AsyncQVI: Asynchronous Randomized Q-Value Iteration For Reinforcement Learning

Fei Feng

joint work with Yibo Zeng and Wotao Yin

INFORMS 2019
Motivation: Accelerate Learning

Figure: Paradigm for RL algorithms with a single computing agent.
Motivation: Accelerate Learning

Figure: Paradigm for RL algorithms with a single computing agent.

Figure: Paradigm for RL algorithms with multiple computing agents.
Technique: Asynchronous Parallel

Sync-parallel:
- probably long idle time;
- little tolerant to communication glitches;
- keeps information consistent.

Async-parallel:
- saves idle time
- more tolerant to communication glitches;
- easier to incorporate new agents;
- information is delayed or inconsistent.

Figure: Pictures from Peng et al. 2016
Challenges and Solution

Error sources:
- randomization
- delayed and inconsistent information

Assumptions:
- delay is uniformly bounded by $B_1$
- the time interval between consecutive updates for each coordinate is uniformly bounded by $B_2$
Challenges and Solution

Error sources:
- randomization
- delayed and inconsistent information

Assumptions:
- delay is uniformly bounded by $B_1$.
- the time interval between consecutive updates for each coordinate is uniformly bounded by $B_2$.
- a generative model.

Fei Feng joint work with Yibo Zeng and Wotao Yin
AsyncQVI: Asynchronous Randomized Q-Value Iteration
Challenges and Solution

Error sources:
- randomization
- delayed and inconsistent information

Assumptions:
- delay is uniformly bounded by $B_1$.
- the time interval between consecutive updates for each coordinate is uniformly bounded by $B_2$.
- a generative model.

Leverage: The contraction property of the Bellman operator.
Key Idea:

Q-value iteration:

\[ Q(s,a)(t+1) = \sum_s p(a|s,s')r_{a|s,s'} + \gamma \sum_s p(a|s,s')\max_{a'} Q(s',a')(t), \forall s,a \]

Revise the former step to coordinate update:

\[ Q(s,a)(t+1) = \{ \sum_s p(a|s,s')r_{a|s,s'} + \gamma \sum_s p(a|s,s')\max_{a'} Q(s',a'), (s_{t+1},a_{t+1}) \}; Q(s,a)(t), \text{o.w.} \]

Implement in an asynchronous parallel manner:

\[ Q(s,a)(t+1) = \{ \sum_s p(a|s,s')r_{a|s,s'} + \gamma \sum_s p(a|s,s')\max_{a'} \hat{Q}(s',a'), (s_{t+1},a_{t+1}) \}; Q(s,a)(t), \text{o.w.} \]

Further revise to a randomized fashion with active sampling:

\[ Q(s,a)(t+1) = \{ \frac{1}{K} \sum r_k + \gamma \frac{1}{K} \sum \max_{a'} \hat{Q}(s',a') - (1 - \gamma) \epsilon, (s_{t+1},a_{t+1}) \}; Q(s,a)(t), \text{o.w.} \]

Fei Feng joint work with Yibo Zeng and Wotao Yin

AsyncQVI: Asynchronous Randomized Q-Value Iteration
Key Idea:

Q-value iteration:

\[ Q_{s,a}(t + 1) = \sum_{s'} p_{ss'}^a r_{ss'}^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} Q_{s',a'}(t), \quad \forall \ s, a \]
Key Idea:

1. Q-value iteration:

\[ Q_{s,a}(t + 1) = \sum_{s'} p_{ss'}^a r_{ss'}^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} Q_{s',a'}(t), \quad \forall \ s, a \]

2. Revise the former step to coordinate update

\[ Q_{s,a}(t + 1) = \begin{cases} 
\sum_{s'} p_{ss'}^a r_{ss'}^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} Q(t)_{s',a'}, & (s_{t+1}, a_{t+1}); \\
Q_{s,a}(t), & \text{o.w.} 
\end{cases} \]
Key Idea:

1. **Q-value iteration:**
   
   \[ Q_{s,a}(t+1) = \sum_{s'} p_{ss'}^a r_{ss'}^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} Q_{s',a'}(t), \quad \forall \ s, a \]

2. Revise the former step to coordinate update
   
   \[ Q_{s,a}(t+1) = \begin{cases} 
   \sum_{s'} p_{ss'}^a r_{ss'}^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} Q(t)_{s',a'}, & (s_{t+1}, a_{t+1}); \\
   Q_{s,a}(t), & \text{o.w.} 
   \end{cases} \]

3. Implement in an asynchronous parallel manner:
   
   \[ Q_{s,a}(t+1) = \begin{cases} 
   \sum_{s'} p_{ss'}^a r_{ss'}^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} \hat{Q}_{s',a'}, & (s_{t+1}, a_{t+1}); \\
   Q_{s,a}(t), & \text{o.w.} 
   \end{cases} \]
Key Idea:

1. Q-value iteration:
   \[ Q_{s,a}(t + 1) = \sum_{s'} p_{ss'}^a r_{ss'}^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} Q_{s',a'}(t), \quad \forall \ s, a \]

2. Revise the former step to coordinate update
   \[ Q_{s,a}(t + 1) = \begin{cases} \sum_{s'} p_{ss'}^a r_{ss'}^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} Q(t)_{s',a'}, & (s_{t+1}, a_{t+1}); \\
   Q_{s,a}(t), & \text{o.w.} \end{cases} \]

3. Implement in an asynchronous parallel manner:
   \[ Q_{s,a}(t + 1) = \begin{cases} \sum_{s'} p_{ss'}^a r_{ss'}^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} \hat{Q}_{s',a'}, & (s_{t+1}, a_{t+1}); \\
   Q_{s,a}(t), & \text{o.w.} \end{cases} \]

4. Further revise to a randomized fashion with active sampling:
   \[ Q_{s,a}(t + 1) = \begin{cases} \frac{1}{K} \sum_k r_k + \gamma \frac{1}{K} \sum_k \max_{a'} \hat{Q}_{s_k,a'} - \frac{(1-\gamma)\varepsilon}{4}, & (s_{t+1}, a_{t+1}); \\
   Q_{s,a}(t), & \text{o.w.} \end{cases} \]
**Algorithm 1: AsyncQVI: Asynchronous-Parallel Q-value Iteration**

**Input:** \( \varepsilon \in (0, (1 - \gamma)^{-1}) \), \( \delta \in (0, 1) \), \( L, K \);

**Shared variables:** \( v \leftarrow 0 \), \( \pi \leftarrow 0 \), \( t \leftarrow 0 \);

**Private variables:** \( \hat{v}, r, S, q \);

while \( t < L \), every agent *asynchronously* do

- select state \( i_t \in S \) and action \( a_t \in A \);
- copy shared variable to local memory \( \hat{v} \leftarrow v \);
- call \( GM(s_t, a_t) \) \( K \) times and collect samples \( \{s'_1, \ldots, s'_K\} \) and \( r_1, \ldots, r_K \);
- \( q \leftarrow \frac{1}{K} \sum_{k=1}^{K} r_k + \gamma \frac{1}{K} \sum_{k=1}^{K} \hat{v}_{s'_k} - \frac{(1-\gamma)\varepsilon}{4} \);
- if \( q > v_{i_t} \) then
  - mutex lock;
  - \( v_{i_t} \leftarrow q \), \( \pi_{i_t} \leftarrow a_t \);
  - mutex unlock;
- increment the global counter \( t \leftarrow t + 1 \);

return \( \pi \)
Theorem 1 (Zeng, Feng, and Yin 2018)

Under the assumptions, AsyncQVI returns an \( \varepsilon \)-optimal policy \( \pi \) with probability at least \( 1 - \delta \) at the sample complexity

\[
\tilde{O}\left(\frac{B_1 + B_2}{(1 - \gamma)^5 \varepsilon^2 \log(1/\delta)}\right).
\]

In [Azar, Munos, and Kappen 2013], it shows that the optimal sample complexity with a generative model is:

\[
O\left(\frac{|S||A|}{(1 - \gamma)^3 \varepsilon^2 \log(\frac{|S||A|}{\delta})}\right).
\]
## Related Algorithms

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Async</th>
<th>Sample Complexity</th>
<th>Memory</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variance-Reduced VI</td>
<td>×</td>
<td>$\tilde{O}\left(\frac{</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Variance-Reduced QVI</td>
<td>×</td>
<td>$\tilde{O}\left(\frac{</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Async Q-learning</td>
<td>✓</td>
<td>−</td>
<td>$O(</td>
<td>S</td>
</tr>
<tr>
<td>AsyncQVI</td>
<td>✓</td>
<td>$\tilde{O}\left(\frac{</td>
<td>S</td>
<td></td>
</tr>
</tbody>
</table>
**Numerical Test: Sailing Problem**

**Figure: Parallel speedup**

**Figure:** Performance with 20 parallel threads and different noises.
Conclusion and Future Work

Conclusion:
- We propose an asynchronous algorithm AsyncQVI for RL with explicit sample complexity.
- AsyncQVI trades a little more samples for less time and memory.
- AsyncQVI has linear parallel speedup empirically.

Future work:
- Add variance reduction trick to achieve a better sample complexity result;
- Relax generative model to exploration policy.
Conclusion and Future Work

**Conclusion:**
- We propose an asynchronous algorithm AsyncQVI for RL with explicit sample complexity.
- AsyncQVI trades a little more samples for less time and memory.
- AsyncQVI has linear parallel speedup empirically.

**Future work:**
- Add variance reduction trick to achieve a better sample complexity result;
- Relax generative model to exploration policy.
Thank you!
References


