Scheduling Parallel Jobs on Shared Resources

Cynthia Phillips*, Sandia National Laboratories
Michael A. Bender, State University of New York, Stony Brook

Summary

We proved that simple randomized scheduling algorithms can effectively use grid computing resources in the worst case even with maximum volatility in machine speed.

The MICS Combinatorial and Global Optimization project team designs and analyzes efficient algorithms for computing (near) optimal solutions to problems with explicit combinatorial structure. This year's accomplishments include enhancements to the PICO parallel integer programming solver. We have also developed formal insight into the underlying reasons for the superior performance of tabu search metaheuristics for capacitated vehicle routing problems. This is a simplified version of the problem faced by DOE in the transportation of special materials. Furthermore, our previous research on finding compact formulations for integer programming problems, first developed for protein comparison, had significant impact in a project outside of MICS. It reduced the running time of an industrial logistics problem from 24 hours to 12 minutes.

This summary focuses on a particular subproblem within this broader scope: scheduling parallel tasks on shared processors such as a grid computing environment. When users compete for shared resources, individual users receive varying and unpredictable service due to the changing loads on the machines. We proved that even with worst-case speed changes, simple randomized algorithms can complete a set of parallel tasks in near-optimal time.

We model the parallel computation as a directed acyclic graph (DAG) such as the one illustrated in Figure 1. Each node represents a subtask of the computation. For this first study we assume all subtasks require the same amount of computation (work). A directed edge from node n_1 to node n_2 represents a precedence constraint: node n_1 must complete before n_2 can begin. This structure is typical of search-based optimization problems. For example, when searching for parameters that elicit extreme behavior in a simulation, the result of one simulation may determine the parameters for the next. This structure is revealed online, meaning a subtask is not known until one or more of its immediate predecessors finish.

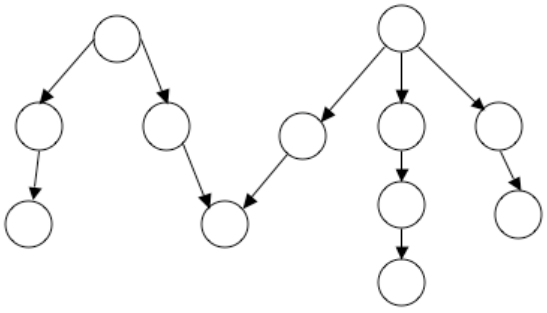

Figure 1 A parallel computation represented as a directed acyclic graph (DAG). An edge

* (505) 845-7296, caphill@sandia.gov

represents a precedence constraint. A task is ready only if all predecessors have finished.

We wish to execute this parallel job on a set of shared processors. Because we cannot control the behavior of other users, the performance of the machines is unpredictable. A machine may even effectively fail after we have dispatched a job. We cannot preempt and restart a job, nor can we migrate to a new processor. However, we can dispatch it to multiple processors and kill duplicates once one succeeds. We wish to minimize the time required to finish the job (i.e. the completion time of the last subtask to finish).

We model machine performance volatility by allowing an adversary to determine the speed of each processor over time. Processors may run arbitrarily fast or slowly and change speeds arbitrarily often. Since processor performance variation is extreme, we cannot use a tool like Network Weather Service (NWS) to predict processor performance. The adversary knows the structure of the program. But our scheduler is allowed to make random choices that the adversary does not know.

Two properties of the DAG effect the job's completion time: the total amount of work and the critical path length. The critical path is the longest chain of precedence constraints, a measure of inherent serial computation. The scheduler does not know W, the total amount of work in the system or D, the length of the critical path. We say a problem is *work-dominated* for a system with p processors if the average work per processor is log p times larger than the critical path length. It is *path-dominated* if the critical path is log p times larger than the average work. Otherwise, it is *balanced*. In describing algorithm performance, we consider two quantities. T_w is the time the system requires to complete the job if all processors are always busy doing non-redundant work. No schedule can finish the job faster than time T_w. T_d is the amount of time required to finish the critical path if a single processor of average speed worked on it consistently. This is not a lower bound on the completion time, since faster processors might work on the critical path, but it is closely related.

We consider two randomized algorithms. In the ALL algorithm, whenever a processor is free, we assign it a task selected uniformly at random from all ready tasks. The LEVEL algorithm selects only from ready tasks closest to a start node of the DAG. ALL tries to reduce the remaining work. LEVEL tries to reduce the remaining critical path.

We proved for work-dominated problems, both algorithms run in (optimal) time[1] T_w. For path-dominated problems, ALL runs in time T_d and LEVEL runs in time log*p T_d where log*p is the number of iterated logarithms needed to bring p to a constant (a small constant for any reasonable value of p). In the balanced case, LEVEL runs in time T_w + log*p T_d and ALL runs in time $(\log p)^k T_w + (\log p)^{1-k} T_d$, where $0 \leq k \leq 1$ depends on balance. There is an example with this runtime. Thus in the balanced case, it is provably better to replicate low-level jobs than to consider all ready jobs.

This shows we can effectively use grid resources even with maximum speed volatility. Randomly replicating job dispatches protects against uncertainty in the worst case with little cost in performance.

For further information on this subject contact:
Dr. Anil Deane, Program Manager
Mathematical, Information, and Computational Sciences Division
Office of Advanced Scientific Computing Research
Phone: 301-903-1465
deane@mics.doe.gov

1. All times are asymptotic, meaning we ignore constant multiplicative factors.