocation algorithms

The tasks are pre-allocated to processors.

No overhead incurs during run time since tasks are permanently assigned to processors at the system initialization time.

1. Utilization Balancing Algorithm
2. Next-Fit Algorithm for RMA
3. Bin Packing Algorithm for EDF

Dynamic allocation algorithms

In many applications tasks arrive sporadically at different nodes.

The tasks are assigned to processor as and when they arise.

The dynamic approach incurs high rim time overhead since the allocator component running at every node needs to keep track of the instantaneous load position at every other node.

1. Focussed Addressing and Binding (FAB)
2. The Buddy Strategy Algorithm

Utilization-Balancing Algorithm

This algorithm attempts to balance processor utilization, and proceeds by allocating the tasks one by one and selecting the least utilized processor.

Objective to balance processor utilization, and proceeds by allocating the tasks one by one and selecting the least utilized processor.

Maintains the tasks in a queue in increasing order of their utilizations. It removes tasks one by one from the head of the queue and allocates them to the least utilized processor each time.

The objective of selecting the least utilized processor is to balance the utilization of different processors.

$$\frac{\sum_{i=1}^{P}(u_i^B)^2}{\sum_{i=1}^{P}(u_i^*)^2} \leq \frac{9}{8}$$

Where u_i^* = P_i's utilization under an optimal algorithm that minimizes $\sum$utilization ^{2}

u_i^B = P_i's utilization under best-fit algorithm

Next-Fit Algorithm for RM-Scheduling

This is a utilization-based allocation heuristic. The task set has the same properties as for the RM uniprocessor scheduling algorithm.

M is picked by user.

Corresponding to each task class is a set of processors that is only allocated to tasks of that class.

It is possible to show that this approach uses no more than N times the minimum possible number of processors.

There are m classes of tasks such that, each class of tasks are assigned to a corresponding set of processors.

T_i belongs to class $j < m$ if

$$2^{\frac{1}{1+j}} - 1 < \frac{e_i}{p_i} \leq 2^{\frac{i}{j}} - 1$$

T_i belongs to class m otherwise.

Bin-Packing Assignment for EDF

Same assumptions on tasks and processors as Next-fit algorithm.

Problem : Schedule a set of periodic independent preemptible tasks on a multiprocessor system consisting of identical processors.

The task deadlines equal their periods and tasks require no other resources.

Solution : EDF - scheduling on a processor and task set is EDF - schedulable if $U \leq 1$

Assign tasks such that $U \leq 1$ for all processors.

The problem reduces to making task assignments to processors with the property that the sum of the utilizations of the tasks assigned to a processor does not exceed one.

Focused Addressing and Bidding (FAB) Algorithm

It uses dynamic allocations.

FAB is a simple algorithm that can be used as an online procedure for task sets consisting of both critical and non-critical real-time tasks.

Critical tasks must have sufficient time reserved for them so that they continue to execute successfully, even if they need their worst? Case execution time.

The non-critical tasks are either processed or not, depending on the system's ability to do so.

The guarantee can be based on the expected run time of the task rather than the worst-case run time (noncritical task).

THE UNDERLYING SYSTEM MODE IS : When a noncritical task arrives at processor $p_{i,}$ the processor checks to see if it has the resources and time to execute the task without missing any deadlines of the critical tasks or the previously guaranteed noncritical tasks — if yes, p_i accepts this new noncritical task and adds it to its list of tasks to be executed and reserves time for it.

The FAB ALGORITHM IS USED WHEN pi determines that it does not have the resources or time to execute the task in this case, it tries to ship that task out to some other processor in the system.

Every processor maintains two tables called : Status table and system load table

1. **STATUS TABLE** indicates which tasks have been already committed to rim including the set of critical tasks (which were preassigned statically) and any additional noncritical tasks that have been accepted, execution time and periods of the tasks.
2. **LOAD TABLE** contains the latest load information of all other processors of the system, the surplus computing capacity available at the different processors can be determined.

THE TIME AXIS is divided into windows, which are intervals of fixed duration, at the end of each window, each processor broadcasts to all other processors the fraction of computing power in the next window for which it has no committed tasks.

1. Every processor on receiving a broadcast from a node about the load position updates the system load table.
2. Since the system is distributed, this information may never be completely up to date.
3. As a result, when a task arrives at a node, the node first checks whether the task can be processed locally, if yes, it updates its status table if not, it looks for a processor to offload the task.

THE PROCESS OF OFF LOADING A TASK is based on the content of the system load table, an overloaded processor checks its surplus information and :

1. Selects a processor (called the focused processor) ps that is believed to be the most likely to be able to successfully execute that task by its deadline.
2. The system load table information might be out of date — the overloaded processor, as insurance against this, will decide to send Requests For Bids (RFB) to other lightly loaded processor in parallel with sending out the task to the focused processor ps, this is to gain time in case ps refuses the task.

The RFB contains the vital statistics of the task -- its expected execution time, any other resource requirements, its deadline, etc.

3. The RFB asks any processor that can successfully execute the task to send a bid to the focused processor ps stating how quickly it can process the task.
4. An RFB is only sent out if the sending processor ps estimates that there will be enough time for timely response to it.

Buddy Strategy

The buddy strategy tries to solve the same problem as the FAB algorithm, soft real time tasks arrive at the various processors of a multiprocessor and, if an individual processor finds itself overloaded, it tries to off load some tasks onto less lightly loaded processors.

The buddy strategy differs from the FAB algorithm in the manner in which the target processors are found.

STRATEGY

1. Each processor has 3 thresholds of loading : Under Loaded (TU), fully loaded (TF), and over loaded (TV).
2. The loading is determined by the number of jobs awaiting service in the processor's queue. If the queue length is Q, the processor is said to be in :
   a. State U (underloaded) if $Q \leq TU$;
   b. State F (fully loaded) if $TF < Q \leq TV$;

c. State V (overloaded) if Q > TV;

3. If in state U a processor is in a position to execute tasks transferred from other processors, if in state V, it looks for other processors on which to offload some tasks, if in state F will neither accept nor offload tasks from/to other processors.
4. When a processor makes a transition out of or into state U, it broadcasts an announcement to this effect.

THE TRANSITION to/from state U is broadcasted to a limited subset of processors called processor's buddy set, each processor is aware of whether any member of its buddy set is in state U. If it is overloaded, it chooses an underloaded member (if any) in its buddy set on which to offload a task.

ISSUES RELATED TO THE BUDDY SET

1. How the buddy set is to be chosen in a multi hop network? If too large the state-change broadcast will heavily load the interconnection network. If too small the success of finding an available U state processor will diminish.
2. If a node is in the buddy set of many overloaded processors, and it delivers a state-change message to them saying that now is underloaded. This can result in each of the overloaded processors dumping their load on this one processor and make it overloaded.
3. THE CHOICE OF THE THRESHOLDS TU, TF, AND TV, in general, the greater the value of TV, the smaller the rate at which tasks are transferred from one node to another.

Assignment with Precedence Conditions

Algorithm that assigns and schedules tasks with precedence conditions and additional resource constraints, basic algorithm idea is to reduce communication costs by assigning(if possible) to the same processor tasks that heavily communicate with one another.

UNDERLYING TASK MODEL : Each task may be composed of one or more subtasks. The release time of each task and the worst-case execution time of each subtask are given.

The subtask communication pattern is represented by a task precedence graph. Ee are also given the volume of communication between tasks.

It is assumed that if subtask si sends output to s2, this is done at the end of si; Associated with each subtask is a Latest Finishing Time (LFT).

The algorithm is a trial-and-error process -- assign subtasks to the processors one by one in the order of their LFT values -- for same LFT the subtask with the greatest number of successor wins.

Check for feasibility after each assignment, if one assignment is not feasible try another one, etc.

A threshold policy kcis followed when to characterize the volume of communication between tasks. When subtasks communicate a lot, if possible, they are assigned to the same processor.