Improvement of Systems Management Policies Using Hybrid Reinforcement Learning

Gerald Tesauro¹, Nicholas K. Jong², Rajarshi Das¹, and Mohamed N. Bennis³

¹ IBM TJ Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 USA

² Dept. of Computer Sciences, Univ. of Texas, Austin, TX 78712 USA

³ Dept. of Computer Science, George Mason Univ., Fairfax, VA 22030 USA

Abstract. Reinforcement Learning (RL) holds particular promise in an emerging application domain of performance management of computing systems. In recent work, online RL yielded effective server allocation policies in a prototype Data Center, without explicit system models or built-in domain knowledge. This paper presents a substantially improved and more practical “hybrid” approach, in which RL trains offline on data collected while a queuing-theoretic policy controls the system. This approach avoids potentially poor performance in live online training. Additionally we use nonlinear function approximators instead of tabular value functions; this greatly improves scalability, and surprisingly, eliminated the need for exploratory actions. In experiments using both open-loop and closed-loop traffic as well as large switching delays, our results show significant performance improvement over state-of-art queuing model policies.

1 Introduction

The ongoing rapid growth in scale and complexity of the world’s IT infrastructure has motivated intense focus on automating management of computing systems. Major IT vendors and universities have recently initiated research on “autonomic” computing systems, seeking means by which computing systems may dynamically reconfigure themselves, continually optimize their performance, detect and repair faults, and protect themselves from external attacks. Machine learning may prove to be of great benefit in developing such capabilities. While standard approaches to systems management rely on extensive domain knowledge, ML may evade the knowledge bottleneck by automatically learning high-quality management policies based solely on observed data.

Reinforcement Learning (RL) methods hold particular promise for systems performance management. RL can obtain high-quality policies without explicit models or extensive built-in system knowledge. Also, by accounting for the long-range consequences of decisions, RL can surpass other methods that treat dynamical effects only approximately, or ignore them altogether (e.g. traditional steady-state queuing theory), or cast the decision problem as a series of unrelated instantaneous optimizations.

Initial work [1,2,3] applying RL to systems management shows promise, but we are concerned with two potentially significant practical problems. First, the above studies suggest that tens of thousands of observations may be required, at natural intervals ranging from several seconds to several minutes of real time. This implies online training times could be as long as several months, which would be unacceptable in many

applications. Second and perhaps more importantly, the performance obtained during live online training may be unacceptably poor, due to two factors: (a) an arbitrarily bad initial policy in the absence of domain knowledge or good heuristics; (b) in general RL procedures also need a certain amount of exploration of suboptimal actions, which may be exceedingly costly to implement in a live system.

To address the above practical limitations, we devise in this paper a hybrid method combining the advantages of both explicit model-based methods and *tabula rasa* RL. Instead of training an RL module online on the consequences of its own decisions, we propose offline training on data collected using a stationary external policy (based e.g. on an appropriate queuing model) to manage the system. This assures acceptable performance while gathering training data, assuming a decent initial policy.

Offline sweeps through the acquired dataset may be orders of magnitude faster than the underlying physical time scales. This permits multiple sweeps through the dataset, enabling training of sophisticated nonlinear value function approximators which learn too slowly to be trained online. Function approximators generalize training experience across states and actions, reducing the need for extensive exploration.

The idea of combining RL with an external policy is not new. Our hybrid RL method is similar to the “implicit imitation” framework of [4], in which an agent learns based on the state transitions and rewards generated using the policy of another agent.

The main contributions of our work are as follows:

1. We demonstrate effectiveness of hybrid RL in a prototype system comprising real servers, realistic Web-based workloads, and realistic time-varying demand. Our results significantly outperform state-of-art queuing models and are much better than our prior online RL approach [2] that only achieved equality to queuing models. To our knowledge this is the first time that queuing models have been surpassed in a transactional performance management task, an achievement considered both significant and impressive by systems experts [5].
2. We provide the first demonstration in a systems management application that RL can deal effectively and automatically with dynamic consequences of management decisions – transients and switching delays – associated with server reallocation.
3. Several aspects of our results may be counterintuitive or surprising to a machine learning audience and hence may spur further research. One such finding is that our methodology works quite well in practice, despite lacking rigorous convergence guarantees. Another surprising finding is that we did not need to add any exploration to the base queuing model policies. Finally, it seems non-intuitive that hybrid RL can outperform queuing models while receiving only a subset of the available inputs.
4. Our approach is much more practical and scalable than the online approach of [2], since we avoid poor performance during online training, and function approximators offer much better scalability than lookup tables. Consequently, we anticipate possible wide usage of hybrid RL in many different types of systems management applications.

The rest of the paper is organized as follows. Section 2 describes our prototype Data Center. Section 3 presents specifics of our hybrid RL approach. Sections 4 and 5

give performance results and discuss why RL outperforms the queuing models. Conclusions are given in Section 6. Our queuing model policies are detailed in [6].

2 Experimental Setup

Our experimental testbed, illustrated in Fig. 1, follows the scenario in [1] for dynamically allocating a set of identical servers among multiple web applications. Each application has its own Application Manager module which communicates with a Resource Arbiter module regarding resource needs. Allocation decisions are made in five-second time intervals as follows: Each Application Manager i reports to the Arbiter a commonly scaled utility curve $V_i(\cdot)$ estimating expected business value as a function of number of allocated servers. Business value is defined in monetary units by a Service Level Agreement (SLA), which stipulates payments or penalties as a function of performance. Upon receipt of the utility curves, the Arbiter solves for the globally optimal allocation maximizing total expected value. It then conveys to each application a list of assigned servers, which are used in dedicated fashion until the next allocation decision.

Our standard testbed uses eight HTTP servers (3.06GHz Xeon machines) and three applications. Two of the applications are separate instantiations of “Trade3,” a web application that provides a realistic emulation of online trading. Each Trade3 SLA is a sigmoidal function of mean response time over the allocation interval. The third application is a long-running “Batch” workload that can be paused and restarted as servers are added and removed. This emulates a CPU-intensive task such as Monte Carlo portfolio simulations. Since Batch has steady need for resource, its SLA is a simple increasing function of number of assigned servers.

Fig. 1. Resource allocation scenario

Demand in each Trade3 environment is driven by a separate workload generator, which can operate either in open-loop or closed-loop mode. The open-loop mode generates Poisson HTTP requests with an adjustable mean arrival rate. The closed-loop mode simulates an adjustable, finite number of customers who alternate between waiting for a request to complete and thinking about sending the next request, with fixed think time distributions. To emulate stochastic bursty time-varying demand, we use a realistic Web traffic model [7] to adjust once per second either the closed-loop number of customers or the open-loop mean arrival rate.

3 Hybrid RL Approach

In the first part of our hybrid RL procedure, given in Algorithm 1, one obtains some initial policy and would expect that adding some form of off-policy exploration (e.g. softmax or ϵ -greedy) would be necessary to facilitate learning. However, we were surprised to discover that we can learn substantially improved policies without any added exploration, so this step may not be necessary. One then runs the initial policy in the system and records a set of $(T + 1)$ observations $\{(s_t, a_t, r_t), 0 \leq t \leq T\}$, where (s_t, a_t, r_t) are the observed state, action and immediate reward at time t . Using these data, we train a value function $Q(s, a)$ estimating long-range expected value taking initial action a in state s . $Q(s, a)$ defines a new RL-based policy which then replaces the original policy.

Algorithm 1 envisions batch training, wherein for each observation (s_t, a_t, r_t) we compute in Line 7 a target Q -value and then regress the current $Q(s_t, a_t)$ values toward their targets. Our targets derive from the well-known Sarsa rule [8]: $\Delta Q(s_t, a_t) = \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$. Sarsa was needed for technical reasons in our specific application, as detailed below, but in other applications it may well be possible to use Q-Learning instead. The batch training details will depend on the function approximator used. In some cases one can do incremental training at each observation; in other cases the entire batch may be needed, e.g., to construct a regression tree. For our problem, we use a standard direct gradient method to train neural net weights, which works well in practice but carries a theoretical risk of divergence. In this case, we could use instead a residual gradient method [9], which guarantees convergence to local Bellman error minima. Typically the batch training will proceed for some number of sweeps through the training set until some error criterion (e.g., SSE) reaches an asymptotic value.

Algorithm 1. Hybrid RL procedure

```

1: Add exploration mechanism to initial policy if necessary
2: Run initial policy and record  $\{(s_t, a_t, r_t), 0 \leq t \leq T\}$ 
3: Initialize  $Q$ -function approximator (e.g. randomly)
4: repeat
5:    $SSE \leftarrow 0$  {sum squared error}
6:   for all  $t$  such that  $0 \leq t < T$  do
7:      $target \leftarrow r_t + \gamma Q(s_{t+1}, a_{t+1})$ 
8:      $error \leftarrow target - Q(s_t, a_t)$ 
9:      $SSE \leftarrow SSE + error \cdot error$ 
10:    Train  $Q(s_t, a_t)$  towards  $target$ 
11:   end for
12: until CONVERGED( $SSE$ )

```

3.1 Experimental Implementation

We could implement hybrid RL at the global Arbiter level, but the global state space scales exponentially with the number of applications. Hence we adopt the decompositional approach of [2] in which RL was implemented separately within each Trade3 application, using only local state and local number of allocated servers. This scales to many applications and worked well empirically despite lacking convergence guarantees.

The separation of learning from decision-making necessitates an on-policy algorithm like Sarsa in Line 7 of Algorithm 1: local Q-Learning’s $\max_a Q(s_{t+1}, a)$ term incorrectly assumes at the next time step that each application gets all the resources.

In each application, the action a_t comprises the local number of servers n_t allocated at time t . To represent the state s_t , many sensor readings could be used, but for simplicity we follow [2] in using only the current demand λ_t . We are also particularly interested in the dynamic consequences of allocation decisions. For example, when a server is added there may be initial transient suboptimal performance, or switching delays, during which the server is unavailable. To handle such effects, we employ a “delay-aware” representation that adds the previous allocation n_{t-1} to the state representation at time t . As long as such effects last no more than one allocation interval, this should suffice to learn the impact on expected value (longer delays would require n_{t-2} , etc.).

We represent the Q -function $Q(\lambda_t, n_{t-1}, n_t)$ with neural networks, due to their robust high-dimensional generalization and prior RL application successes. In each Trade3 application, we train by backpropagation a standard MLP with three input units, 12 sigmoidal hidden units, and one linear output unit. We scale the inputs to the interval $[0, 1]$, and using a learning rate of 0.005, 10-20 thousand sweeps through the dataset suffice to converge empirically. We set the discount parameter $\gamma = 0.5$.

4 Results

We first present results for open-loop and closed-loop systems without switching delays. The performance measure is total SLA revenue per allocation decision summed over all three applications. Each data point represents performance in a 12-hour run using a specific demand time series in each Trade3 that is repeated in all of the experiments. For a variety of initial model-based policies, we compare the initial policy performance with that of its corresponding hybrid RL trained policy.

The open-loop and closed-loop results are shown in Fig. 2. The percentage figures denote relative improvement of hybrid RL policies over their corresponding initial queuing models. (For the random initial policies, such percentages are shown in brackets as they have dubious meaning in our opinion.) The error bars denote 95% confidence intervals for the reported values; this calculation does not reflect the nearly identical demand traces used in each experiment. To address this factor we also performed paired T-test when comparing the hybrid RL results with each corresponding initial policy.

In the open-loop case we examine three initial policies: our open-loop models with and without exponentially smoothed parameter estimates [6], and for a baseline comparison, a uniform random allocation policy. We see substantial improvement of hybrid RL over each corresponding initial policy in both relative and absolute terms. In all pairs of experiments, a paired T-test rejects the null hypothesis that there is no difference between the performance means at 1% significance level with P-value $\leq 10^{-6}$.

In the closed-loop case we again examine a random initial policy, plus four different queuing model policies whose details are described in [6]. Once again we find substantial improvement of hybrid RL over each initial policy. The improvement over the random policy is enormous, while improvement over the queuing models is consistently at a double-digit percentage level with high statistical significance (rejected each null hypothesis using paired T-test at 1% significance level with P-value $\leq 4 \times 10^{-3}$).

Fig. 2. (a)Performance of policies in open-loop zero-delay scenario. (b) Performance of policies in closed-loop zero-delay scenario. (The random policy performance lies off the scale at -23.0.).

We also note that, in replicating the online method of [2], we obtained results $\sim$ 5-10% worse than our best queuing models, so that hybrid RL also outperforms online RL. This may reflect difficulty in scaling online RL to larger state/action spaces.

Fig. 3. Comparison of delay=4.5 sec with delay=0 results in open-loop and closed-loop scenarios

Fig. 3 compares our zero-delay results, using our best open-loop and closed-loop queuing models, with corresponding experiments that impose a delay of 4.5 seconds when a server is reassigned to a different application. The delay is asymmetric in that the server is immediately unavailable to the old application, but does not become available to the new application until 4.5 seconds have elapsed. We chose the delay to be a huge fraction of the five second allocation interval so that its empirical effects would be as clear as possible. Modeling of large delays is also of practical importance, since reallocation of servers in real Data Centers could entail several minutes of downtime. Fig. 3 shows that imposing this delay does in fact substantially harm the average performance in all cases. However, the amount of policy improvement of hybrid RL over its initial policy increases in both absolute and relative terms. In the open-loop scenario

the improvement increases from 10.0% to 16.4%, while in the closed-loop scenario the improvement jumps from 11.9% to 27.9%.

5 Insights into Hybrid RL Outperformance

We offer three insights as to how hybrid RL is able to outperform the initial queuing model policies. The first has to do with estimation bias. For reasons too technical to detail here, the queuing model policies end up having a bias toward overprovisioning, due to interactions of their response time estimates with the nonlinear SLA function. However, the RL nets, by learning to estimate utility directly, are able to achieve less biased estimation errors. This leads to the Trade3 applications receiving slightly fewer servers on average, with a slight loss of Trade3 utility, but the loss is more than made up by substantially greater Batch utility.

The second point is that our RL nets are able to properly treat transients and switching delays, unlike the steady-state queuing models. The learned policies exhibit hysteresis, a tendency to prefer steady allocations over switching based on instantaneous state. Some evidence for this is seen in Table 1, which exhibits basic statistics averaged over the two Trade3 applications $T1$ and $T2$ from the eight experiments shown in Fig. 3. The quantity $\langle n_T \rangle = (\langle n_{T1} \rangle + \langle n_{T2} \rangle)/2$ is the average number of assigned servers, while $\langle \delta n_T \rangle = (\langle \delta n_{T1} \rangle + \langle \delta n_{T2} \rangle)/2$ is the RMS change in number of assigned servers from one time step to the next. We see that $\langle n_T \rangle$ is slightly less for the RL nets than for the queuing models, and there is a further slight reduction for the RL nets for 4.5 sec delay compared to zero delay. More importantly, the $\langle \delta n_T \rangle$ statistics reveal noticeably less server swapping when using RL nets compared to queuing models, with the effect becoming quite pronounced ($\sim 50\%$ reduction) in the 4.5 sec delay case.

Table 1. Mean number of servers $\langle n_T \rangle$ assigned to a Trade3 application, and mean change in number of assigned servers $\langle \delta n_T \rangle$ per time step, in the eight experiments plotted in Fig. 3

Experiment	$\langle n_T \rangle$	$\langle \delta n_T \rangle$
Open-loop Delay=0 QM	2.27	0.578
Open-loop Delay=0 RL	2.04	0.464
Open-loop Delay=4.5 QM	2.31	0.581
Open-loop Delay=4.5 RL	1.86	0.269
Closed-loop Delay=0 QM	2.38	0.654
Closed-loop Delay=0 RL	2.24	0.486
Closed-loop Delay=4.5 QM	2.36	0.736
Closed-loop Delay=4.5 RL	1.95	0.331

The reduction in $\langle \delta n_T \rangle$ generally reflects hysteresis in the RL policies, and relates to our third insight, that RL policies exhibit greatly reduced thrashing. In experiments with 4.5 sec delay, massive thrashing under high load is a significant problem using the queuing model policies. An example of this is given in Fig. 4, which shows a five-minute interval in which $T1$ is moderately loaded and $T2$ is very heavily loaded. The

Fig. 4. Reduction of thrashing using Hybrid RL in closed-loop, 4.5 sec delay experiment

left plot shows the queuing model allocations, while the right plot shows the hybrid RL allocations under the same demand trace. We see much steadier allocations in the latter case. This is due partly to the RL value functions’ general preference to have no more than five servers, and partly to their projected high switching cost which inhibits a large instantaneous increase in servers (say, from 2 to 5-6) within an application.

6 Conclusions

Our hybrid RL approach neatly takes advantage of RL’s ability to learn in a knowledge-free manner, without an explicit system model or traffic model, and requiring little or no domain knowledge built into its state-space and value-function representations. However, our approach can exploit any available knowledge contained in an external policy, without having to interface to such knowledge. Moreover, using a simple “delay-aware representation” including the previous allocation decision, our approach also naturally handles transients and switching delays, which are dynamic consequences of reallocation lying outside the scope of traditional steady-state queuing models. On the other hand, hybrid RL also exploits the ability of model-based policies to achieve decent performance levels within a given system. This maintains acceptable performance while gathering training data, and avoids poor live performance expected in using online RL. We may also exploit robustness of model-based policies under various types of system changes, e.g. hardware upgrades or changes in the SLA, which require retraining of the RL value functions. When such changes occur, we can fall back on the model-based policy to deliver acceptable performance while accumulating new training data.

Hybrid RL may have wide applicability in many other areas of systems management. The most promising applications would exhibit: (a) a tractable state-space representation; (b) frequent online decision making depending upon time-varying system state; (c) frequent observation of numerical rewards in an immediate or moderately delayed relation to management actions; (d) pre-existing policies that obtain acceptable performance levels. Many applications clearly have such properties: among them are dynamic allocation of other types of resources, e.g., bandwidth, memory, LPARs, etc. We would also include performance-based online tuning of system control parameters, such as web server, OS or DB parameters. Finally, hybrid RL could conceivably encompass simultaneous management to multiple criteria (e.g. performance and availability), as long as the rewards pertaining to each criterion are equivalently scaled.

In future work we plan further study of the scalability of hybrid RL/function approximation to larger state spaces. We will also study whether further performance improvements can be obtained via multiple iterations of hybrid RL. Our testbed may also provide an interesting test of novel theoretical research on learning good policies without explicit exploration [10], and on exploration-exploitation tradeoffs.

References

1. Das, R., Tesauro, G., Walsh, W.E.: Model-based and model-free approaches to autonomic resource allocation. Technical Report RC23802, IBM Research (2005)
2. Tesauro, G.: Online resource allocation using decompositional reinforcement learning. In: Proc. of AAAI-05. (2005)
3. Vengerov, D., Iakovlev, N.: A reinforcement learning framework for dynamic resource allocation: First results. In: Proc. of ICAC-05. (2005)
4. Price, B., Boutilier, C.: Accelerating reinforcement learning through implicit imitation. *J. of AI Research* **19** (2003) 569–629
5. Lavenberg, S.S.: Personal communication (2006)
6. Tesauro, G., Jong, N.K., Das, R., Bennani, M.N.: A hybrid reinforcement learning approach to autonomic resource allocation. In: Proc. of ICAC-06. (2006) 65–73
7. Squillante, M.S., Yao, D.D., Zhang, L.: Internet traffic: Periodicity, tail behavior and performance implications. In Gelenbe, E., ed.: *System Performance Evaluation: Methodologies and Applications*, CRC Press (1999)
8. Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*. MIT Press (1998)
9. Baird, L.: Residual algorithms: Reinforcement learning with function approximation. In: Proc. of ICML-95. (1995)
10. Abbeel, P., Ng, A.Y.: Exploration and apprenticeship learning in reinforcement learning. In: Proc. of ICML-05. (2005)