ADAPTIVE STEADY-STATE SCHEDULING FOR GRID PLATFORMS

WALSAIP

www.walsaip.uprm.edu

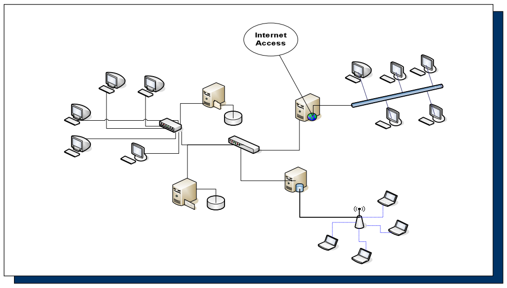
Jaime Andrés Ballesteros – PhD. Student
PDC Group, CISE Department,
Email: jaime.ballesteros@ece.uprm.edu

Luis De La Torre MSc.– PhD Student
CISE Department
Email: torre_dl@math.uprm.edu

Dr. Jaime Seguel – Advisor
University of Puerto Rico,
Mayagüez Campus

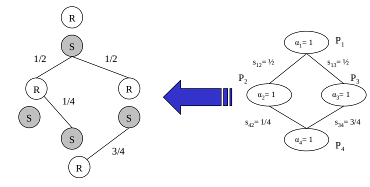
1 Problem Formulation

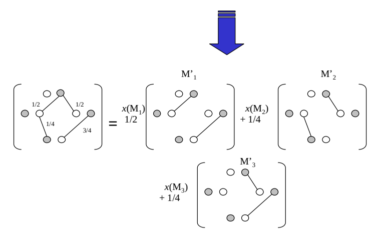
Harnessing the power of wide-area distributed computing platforms is a major challenge nowadays, and scheduling is crucial for achieving this goal. Traditional scheduling minimizes the makespan of the execution of a given set of jobs. In most practical situations, the exact computation of a minimal makespan is NP-Hard.



Construction of the schedule:

The actual schedule is generated by an algorithm for coloring a weight edge bipartite graph. This graph is decomposed into a weighted sum of matchings such that the sum of the coefficients is smaller than one.





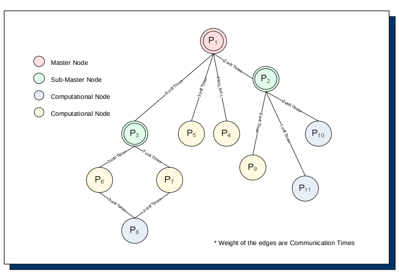
2 Proposed Solution

An interesting alternative introduced by D. Bertsimas, D. Gamarnik in 1999, is a schedule that optimizes the steady-state operation of the system. This approach is proven to be particularly well-suited for master-slave tasking, and in general, for divisible load applications.

Because the schedule is periodic and computable in polynomial time, it is possible to observe its actual performance in a period, inject that information into the polynomial methods, and re-compute the optimal steady-state schedule for upcoming periods. This is particularly useful in wide-area distributed systems where hard-to-predict communications jams may occur.

3 Theoretical Framework

Banino et al (2004) use a nonoriented graph to model a hybrid computer platform. The optimal steady state is defined as the fraction of time spent computing and the fraction of time spent sending or receiving tasks along each communication links, so that the overall number of tasks processed at each time step is maximum



Graphical representation of a Master – Slave Computing Platform

Master Slave Scheduling Problem:

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

Subject to

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

Master-Slave Scheduling Linear Program

A linear program maximizes the throughput in the system:

Let **G = (V,E,*w*,*c*)** be the platform graph model and

w_i: the weight of the node *Pi* in G represents units of time required for P_i to process one task.

c_{ij}: be the weight of the edge between the nodes *Pi* and *Pj*, which represents the time needed to communicate one task in both directions.

α_i: the fraction of time spent by P_i computing,

s_{ij}: the fraction of time spent by P_i sending tasks to each neighbor processor P_j

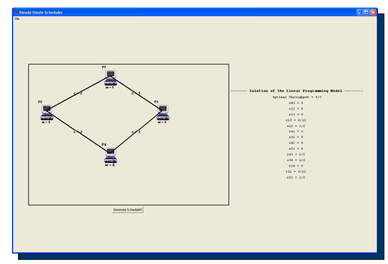
r_{ij}: the fraction of time spent by P_i receiving tasks from each neighbor processor P_j.

4 Demo Construction

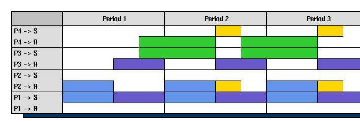
Our demo (Steady State Scheduler V 1.0) was written in Python® and allows us to change execution and computation times in order to show the effects of adaptivity.

This demo:

- Allows to change communication and execution times.
- Uses Glpk® to solve the Master-Slave linear programming problem.
- Uses the previous algorithms to build the schedule
- Display the generated schedule in a Gantt chart.



Example execution of Steady State Scheduler V. 1.0 Program



Gannt Chart of the new schedule

5 Conclusion

Currently the demo accepts variations in communication and execution times. By observing these variations it become apparent that, with one master, no throughput is higher than two, independent of the number of nodes, communication and execution times.

We expect to evaluate a problem with more nodes, simulating a grid based platform and we will used this results in order to construct a prototype system related to a specific problem.

6 References

C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand and Y. Robert, Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms, IEEE Transactions on Parallel and Distributed Systems 15 (4) (2004) 319-330.

Li X. and Zang W., A Combinatorial Algorithm for Minimum Weighted Colorings of Claw-free Perfect Graphs, Journal of Combinatorial Optimization. The Netherlands, Springer, 2005, 9: 331-347. (Publication No. : 108691)

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Introduction to Algorithms, second edition, MIT Press and McGraw-Hill. ISBN 0262531968

Papadimitriou, Christos H.; Steiglitz, Kenneth. Combinatorial Optimization: Algorithms and Complexity, Dover Publications Inc. 1998. ISBN 0486402584

Supported By