Learning Caching Policies with Subsampling

Haonan Wang, Hao He, Mohammad Alizadeh, Hongzi Mao
MIT Computer Science and Artificial Intelligence Laboratory
{haonanw,haohe,alizadeh,hongzi}@mit.edu

Abstract

We concern applying reinforcement learning (RL) on caching systems. The main challenge of this problem is the excessively long MDP time horizon, since a practical cache can host millions of objects and the reward for each caching action can delay for millions of steps on average — such horizon is at least $100\times$ longer than state-of-the-art RL applications such as game of Go or Dota 2. We purpose to use a subsampling technique to shorten the MDP horizon by hashing the objects in a trace and reducing the cache size. We also develop an accompanying theory that describes how the MDP reduction technique can be used for more general applications. In our experiment, we show that the subsampling technique can generalize the learned RL policy to $100\times$ larger caches.

1 Introduction

Reinforcement learning (RL) has recently been used to train powerful controllers in many computer systems, including datacenter scheduling [1, 11, 12, 15, 16], database query optimization [10, 14], network congestion control [7, 8], routing [23] and more. We concern RL in the context of caching [21]. Computer systems with hierarchical hardware structures (e.g., fast memory and slow disks) often deploy caching mechanisms to prioritize the limited memory space among different objects. In this setting, an RL agent observes the information of an incoming object and the existing objects in the cache; it then decides whether to admit the incoming object and which objects to evict if necessary. The goal is to maximize the object hit rate so that more objects are retrieved from the fast memory.

In many system applications, the optimal caching policy depends on the characteristic of different workloads. For example, a CDN caching policy may set a lower threshold on the object size for admission in the daytime (which primarily serves text-based website traffic) than during the night (which serves mostly video streaming traffic) [4, §3.2]. As a result, recent work has proposed to learn caching policies using imitation learning on workload-specific optimal policies calculated offline [3].

Despite being a good fit conceptually (prior work has used tabular Q-learning for a DRAM caching problem [6]), modern RL-based caching systems have not yet been competitive with existing heuristics [3, §1]. The main obstacle is the long horizon of MDP in caching problems. For example, the size of a typical CDN cache is in the scale of 1GB and can host up to millions of objects. Since the feedback for an caching action only shows up until the corresponding object is retrieved again, the average delay for each reward is in millions of MDP steps. As a comparison, state-of-the-art RL methods deal with game of Go [19, 20] in hundreds of steps (placing 19×19 stones) and Dota 2 in about $80{,}000$ time steps [18] — the horizon lengths for practical caching problems are at least $100\times$ longer. For long horizon RL problems, it is well known that reward credit assignment becomes difficult and prohibits training strong policies [22].

Fortunately, caching problems exhibit a unique structure that allows significant reduction in the MDP horizon length. The key observation is that the relative object hit ratio (i.e., cache hit normalized to the total object count) remains the same if we subsample the objects in a trace (by hashing on the object IDs) and accordingly reduce the cache size [2, 9]. The subsampled caching environment effectively

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

results in an MDP with shorter time horizon. We therefore train the RL agent on the reduced caching problem and deploy the learned policy on the original setting (§3.1). More broadly, we develop a theoretical framework (§3.2) for more general RL applications that can leverage subsampling to reduce the horizon length and subsequently improve training efficiency. In our experiments with a CDN cache (§4), subsampling enables RL to learn strong cache admission policies that rival or outperform state-of-the-art heuristics and the learned policies are able to generalize to $100\times$ larger caches.

2 Problem Setup

For concreteness, we focus on the cache admission setting. Each RL episode contains a series of objects continuously arriving over time. The caching environment invokes the RL agent when an incoming object is missing from the cache. Specifically, at each step t for an missing object, the agent observes a feature vector $[s_t, c_t, f_t, z_t]$, where s_t is the size of incoming object, c_t is the steps since the same object visited last time,[1] f_t is the object's number of visits so far, z_t is the portion of remaining cache size. The agent then takes a binary action $a_t \in \{0,1\}$ to decide whether or not to admit the incoming object. To make room for the new object, the system environment evicts the least recently used objects from the cache. The reward r_t at each step is the total byte hits since that last action a_{t-1}. Thus, the sum of the rewards amounts to the total byte hits for all objects in a trace. The detailed implementation of the caching environment is retrieved from the Park dataset [13].

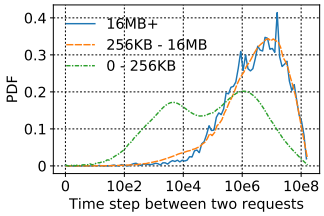


Figure 1: Inter-arrival time for objects with different sizes.

To quantify the MDP time horizon and the effect of delayed rewards, Figure 1 depicts the distribution of inter-arrival time for objects with different sizes. As shown, more than 30% of the objects take at least 10^6 steps between two consecutive visits.

3 Methodology

In this section, we first describe the subsampling procedure tailored for the caching application. We then empirically evaluate the generalizability of the learned caching policy on larger caches with longer trace. Finally, we develop a more general theory that allows MDPs with certain properties to reduce their state space and time horizon while keeping the optimal policy generalizable.

3.1 Training Caching Policies with Subsampling

We subsample the objects from the trace with a random hash on object IDs with a ratio $1/K$. It is important to notice that the random hash maintains the statistics of the trace, such as the object size distribution and the byte hit rate under the same policy. By contrast, randomly omitting the objects from the trace would cause shifts for important statistics such as the visiting counts of an object (which is a feature observed by the agent as described in §2). Meanwhile, we reduce the cache size by the subsampling factor K. Under this construction, the input features to the agent are scale invariant, since the observations are either independent of the trace length (e.g., object size) or normalized by the cache size (e.g., remaining cache portion). Finally, we perform standard advantage actor critic (A2C) [17] training on the reduced caching setting and evaluate the learned policy on the original cache and trace.

3.2 Theoratical Analysis

In general, our strategy of reducing the RL time horizon is to reduce the MDP state space. Intuitively, if an agent only takes actions for 1% of the states (the rest decisions are made by some default policy), the agent is effectively trained on a smaller MDP with its horizon shorten to 1%. Importantly, we need to construct the smaller MDP with an "aggregated" state transition and reward function from the original MDP. This builds a connection between the horizon reduction and state space subsampling. By choosing different parts of the state space at each episode, we can cover all the states for the agent to generalize to the original setting. In the following, we first formally describe the state reduction strategy and then remark on how it matches with our subsampling approach for the caching problem.

[1] If an object was never visited before, we set a large number on its recency feature $c_t = 1000$.

Notations. An MDP $\mathcal{M}=(\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R})$ is consist of four parts: states $\mathcal{S}$, actions $\mathcal{A}$, state transition probability $\mathcal{P}:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\to\mathbb{R}$ and immedite reward $\mathcal{R}:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\to\mathbb{R}$. $\mathcal{P}(s,a,s')$ is the probability of transitioning to state s' after taking action a at state s, i.e $\mathbb{P}(s_{t+1}=s'|s_t=s,a_t=a)$. For a policy π, we define the total reward $G(\pi;\mathcal{M})\triangleq\mathbb{E}_\pi[\sum_t\gamma^t\mathcal{R}(s_t,a_t,s_{t+1})]$ as the summation of rewards at every step discounted by a factor $\gamma<1$. The goal of reinforcement learning is to find an optimal policy π^* that maximize the total reward, i.e. $\pi^*=\operatorname{argmax}_\pi G(\pi;\mathcal{M})$.

Definition 1 (Constrained traversal event)**.** *Given two states $s,s'\in\mathcal{S}$, an action $a\in\mathcal{A}$, a subset of states $\tilde{\mathcal{S}}\subset\mathcal{S}$, at any step t, we define constrained traversal event $E_t(s,a,s';\tilde{\mathcal{S}})\triangleq\{s_1,\cdots,s_{t-1}\in\tilde{\mathcal{S}},s_0=s,a_0=a,s_t=s'\}$.*

These events describe all possible trajectories starting from state $s_0=s$, taking action $a_0=a$ and ending up with state $s_t=s'$, while *only* visiting the constrained set $\tilde{\mathcal{S}}$ in the intermediate steps.

Definition 2 (State-reduced MDP)**.** *Given an MDP $\mathcal{M}=(\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R})$, a subset of states $\mathcal{S}'\subset\mathcal{S}$ and a policy π, we define the state-reduced MDP $\phi(\mathcal{M},\mathcal{S}',\pi)=(\mathcal{S}',\mathcal{A},\mathcal{P}',\mathcal{R}')$ where the new transition probability $\mathcal{P}'(s,a,s')\triangleq\mathcal{P}(s,a,s')+\sum_{T=1}^\infty\mathbb{P}_\pi\big(E_T(s,a,s';\mathcal{S}\setminus\mathcal{S}')\big)$ and the new reward $\mathcal{R}'(s,a,s')\triangleq\mathcal{P}(s,a,s')\mathcal{R}(s,a,s')+\sum_{T=1}^\infty\mathbb{P}_\pi\big(E_T(s,a,s';\mathcal{S}\setminus\mathcal{S}')\big)\mathbb{E}_\pi\Big[\sum_{t=0}^{T-1}\gamma^t\mathcal{R}(s_t,a_t,s_{t+1})\Big|E_T(s,a,s';\mathcal{S}\setminus\mathcal{S}')\Big]$.*

Definition 3 (Locally optimal policy)**.** *A policy π is locally optimal for a subset of states $\mathcal{S}'\subset\mathcal{S}$ if it maximizes the total reward amongst all policies π' that differ with π only on states in $\mathcal{S}'$, i.e., $G(\pi;\mathcal{M})\geq G(\pi';\mathcal{M}),\forall\pi'$ s.t. $\forall s\in\mathcal{S}\setminus\mathcal{S}',\pi'(\cdot|s)=\pi(\cdot|s)$.*

By the definitions above, if a policy $\tilde{\pi}$ is locally optimal for $\mathcal{S}'$, it is the optimal policy for the state reduced MDP $\phi(\mathcal{M},\mathcal{S}',\tilde{\pi})$ since only changing π at states in $\mathcal{S}'$ can not improve the total reward. More formaly we have $\tilde{\pi}=\operatorname{argmax}_\pi G\big(\pi;\phi(\mathcal{M},\mathcal{S}',\tilde{\pi})\big)$.

Lemma 1. *The optimal policy π^* of the MDP $\mathcal{M}$ is locally optimal for any subset of the states, i.e. $\forall\mathcal{S}'\subset\mathcal{S}$, $\pi^*=\operatorname{argmax}_\pi G\big(\pi;\phi(\mathcal{M},\mathcal{S}',\pi^*)\big)$.*

Proof. If the optimal policy π^* is not locally optimal for the state subset $\mathcal{S}'$, then, by definition, there is a policy π' that differs with π^* at $\mathcal{S}'$ and has a better total reward than π^* which voliates π^*'s optimality. □

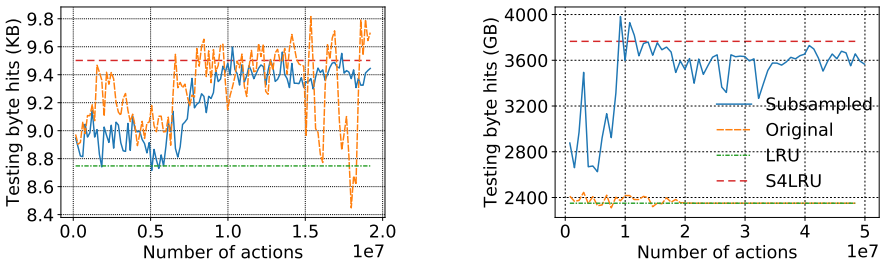
Theorem 1 (Fixed point)**.** *Let π^* be the optimal policy for the MDP $\mathcal{M}$. Consider a distribution q over $2^{\mathcal{S}}$ (all possible subsets of $\mathcal{S}$) and the policy iteration via $\pi^{(t+1)}=\operatorname{argmax}_\pi\mathbb{E}_{\mathcal{S}'\sim q}G\big(\pi;\phi(\mathcal{M},\mathcal{S}',\pi^{(t)})\big)$. Then π^* is a fixed point of the policy iteration under any distribution q, i.e. $\pi^*=\operatorname{argmax}_\pi\mathbb{E}_{\mathcal{S}'\sim q}G\big(\pi;\phi(\mathcal{M},\mathcal{S}',\pi^*)\big)$, $\forall q$.*

Proof. We define the following functions: total objective $f(\pi)\triangleq\mathbb{E}_{\mathcal{S}'\sim q}G\big(\pi;\phi(\mathcal{M},\mathcal{S}',\pi^*)\big)$ and individual objective $f_{\mathcal{S}'}(\pi)\triangleq q(\mathcal{S}')G(\pi;\phi(\mathcal{M},\mathcal{S}',\pi^*))$ for any $\mathcal{S}'$. By lemma 1, the optimal policy π^* maximizes every individual objective. Thus π^* also maximizes the summation of all individual objective, i.e. $\pi^*=\operatorname{argmax}_\pi f(\pi)=\operatorname{argmax}_\pi\mathbb{E}_{\mathcal{S}'\sim q}G\big(\pi;\phi(\mathcal{M},\mathcal{S}',\pi^*)\big)$. □

Theorem 2 (Uniqueness)**.** *If a policy $\tilde{\pi}$ is locally optimal for any subset of states, then $\tilde{\pi}$ is the optimal policy. Formally, if policy $\tilde{\pi}$ satisfies $\tilde{\pi}=\operatorname{argmax}_\pi G(\pi;\phi(\mathcal{M},\mathcal{S}',\tilde{\pi})),\forall\mathcal{S}'\subset\mathcal{S}$, then $\tilde{\pi}=\operatorname{argmax}_\pi G(\pi;\mathcal{M})$.*

Proof. Recall the Bellman equation $V^\pi(s)=\operatorname{argmax}_a\mathcal{R}(s,a)+\mathbb{E}_{s'\sim\mathcal{P}}V^\pi(s')$, where the value function $V^\pi(s)$ is the expected total reward starting from state s following policy π. The uniqueness of Bellman equation solution [22, §3.5] ensures that only the optimal policy π^* satisfies the equation above. Now consider any non-optimal policy $\tilde{\pi}$. It must violate the Bellman equation, i.e. $\exists\, s\in\mathcal{S}$ such that $V^{\tilde{\pi}}(s)<\operatorname{argmax}_a\mathcal{R}(s,a)+\mathbb{E}_{s'\sim\mathcal{P}}V^{\tilde{\pi}}(s')$. Thus, the policy π is not locally optimal for the state set $\mathcal{S}'=\{s\}$. This implies any non-optimal policy cannot satisfy $\tilde{\pi}=\operatorname{argmax}_\pi G(\pi;\phi(\mathcal{M},\mathcal{S}',\tilde{\pi})),\forall\mathcal{S}'\subset\mathcal{S}$, which completes the proof. □

Remark. In practice, if we cover a large amount of state-reduced MDPs (covering all states) and the learned policy can converge to the local optimal for these MDPs, Theorem 2 guarantees that learning in state-reduced MDPs converges to the optimal policy. However, in some cases we might only cover part of the state space when reducing the MDPs (e.g., too few samples during training). Nevertheless, Theorem 1 shows that even under such weak condition, we still guarantee that the optimal policy is always a fixed point of the optimization no matter which state-reduced MDPs distribution is used.



(a) Small cache (256 Byte, ∼200 objects). **(b)** Large cache (1 GB, ∼100K objects).

Figure 2: Learning curve of RL training on the subsampled and the original caching problem. The performance is evaluated with unseen part of a public CDN trace [3, 13] without subsampling or cache reduction.

We argue that our subsampling approach for the caching problem implicitly constructs the state-reduced MDP. Following Definition 2, hashing on object IDs samples a subset of objects for the state space (with a factor K). The transformation of the state transition probability and the reward function for the state-reduced MDP can be approximated as follows. We can view reducing the cache size as dispatching an incoming object to a small cache (with $1/K$ total cache size) if the object is sampled or to a large cache (with $(K-1)/K$ total cache size) otherwise. Suppose the caching policy is the same on both separated caches, then the state transition follows Definition 2 and the reward in the state-reduced MDP is roughly linearly scaled by $1/K$ from the original MDP.

4 Experiments

In our experiment, we first use a small cache (containing 200 objects on average) and a short trace (10k objects in total) to verify that the subsampling technique allows training an RL policy comparable to directly trained on the original problem. At this small scale, standard RL can directly train strong caching policy. For subsampling, we shrink the trace and cache size by 8. The learning curve in Figure 2a shows that the RL policy the on the reduced caching setting is similar to the RL performance trained directly on the original setting. Also, as expected, RL policy is empirically more robust in performance with the subsampling setting. Moreover, the learned policy is strong enough to rival or outperform existing cache admission heuristics such as S4LRU [5].

We then evaluate the subsampling technique on a more realistic setting, where a longer trace contains 100,000 objects and the cache size is 1GB (similar to a typical CDN cache). As shown in Figure 2b, the vanilla A2C approach cannot learn strong caching policy at this scale — the performance falls behind the existing heuristics and is similar to a random policy. However, by subsampling the trace and reducing the cache size by $100\times$, the RL agent can successfully train and generalize its policy to the original problem. Compared to other state-of-the-art heuristics, the generalized learned policy performs similarly on the average byte hit rate.

5 Conclusions and Discussions

The subsampling technique can significantly mitigate the long MDP horizon problem for caching application. In our experiment, the learned RL policy on subsampled caching environment is able to generalize to caches with at least $100\times$ larger sizes. We also develop a general theory that describe how the MDP reduction allows policy iteration to converge to the original optimal point.

We are currently exploring similar techniques to a wider family of caching policies, including learning the eviction policy. Also, the subsampling technique can generally apply to system problems with queuing structures. For example, subsampling the jobs in a trace and reduce the number of servers may help improving the RL training efficiency for load balancing and job scheduling applications [13].

In our current theorems, we only describe the optimal policy invariance for MDPs following the general state-reduction process. More broadly, we are looking for a systematic way of discovering RL applications beyond queuing systems that allow state reduction in this framework. For these problems, it is interesting to apply some form of subsampling (or other approach to reduce MDP horizon) to improve RL training efficiency.

Acknowledgments. We thank the anonymous NeurIPS reviewers for their constructive feedback. This work was funded in part by the NSF grants CNS-1751009, CNS-1617702, a Google Faculty Research Award, an AWS Machine Learning Research Award, a Cisco Research Center Award, an Alfred P. Sloan Research Fellowship, and sponsors of the MIT DSAIL lab.

References

[1] R. Addanki, S. B. Venkatakrishnan, S. Gupta, H. Mao, and M. Alizadeh. "Placeto: Efficient Progressive Device Placement Optimization". In: *NIPS Machine Learning for Systems Workshop*. 2018.

[2] N. Beckmann and D. Sanchez. "Talus: A simple way to remove cliffs in cache performance". In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2015, pp. 64–75.

[3] D. S. Berger. "Towards Lightweight and Robust Machine Learning for CDN Caching". In: *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. ACM. 2018, pp. 134–140.

[4] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter. "AdaptSize: Orchestrating the hot object memory cache in a content delivery network". In: *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 2017, pp. 483–498.

[5] Q. Huang, K. Birman, R. Van Renesse, W. Lloyd, S. Kumar, and H. C. Li. "An analysis of Facebook photo caching". In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM. 2013, pp. 167–181.

[6] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana. "Self-optimizing memory controllers: A reinforcement learning approach". In: *ACM SIGARCH Computer Architecture News*. Vol. 36. 3. IEEE Computer Society. 2008, pp. 39–50.

[7] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar. "A Deep Reinforcement Learning Perspective on Internet Congestion Control". In: *International Conference on Machine Learning*. 2019, pp. 3050–3059.

[8] N. Jay, N. H. Rotman, P Godfrey, M. Schapira, and A. Tamar. "Internet congestion control via deep reinforcement learning". In: *arXiv preprint arXiv:1810.03259* (2018).

[9] R. E. Kessler, M. D. Hill, and D. A. Wood. "A comparison of trace-sampling techniques for multi-megabyte caches". In: *IEEE Transactions on Computers* 43.6 (1994), pp. 664–675.

[10] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein, and I. Stoica. "Learning to optimize join queries with deep reinforcement learning". In: *arXiv preprint arXiv:1808.03196* (2018).

[11] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. "Resource Management with Deep Reinforcement Learning". In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets)*. Atlanta, GA, 2016.

[12] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh. "Learning Scheduling Algorithms for Data Processing Clusters". In: *Proceedings of the 2019 ACM SIGCOMM Conference (SIGCOMM)*. Aug. 2019.

[13] H. Mao, P. Negi, A. Narayan, H. Wang, J. Yang, H. Wang, R. Marcus, et al. "Park: An Open Platform for Learning Augmented Computer Systems". In: *Advances in Neural Information Processing Systems* (2019).

[14] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, et al. "Neo: A Learned Query Optimizer". In: *arXiv preprint arXiv:1904.03711* (2019).

[15] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, and J. Dean. "A Hierarchical Model for Device Placement". In: *International Conference on Learning Representations*. 2018.

[16] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, et al. "Device Placement Optimization with Reinforcement Learning". In: *Proceedings of The 33rd International Conference on Machine Learning*. 2017.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, et al. "Asynchronous methods for deep reinforcement learning". In: *Proceedings of the International Conference on Machine Learning*. 2016, pp. 1928–1937.

[18] OpenAI. *OpenAI Five*. https://blog.openai.com/openai-five/. 2018.

[19] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, et al. "Mastering the game of go without human knowledge". In: *Nature* 550.7676 (2017), p. 354.

[20] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[21] A. J. Smith. "Cache memories". In: *ACM Computing Surveys (CSUR)* 14.3 (1982), pp. 473–530.

[22] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction, Second Edition*. MIT Press, 2017.

[23] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar. "Learning to route with deep rl". In: *NIPS Deep Reinforcement Learning Symposium*. 2017.