Scheduling Divisible Tasks with Message Passing Interface

By: Jaime Ballesteros, PhD Student

Advisor:
Prof. Jaime Seguel

WALSAIP

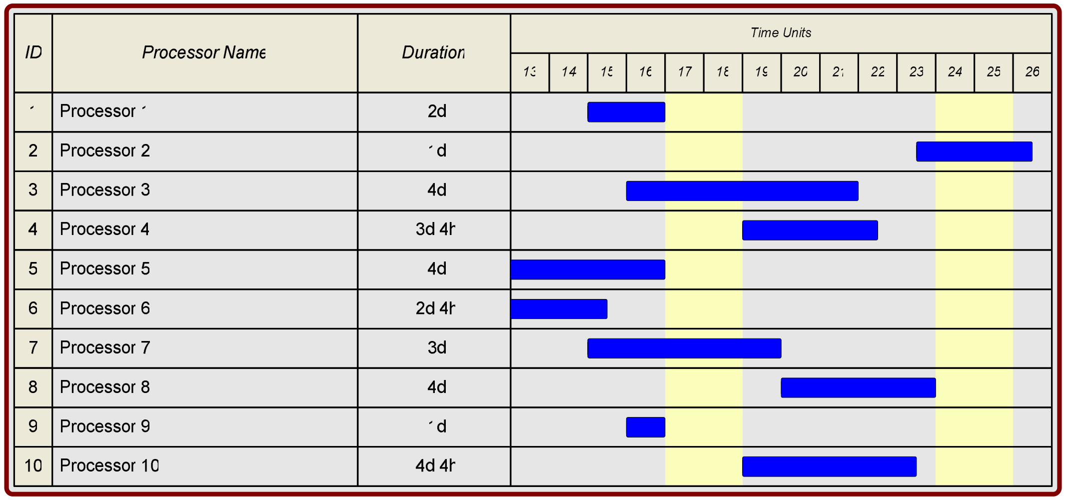
Parallel and Distributed Computing Laboratory
University of Puerto Rico at Mayaguez (UPRM)
May 2007

Problem formulation

How to effect task scheduling in a distributed system, specifically in numerical simulations with an MPI-flavor, in order determine the most effective orchestration of communications and computations that will give the optimal throughput of the system.

Justification:

Numerical simulations require optimal throughput in order to return accurate results in a reasonable time. Sometimes using a distributed system to compute simulations will give better results, but orchestration of communications and computations has to obey a predefined scheduling policy, that, in some cases, is unlikely to reach an optimal throughput in a expected time.



Methodology (Solution Approach)

Steady-state scheduling:

• Provides the asymptotically optimal throughput scheduler in master-slave applications (i.e. divisible applications).

• Solved in polynomial time with linear programming

$$\text{Maximize } n_{\text{task}}(G) = \sum_{i=1}^{p} \frac{\alpha_i}{w_i}$$

Subject to

$$\begin{array}{ll}
\forall i, & 0 \le \alpha_i \le 1 \\
\forall i, \forall j \in n(i), & 0 \le s_{ij} \le 1 \\
\forall i, \forall j \in n(i), & 0 \le r_{ij} \le 1 \\
\forall e_{ij} \in E, & s_{ij} = r_{ij} \\
\forall i, & \sum_{j \in n(i)} s_{ij} \le 1 \\
\forall i, & \sum_{j \in n(i)} r_{ij} \le 1 \\
\forall e_{ij} \in E, & s_{ij} + r_{ij} \le 1 \\
\forall i \ne m, & \sum_{j \in n(i)} \frac{r_{ij}}{c_{ij}} = \frac{\alpha_i}{w_i} + \sum_{j \in n(i)} \frac{s_{ij}}{c_{ij}} \\
\forall i \in n(m), & r_{mj} = 0
\end{array}$$

Elements of divisible task complexity theory:

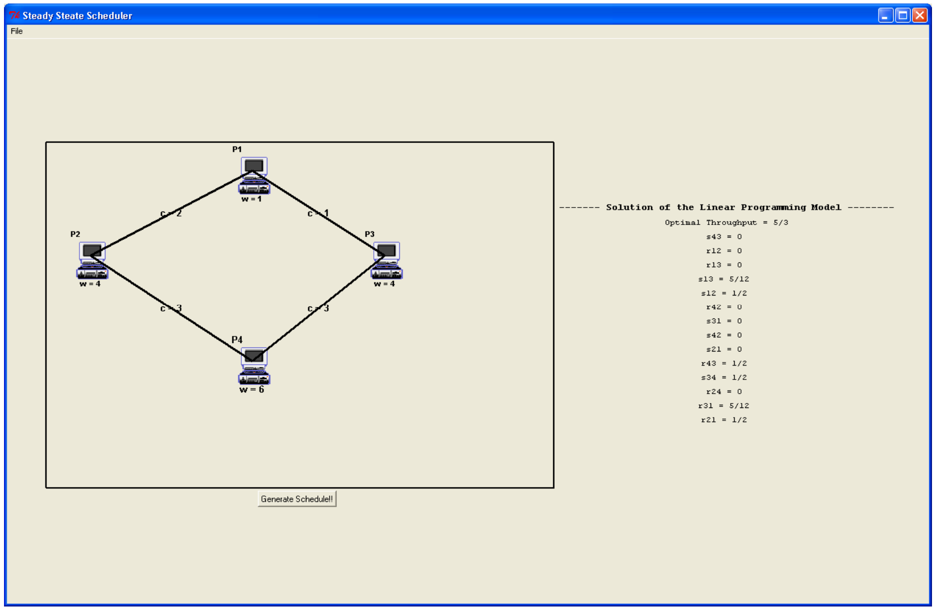
• Atomic tasks, unit tasks; volume and communications

• Provides the asymptotically shortest schedule for processes with single communication phase. All process end processing at the same point in time.

• Solved in polynomial time

The system will receive the user code, identify the atomic tasks and communication graph, applied the theoretical framework and emit code, pretty much in the spirit of the FFTW or some parallel data base search algorithms.

Applications Tools

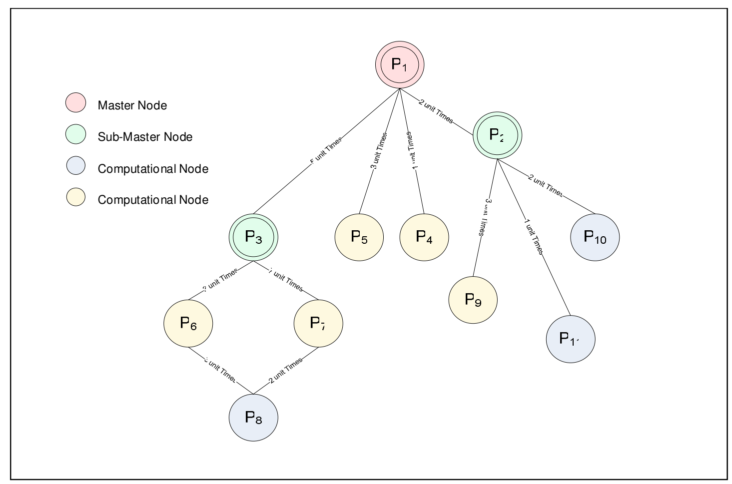


In our first study of Steady State scheduling theory we implement Steady State Scheduler V1.0 in Python and it:

- Allows to change communication and execution times.
- Uses Glpk® to solve the Master-Slave linear programming problem.
- Constructs a theoretical schedule.

Open MPI and GCC will be our primary tools:

Process and communication scheduling will be reflected in the source code and "controlled" with a feed-back control mechanism based on the communication flow.



Research Results

1. By using the demo, we could identify some observables in the system and the subsequent behavior in order to make an in-depth study of steady state scheduling mechanism.
2. We have extended (or perhaps) unified the theory of load divisible and steady state scheduling for application tasks that can be mapped as starts or trees.
3. We first modified the divisible load scheduler to make it periodic, saving thus a significant amount of start-up overhead. Then, we applied the same technique to the steady state scheduler to get a hybrid method that is provable superior to both of its ancestors.
4. We also developed a communication centric formulation of the new scheduler. Such formulation allows for the absorption of transients (decline in the processor or network speed).