Reinforcement Learning for Capacity Tuning of Multi-core Servers

Liat Ein-Dor
Intel Research Israel Labs

Yossi Ittach
Intel Research Israel Labs

Aharon Bar-Hillel
Intel Research Israel Labs

Amir Di-Nur
Intel IT

Ran Gilad-Bachrach
Intel Research Israel Labs

Abstract

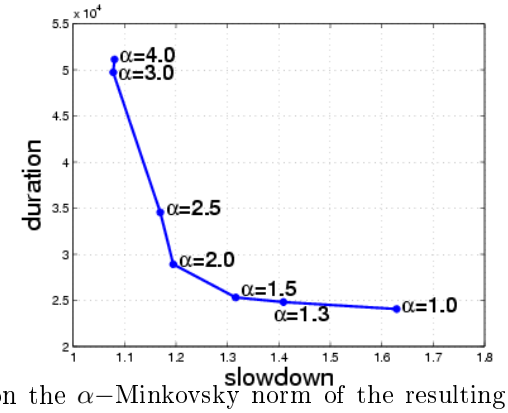
The computing power in mainstream computers keeps on growing in exponential rates. However, the progress in the serial computing power has slowed down and much of the progress is achieved by making parallel computing the mainstream. Indeed, two-core and Quad-core computers are already widely spread, and 80 cores CPUs have already been demonstrated. At the same time, more and more software product uses the parallel computing power by multi-tasking, threading and other techniques. In this setting, the question "how many jobs should a server concurrently run?" becomes more involved, and the optimal number depends in a non-trivial way on the job's nature and the exact optimality criterion. We claim here that reinforcement learning techniques can be used to continuously monitor the number of running jobs, which we term the *server's capacity,* thus leading to significant performance improvements.

A server's performance can be measured in several dimensions, and the most important quantities are throughput—the number of successful job executions per time unit, and average job slowdown—the time it takes a job to complete compared to the time it would have taken if it was singly running on a dedicated machine. Increasing throughput and reducing slowdown are often conflicting aims[3], since running more jobs in parallel increases throughput but it also increases the time it takes to finish each job. The default manually-tuned policy used in our organization ignores this trade-off, and simply sets the number of concurrently running jobs equal to the number of cores the server has. We suggest to replace this policy with a dynamic learning agent optimizing both quantities, with the trade-off between them determined by the system administrator.

An important observation for the capacity tuning problem is that a static policy, which does not respond to changes in job types or machine loads, can only reach sub-optimal resource utilization. For example, two heavy jobs with high memory usage are often run in parallel, causing memory swaps that reduce throughput dramatically. Similarly, running "multi-core aware" jobs together, each of which fully utilizes the computer resources when run alone, increase job slowdown in comparison with sequential runs, without any increase in throughput value. The opposite situation occurs for light jobs with low CPU and memory requirements. Many such jobs can be packed together with almost no impact on slowdown and a significant improvement in throughput.

We pose the problem as an MDP with three possible actions for the agent: *accept* an additional job, *resubmit* a running job (for later execution), and *keep* the current situation. The agent is activated in fixed time intervals and the state space it monitors is a continuous vector in R^7, including measurements as free memory, machine load, system/idle time and number of currently running jobs. To compute the state's reward, we measure the fraction of CPU utilization of all running jobs in the last interval. The reward is defined based

Figure 1: **Right: The role of α.** The trade-off between slowdown (on the X-axis) and workload duration (1/throughtput on the Y-axis) as α varies. Results were obtained using simulated workload with the Iterative Q-fitting algorithm. **Left: The advantage of ML-tuned servers:** statistics of throughput improvement obtained by fitted Q-iteration in 10-days experiments with real workloads. Improvement duration measures the fraction of time in which the ML-tuned machines were superior to manually-tuned machines.



Improvement	2-core	4-core
Mean	6.7%	8.6%
Median	2.4%	6.3%
duration	80%	99%

on the $\alpha-$Minkovsky norm of the resulting utilization vector. For $\alpha = 1$ this reward is just the total fraction of CPU time used, but for $\alpha > 1$ states with high concentration of CPU resources on a single job are prefered. Changing α moves the emphasize between throughput enhancement (requiring high CPU utilization and hence low α values) and slowdown reduction (requiring CPU concentration on a single job).

We learnt control policies for capacity tuning using three reinforcement learning algorithms, representing different approaches: Hidden Markov Model (HMM) + Linear Programming (LP)[1], online learning with TD-λ[4], and fitted Q-iteration with Parzen window regression[2]. Experiments on a small grid using simulated workload have shown that all three algorithms were able to learn policies which outperform the manual policy both in terms of reward and of server's throughput. The impact of the reward parameter α was teseted empirically, and varying it enabled us to produce a trade-off curve between throughput and average job slowdown (see figure 1 right). The fitted Q-iteration algorithm performed slightly better then its competitors, and was hence used in further experiments done on a living grid with real jobs. In these experiments the learning agent was tested on 2-core and 4-core machines, and achieved stable and significant throughtput enhancements of 6-9% over manually-tuned servers (see figure 1 left). Such enhancement implies possible annual saving of dozens of million dollars for organizations employing large grids or batch systems.

References

[1] L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *National Conference on Artificial Intelligence*, pages 183–188, 1992.

[2] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 6:503–556, 2005.

[3] A. Snavely and J. Kepner. 99% utilization—is 99% utilization of a supercomputer a good thing? In *ACM/IEEE conference on Supercomputing*, page 37, New York, NY, USA, 2006. ACM Press.

[4] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.