Reinforcement Learning for Selfish Load Balancing in a Distributed Memory Environment

Presented by: Hoi Fung Lam

Based on the paper: S. M. Majercik and M. L. Littman., "Reinforcement learning for selfish load balancing in a distributed memory environment," in *Proceedings of the International Conference on Information Sciences*, pp. 262–265, 1997.

Introduction

1. Perspectives on load balancing
2. Simulation model
3. Simulation algorithm
4. MDP formulation
5. MDP solving

Perspectives

- Altruistic load balancing — given a series of processes with varying interprocess communication needs, the goal is to schedule the processes on the network in order to minimize the time to completion of *all* processes.
- Solipsistic load balancing — assumes that one computation has exclusive use of the whole system, and the goal is to distribute the computation in order to minimize the time to completion of this *single* computation.
- Selfish load balancing — focus on completing one particular computation as fast as possible, but also recognizes that there are other jobs on the system and that no one job has exclusive use of the processors.

Molecular Dynamics Simulation Model

- Given n processors and t MD steps to compute, an agent allocates the MD work among the processors at the beginning of each MD step.
- Agent monitors the processor loads and the progress.
- At each processor time step, the agent decides whether to let the computation continue or to intervene and reallocate the work, losing all the computation done so far on this MD step.
- The amount of work allocated to each processor was quantify by estimating the total dedicated processor time needed to calculate the MD step.

MD Simulation Algorithm

1. Update the size of the processor run queues assuming Poisson arrivals with varying means, exponential service time with a specified mean that does not vary.
2. Calculate f_i, the fraction of the quantum available for the MD computation on processor p_i:

   $$f_i = e^{-\beta_i r_i}$$

   where β_i is a constant and r_i is the number of jobs in the run queue of p_i. The amount of work get done in a quantum, w_i is:

   $$w_i = f_i q$$

   where q is the wall-clock duration of a quantum.
3. Advance wall-clock time a single quantum.

Load Balancing as an MDP

state vector $= (\vec{L}, \vec{D}, \vec{J}, H, M, Q, R, B)$

- $\vec{L} = L_i$, load expressed as number of jobs in the run queue of the processor i
- $\vec{D} = D_i$, amount of MD work allocated at processor i
- $\vec{J} = J_i$, amount of MD work remaining at processor i
- $H : M =$ 24-hour wall-clock time
- Q number of quanta used so far in an MD step
- R binary scalar indicating whether we are at the beginning of an MD step
- B bias feature

Load Balancing as an MDP (cont.)

action

- even allocation
- allocation according to Boltzmann distribution, processor p_i gets a fraction of the total work equal to $\exp(-r_i)/(\sum_i \exp(-r_i))$ where r_i is the number of jobs in the run queue of p_i.
- repeat the previous action
- do nothing

cost and reward

- explicit: 1 or 10 for each action, 1000 when the entire sequence of MD steps has been computed
- implicit: time needed to poll the processors to obtain Boltzmann distribution cost a delay of 5 quanta

Solving the MDP

The MDP is solved using guided off-line Q-learning with linear function approximation. Actions are chosen with a bias toward the naively correct action. The update equation for the linear-function coefficient c_i is:

$$c_i = c_i + \alpha x_i \; sigmoid(dv)$$

where α is the learning rate, x_i is the i^{th} feature of the current state vector, and dv is the difference between the target value and the value of the current state. Sigmoid function is used because some features quickly become substantially larger than others and tend to dominate the state.