Synchronous Value Iteration for Infinite Horizon

Divya Bajaj
bajajd@oregonstate.edu

Tyler Lawson
lawsonty@oregonstate.edu

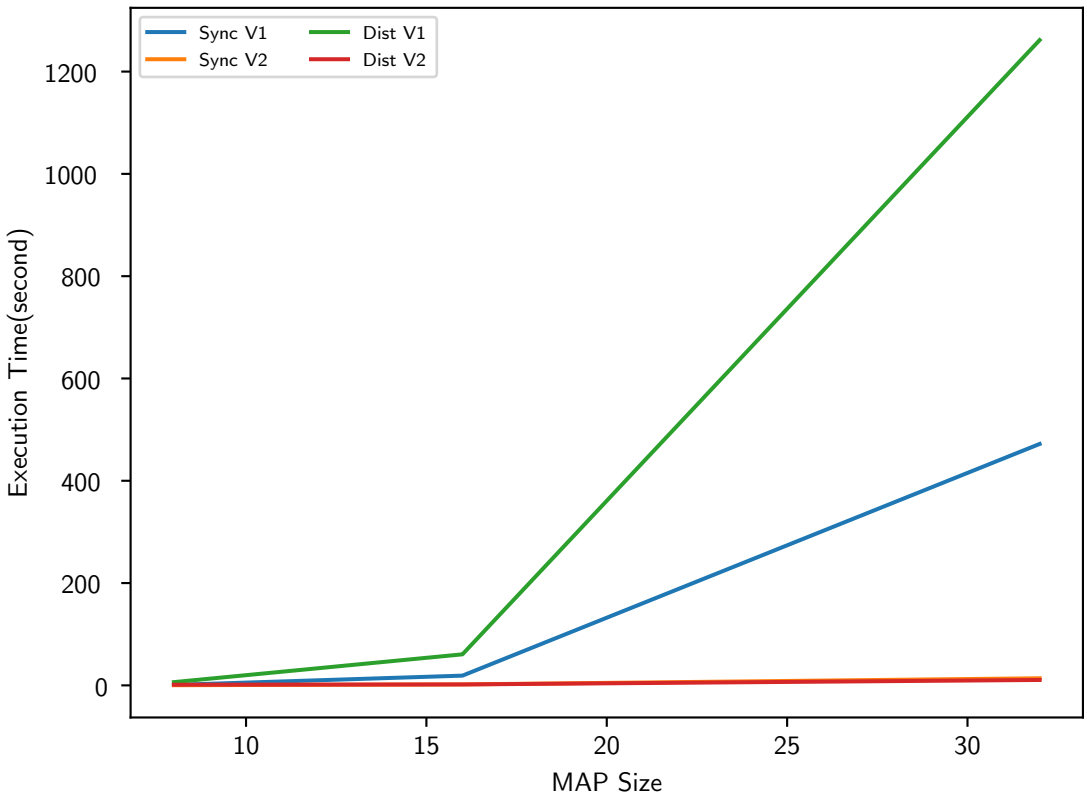
April 21, 2020

1 Introduction

Synchronous value iteration algorithm is an iterative algorithm that finds an optimal value function V^* for each state in a given Markov Decision Process model. The MDP model given in this assignment is the Frozen Lake environment. The algorithm was implemented using two non-distributed and two distributed approaches. Section 2 explains how different map sizes effect the execution time of all the different approaches. Section 3 explains how increasing the number of workers in the distributed approaches effect their execution time. Section 4 provides a comparison between the two distributed approaches. Finally, Section 5 compares the best distributed programming approach with the best non-distributed programming approach in terms of their execution time.

2 Impact of Map Size on Execution Time

Approach	8 X 8	16 X 16	32 X 32
Synchronous V_1	0.8688204288482666	19.028144359588623	471.98448061943054
Synchronous V_2	0.29715871810913086	1.8688197135925293	13.882546663284302
Distributed V_1	6.340137481689453	60.673938274383545	1261.5731217861176
Distributed V_2	1.306192398071289	1.7891209125518799	10.558962106704712



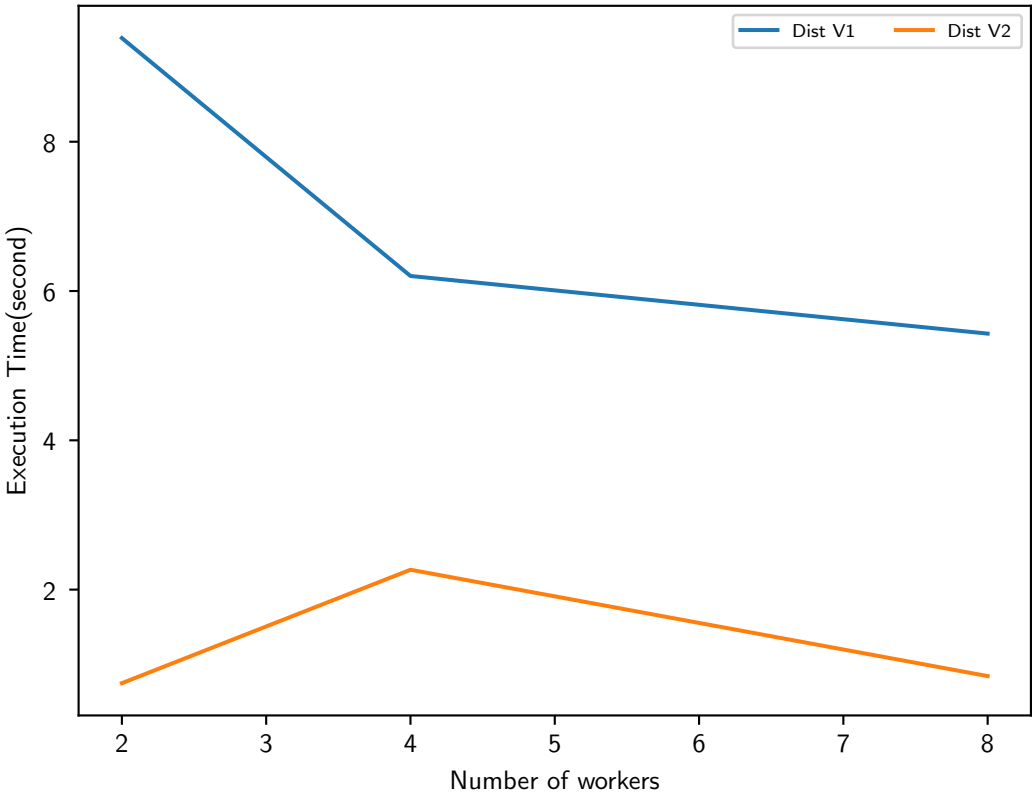
Increase in map size implies increase in the number of states. A grid map of size 8 X 8 has lesser states than that of size 16 X 16. Thus, with increase in number of states the computation takes longer time, thus, increasing the execution time.

The execution time for Non-distributed approach in version 1 is greater than that in version 2 for each map size because the time complexity of version 1 is quadratic in the number of states. While, in version 2 only the successor states are used in computation of the Value function for the state. This approach of using only successor states for computation resulted in a much more efficient implementation.

The distributed approach in version 2 divides states into batches among workers. While in version 1, each worker computes value function of only one state at a time. This approach of assigning a batch of states to a worker reduces the wait time between different states as the update for a complete batch of states can happen at once. In contrast, each worker in version 1 has to wait for another worker to update one single state. This approach of assigning a batch of states to a worker has also turned out to be very useful in terms of efficiency.

3 Impact of Number of Workers on Running time

Approach	2	4	8
Distributed V_1	9.388800859451294	6.2014594078063965	5.429482460021973
Distributed V_2	0.7465131282806396	2.2660014629364014	0.8415665626525879



The first version of distributed approach assigns one state to each worker. Thus, the number of states whose value functions can be computed in parallel is exactly equal to the number of workers. With increase in number of workers the execution time for this version decreases.

But, the effect of the number of workers on the version 2 of distributed programming is much more interesting to observe. This version assigns a batch of states to each worker. While the execution time for this version was expected to decrease with increase in number of workers, it is observed that this approach performs better with 2 workers than with 4 workers. Parallel programming, especially on distributed systems, has overhead costs of creating threads, and organizing processes when multiple processes write to the same object. There is a bottleneck that is observed in both the distributed approaches when performing updates. Thus, with only 2 workers in version 2, value function and policy of 50% of the states were updated by a single worker without having to wait for another. But with 4 workers, there were instances where a worker had to wait in a queue for

longer.

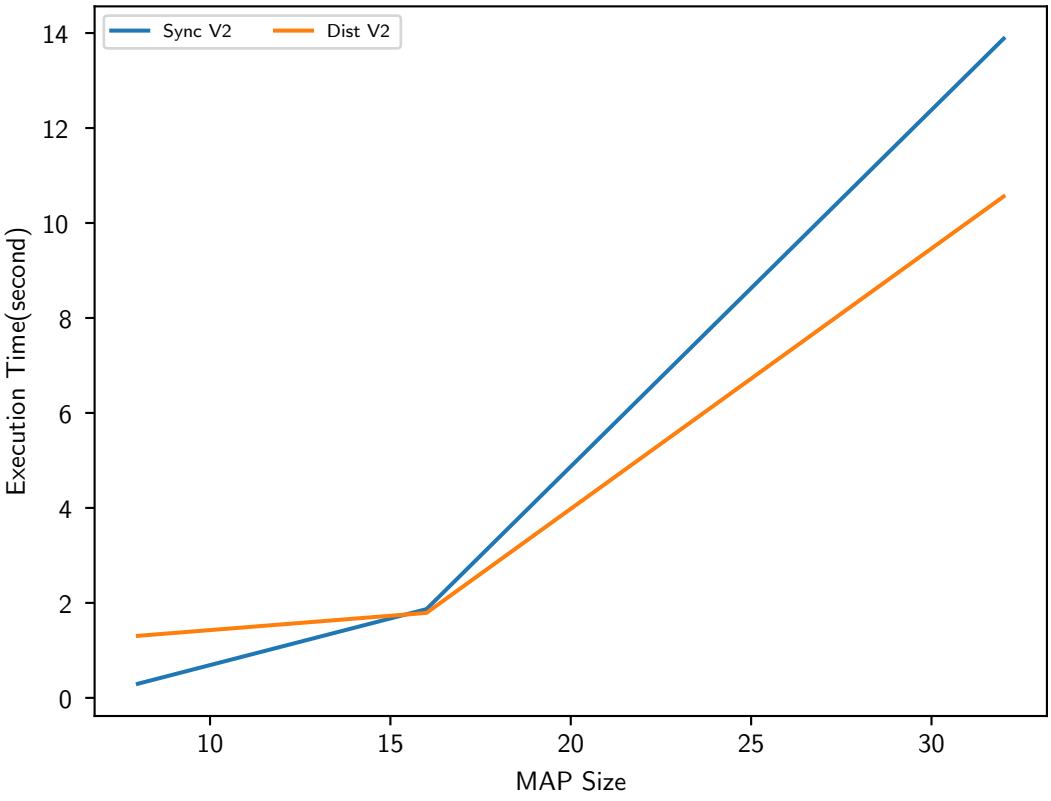
Distributed programming has its benefits in terms of performing computations in parallel. These benefits could be very well experienced as the number of workers were increased to 8.

4 Different distributed programming approaches

Distributed programming approach employed in version 2 outperforms the one employed in version 1 for each map size and for each number of workers used. This is mainly due to how the states are distributed amongst workers. In version 1, each worker gets one state. It computes the value function and policy of the state and updates it to the object. When this worker updates, every other worker is in the queue. Thus, every worker waits in the queue only to update the value function and policy for a single state, and, is then assigned a new state.

In distributed programming it has been observed that performing a batch of operations in a iteration by each process is better than performing a single operation in a iteration by each process. This is due to the waiting time for the processes in order to perform updates. When a process has to perform a batch of operations, it computes all of them first before getting in the queue and waiting for other processes in order to perform updates. In version 2, as the number of batches are equal to the number of workers, each worker has to wait in a queue only once at maximum.

5 Distributed V/s Non-distributed approach



Version 2 of non-distributed approach outperformed the version 1 of non-distributed approach, and version 2 of distributed approach outperformed the version 1 of distributed approach. Version 2 of both distributed and non-distributed approaches have almost the same performance on average. While the non-distributed version performs better on a map of size $8X8$. But as the map size increases and so does the number of states, the distributed approach performs better.

In this distributed approach, a batch of states are assigned to each worker. Thus, the parallelism exists between workers. While each worker computes the value function and policy of each state in a batch iteratively. Thus, the performance of non-distributed approach is identical to the distributed approach if there existed only one worker and the total number of states was equal to the number of states in a batch.

This distributed approach outperforms the version 2 of non-distributed approach when number of states are higher. Thus, computation of value function and policy among batches is performed parallely. This helps in producing a better, more efficient implementation.