Towards Autonomic Computing: Adaptive Network Routing and Scheduling

Shimon Whiteson and Peter Stone
Department of Computer Sciences, The University of Texas at Austin
{*shimon,pstone*}@*cs.utexas.edu*, `http://www.cs.utexas.edu/~{shimon,pstone}`

Introduction

Computer systems are rapidly becoming so complex that maintaining them with human support staffs will be prohibitively expensive and inefficient. In response, visionaries have begun proposing that computer systems be imbued with the ability to configure themselves, diagnose failures, and ultimately repair themselves in response to these failures. However, despite convincing arguments that such a shift would be desirable, as of yet there has been little concrete progress made towards this goal.

We view these problems as fundamentally *machine learning* challenges. Hence, we define and study learning-based methods for addressing the problems of packet routing and CPU scheduling in (simulated) computer networks. Our experimental results verify that methods using machine learning outperform heuristic and hand-coded approaches on an example network designed to capture many of the complexities that exist in real systems.

To pursue our research goals, we have created a high-level simulator that is capable of modeling the relevant types of interactions among the many different components of a computer system. The simulator represents a computer network as a graph: nodes represent machines or users and links represent the communication channels between them. Users create jobs that travel from machine to machine along links until all of their steps are completed. When a job is completed, it is assigned a score according to a given *utility function*, which depends on how much time the job took to complete. The utility function may vary depending on the type of job or the user who created it.

All of the experiments presented in this paper were performed on the network depicted in Figure 1. This network, although abstract, nonetheless captures the essential difficulties posed by such optimization problems. For example, each Database has a different speed, which means that if the Database Load Balancer routes packets randomly, it will overload the slower Databases and underload the faster ones. Even if we take machine speed into account, routing on this network is far from trivial. Consider a Web Server that wishes to forward a Mail Job to a Mail Server. Since there is no load balancer governing the Mail Servers, the Web Server must pick directly from among the Mail Servers to which it is connected. Doing so optimally requires considering not just the speed of each Mail Server but also how busy that server is completing jobs sent by other Web Servers. In addition, the scores assigned to each job are determined by the non-linear utility functions reflecting the relative importance of the two users.

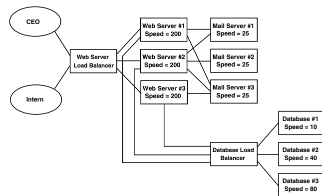

Figure 1: The network used in our experiments; ovals represent users and rectangles represent machines; the lines between them represent links that allow communication of jobs or other packets.

Method

Routing

To improve the performance of our network, we adapt a technique called Q-routing (Boyan & Littman 1994), an online learning technique in which a reinforcement learning module is inserted into each node of the network. In Q-routing, each node x maintains a table of estimates about the *time-to-go* of packets if they are routed in various ways. Each entry $Q_x(d, y)$ is an estimate of how much additional time a packet will take to travel from x to its ultimate destination d if it is forwarded to y, a neighbor of x. If x sends a packet to y, it will immediately get back an estimate t for x's time-to-go, which is based on the values in y's Q-table:

$$t = min_{z \in Z} Q_y(d, z)$$

where Z is the set of y's neighbors. With this information, x can update its estimate of the time-to-go for packets bound for d that are sent to y. If q is the time the packet spent in x's queue and s is the time the packet spent traveling between x and y, then the following update rule applies:

$$Q_x(d, y) = (1 - \alpha) Q_x(d, y) + \alpha(q + s + t)$$

where α is a learning rate parameter (0.7 in our experiments). In the standard terms of reinforcement learning (Sutton & Barto 1998), $q + s$ represents the instantaneous reward (cost) and t is the estimated value of the next state, y.

By bootstrapping off the values in its neighbors' Q-tables, this update rule allows each node to improve its estimate of a packet's time-to-go without waiting for that packet to reach its final destination. This approach is based directly on the Q-learning method (Watkins 1989). Once reasonable Q-values have been learned, packets can be routed efficiently by simply consulting the appropriate entries in the Q-table and routing to the neighbor with the lowest estimated time-to-go for packets with the given destination.

Scheduling

The routing technique discussed above attempts to minimize the time that passes between the creation and completion of

Proceedings of the International Conference on Autonomic Computing (ICAC'04)
0-7695-2114-2/04 $20.00 © 2004 **IEEE**
IEEE COMPUTER SOCIETY

a job. However, this completion time is only indirectly related to the score, which it is our goal to maximize. The score assigned to any job is determined by the utility function, which can be different for different types of jobs or users. When jobs do not all have equal importance, minimizing the completion time of less important jobs can be dramatically suboptimal because it uses network resources that would be better reserved for more important jobs.

To address these difficulties, we combine Q-routing with a new technique for prioritizing the jobs waiting to use a machine's CPU, called the *insertion scheduler*. Each time a new job arrives at a given machine, the scheduler must decide how best to reorder all the jobs in the queue. The utility of an ordering is the sum of the constituent jobs' scores and a given job's score is a known function of completion time. Thus, the problem of estimating an ordering's utility reduces to estimating the completion time of all the jobs in that ordering. A job's completion time can be estimated by summing the following three values: 1) how old the job was when it arrived at the current machine, 2) how long the job will wait in this machine's queue given the considered ordering, and 3) how much additional time the job will take to complete after it leaves this machine. The first factor is known and the second factor is easily computed given the speed of the machine and a list of the jobs preceding this one in the ordering. The third factor is not known but, if the insertion scheduler is coupled with a Q-router, we have an estimate of the needed value in the machine's Q-table.

Since computing the utility of each of the $n!$ possible orderings is computationally infeasible, the insertion scheduler instead uses a simple and fast heuristic. For each new job, it decides at what position to insert it into the current ordering such that utility is maximized. Hence, it needs to consider only n orderings.

Results

To evaluate our routing and scheduling methods, we ran a series of experiments on the example network described above. Figure 2 shows the result of the first set of experiments, in which Q-routing is compared to two methods that do not use learning: a *random router*, which chooses blindly among its neighbors, and a *heuristic router*, which routes in proportion to the CPU speed of its neighbors. Each method is employed in conjunction with a FIFO scheduler in simulations that run for 20,000 timesteps. At timestep #10,000, a network catastrophe is simulated in which Web Server #1, Mail Server #1, and Database #3 simultaneously go down and remain down for the duration of the run. The scores are averaged over 25 runs.

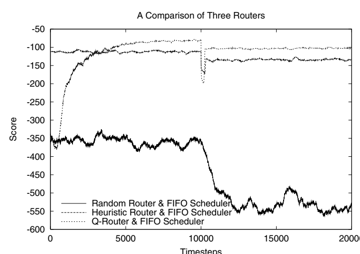

Figure 2: A comparison of three routing methods. A network catastrophe occurs at timestep #10,000.

The graph clearly indicates that routing randomly is dramatically suboptimal. The heuristic router, which routes in proportion to the speed of its neighbors, performs much better. The Q-routing method, whose initial policy lacks even the primitive load balancing of the random router, starts off as the worst performer but improves rapidly and plateaus with a higher score than the heuristic. Both the Q-router and the heuristic router are able to respond gracefully to the loss of machines in the middle of the simulation; the Q-router maintains its superiority afterwards. The random router, however, suffers a major loss of performance from the sudden change in network configuration.

Figure 3 shows the results of a second set of experiments. These experiments are identical to the first except that now the insertion scheduler is compared to two alternative schedulers: a FIFO scheduler and a *priority scheduler*, which always puts the CEO's jobs at the front of the queue. All three are used in in conjunction with our best router, the Q-router.

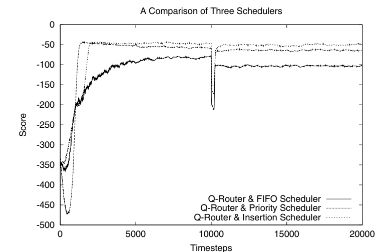

Figure 3: A comparison of three scheduling methods. A network catastrophe occurs at timestep #10,000.

The FIFO scheduler, which ignores complications that arise from the utility functions, performs relatively poorly, while the priority and insertion schedulers fare much better. Though the lines appear close in the graph, the insertion scheduler, by capitalizing on the information learned in each machine's Q-table, obtains scores that are approximately 20% higher than those of the priority scheduler. Since all of these runs use Q-routing, they all recover from the network catastrophe at timestep #10,000.

Discussion

Our experimental results indicate clearly that machine learning methods offer a significant advantage in optimizing the performance of complicated networks. Both the router and scheduler placed in each machine benefit substantially from the time-to-go estimates discovered through reinforcement learning. Furthermore, the best performance is achieved only by placing intelligent, adaptive agents at more than one level of the system: the Q-router and the insertion scheduler perform better together than either could apart. Hence, they benefit from a sensible division of optimization tasks; the router focuses on routing jobs efficiently and balancing load throughout the network while the scheduler focuses on prioritizing those jobs whose effect on the score will be most decisive.

References

Boyan, J. A., and Littman, M. L. 1994. Packet routing in dynamically changing networks: A reinforcement learning approach. In Cowan, J. D.; Tesauro, G.; and Alspector, J., eds., *Advances In Neural Information Processing Systems 6*. Morgan Kaufmann Publishers.

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

Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation, King's College, Cambridge, UK.

0-7695-2114-2/04 $20.00 © 2004 **IEEE**
IEEE COMPUTER SOCIETY