AI-based Distribution & Smart Contract for OEMs using Reinforcement Learning and Hyperledger Fabric

Shivam Rai, Niraj Kesarwani, Siddharth Parashar

Abstract: OEM (original equipment manufacturers) Supply Chain Management always require a mechanism to ensure trust via contract methods and intelligent allocation of delivery load & logistics to save cost. Smart Contracts on Blockchain technologies is one of the best suited mechanisms to ensure trust in contracts. These contracts are immutable and transparent to related participants. There are multiple optimization methods being used in the industry to address demand allocation problems but as the size of the problem grows most of the methods become inefficient in performance and practicality. We formulate MDP (Markov Decision Process) to model this situation .Problem of dimensions increasing due to state and action increment as the horizon increases has been described as "Curse of Dimensionality" by Bellman who addressed the problem with condition of optimality. Policy Iteration algorithm is a known methodology in reinforcement learning that addresses the problem of decision making by formulating a deterministic in cost or reward and probabilistic in state transition model that solves a stochastic optimization. Implementation described in this paper has been achieved on Google Cloud Platform using Hyperledger Fabric, Node-Red and R Server. The text describes the stack architecture, flow and algorithm and also focuses upon cost viability of the solution.

Index Terms: Artificial Intelligence, Markov Decision Process, Policy Iteration, Hyperledger, Smart Contract, Blockchain, Distribution System

1. INTRODUCTION

Smart distribution systems and contract management is indispensable for the business in Supply Chain Management. So the contracts between the Logistics partners and Warehouse distribution system requires a method that can smartly manage the contracts as "law-of-code" and is immutable.

Blockchain based application are gaining momentum in SCM domain due to distributed application nature and contracts being executed upon rules and security of the artefacts. The participants cannot change what has been committed to the chain and hence immutability is one core strength of Blockchain applications. Participants only have a choice of being or not being part of the network.

To allocate demand and plan distribution of the assets to the customers or destinations require an optimization based upon cost of the delivery and SLA.

"Authors are member of Professional Services Group at TIBCO Software India"

There are a few methods that involve linear programming approach but are deterministic and do not address sub-problem scenarios .Hence they can't be implemented for real-time systems.

There are algorithms that are widely used in Artificial Intelligence domain like Markov Decision Process with Policy Iteration that enables system to plan action for every available state or location. These techniques are very popular in stochastic optimization as number of states and actions are large. MDP problems are a better fit for online learning as future state depends only upon the current state.

Approximate Dynamic Programing has emerged as a powerful tool to tackle such problems. The recurring theme of these algorithms enable learning policies quickly and efficiently and also in online setting.

In this paper, we will discuss the implementation of Blockchain using Hyperledger Fabric and policy iteration algorithm for smart allocation and distribution that will be implemented using MDP Tool Box in R. The entire setup has been done on *Google Cloud platform (GCP).*

2. HYPERLEDGER FABRIC

Hyperledger is Blockchain project that follows a design philosophy that includes a modular extensible approach, interoperability, an emphasis on highly secure solutions, a token-agnostic approach with no native crypto-currency and ease of use.

Smart Contract is a business logic running on a Blockchain. These smart contracts can be as simple just as a field update or they may be complex as hundreds of rules executed with interdependencies. There are two types of smart contracts:

(a) *Installed Smart Contracts*: They install business logic on the validators in the network before the network is launched.
(b) *On-Chain Smart Contracts*: They deploy business logic as a transaction committed to the Blockchain and then called by subsequent transactions.

A smart contract in Hyperledger is a chain code that can be written in Go or Java-script language. It runs in secure Docker container that remains isolated from the endorsing peer processes. Chaincode initializes and manages the ledger state through transactions submitted by applications. There are two types of chaincode, (i) system chaincode and (ii) application chaincode. A chaincode starts with a package that encapsulates critical metadata about the chaincode that includes name, version and counterparty signatures to ensure the integrity of the code and metadata [11]. The Hyperledger services can be categorized as (i) Identity, (ii) Policy, (iii) Blockchain, (iv) Transactions, (v) Smart Contracts.

Modular Hyperledger is developing modular, extensible frameworks with common building blocks that can be reused. This modular approach enables developers to experiment with different types of components as they evolve, and to change individual components without affecting the rest of the system. This helps developers to create components that can be combined to build distributed ledger solutions well-suited to different requirements. This modular approach also means that a diverse community of developers can work independently on different modules, and re-use common modules across multiple projects [12].

3. SMART DISTRIBUTION

The smart distribution is a system that relies on demand and fulfilment according to the cost and transition probability of moving from one state to other. Here is a detailed description:

3.1 Markov Decision Process

A Markov Decision Process [1], [3], [5] is a controlled stochastic process satisfying the Markov property with costs assigned to the state transitions. A *Markov Decision problem* is a MDP with a performance criterion. Solution to a Markov Decision problem is a policy that maps state to actions and tries to determine the transitions that minimize the cost or maximize the reward according to the performance criteria.

The environment evolves probabilistically occupying a finite set of discrete states and for each environmental state there is a finite set of possible actions that may be taken by the learning system. Every time the learning system or agent takes an action, certain cost is incurred. States are observed and actions are taken with cost incurred at discrete time [10]. Most importantly, the transition probability from *state i to state j* depends entirely on the *current state i* and corresponding action a_{ik} where i the present state and k is the number of actions available.

A Markov decision process illustrates the dynamics of an agent interacting with the stochastic environment. A policy π is a mapping from states to actions. If the policy is independent of the current stage, it is said to be stationary. A fundamental outcome of MDPs is that there exists a stationary policy that dominates or has equal cost to every other policy (Bellman, 1957). Such policy is termed as *optimal policy* and total cost that it assigns to every state is called as the optimal cost. A ε-optimal solution is a policy whose total cost for every state is in ε difference from the optimal cost. In these types of optimality condition, we shall remain focused upon total cost function which is unique but optimal policy may not be unique. To understand this we need to see the *expected discounted cumulative cost* performance criteria [3].

In the expected discounted cumulative cost criteria , the cost of the path traversed by policy in all the time steps is discounted by a factor γ such that $0 < \gamma < 1$ for every instantaneous time step.

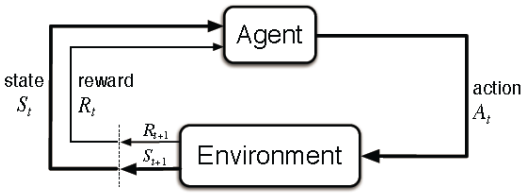

Fig 1: Agent-Environment Interaction and reward flow. (Courtesy: Sutton and Barto)

A finite MDP can be defined by the tuple combination of state and action. The probability of moving from state 's' to 's`' with action 'a' can be denoted as:

$P(s`|s,a) = \Pr(S_{t+1}=s` \mid S_t=s, A_t = a)$

We can represent our system as being in state 's' at time 't'. If we choose action 'a' and then we let a probability p(s`|s, a) be defined that we land in state's`' from 's'. If the contribution /reward for action 'a' in state 's' be C(s, a), then we can find best action by solving Bellman's optimality equation as:

$$V(s) = max_a (C(s, a) + \gamma \Sigma_{s`} p(s`|s, a) V(s`)) \quad \text{eq. (1)}$$

Widely used algorithms for solving MDPs are iterative methods. The one implemented in the model used for this paper is policy iteration [4]. Later we will check the performance of Value Iteration and LP models as well. Policy Iteration algorithms has two phases that run sequentially in the same iteration. (i) Policy Value Determination (ii) Policy Improvement.

The basic policy iteration algorithm works as:

1. Let $\pi`$ be a deterministic stationary policy.
2. Loop
   (a) Set π to be $\pi`$. (This step allocates value of arbitrary policy that is considered as stationary)

   (b) Determine, for all $i \in \Omega_s$, $E_\pi(\Sigma_\gamma | i)$ by solving the set of N equations in N unknowns described by equation 1. $E_\pi(\Sigma_\gamma | i)$ is estimated value of V(s)

   (c) For each $i \in \Omega_s$, if there exists some $k \in \Omega_A$ (action) such that

   $$[C_i^k + \gamma \Sigma p_{ij}^k E_\pi(\Sigma_\gamma | j)] < E_\pi(\Sigma_\gamma | i)$$

   (Where p_{ij}^k is the probability of transition from state 'i' to state 'j' for any action 'k'.)
   Then set $\pi`$ (i) to be k else set $\pi`$ (i) to be π (i).

   (d) Repeat loop if $\pi \neq \pi`$.
3. Return policy π.

3.2 Dynamic Programing

Stochastic optimization problems vary in notation and that's the reason why contextual problems related to MDPs arise. This also causes issues while communicating between different communities like decision science, control systems or operational research.

According to Haykin, "A dynamic programming problem can be of finite or infinite horizon". ADP (Approximate Dynamic programing) largely used for solving finite horizon problems is both a modelling and algorithmic framework [4].This is one way to solve the Bellman equation by modelling stochastic optimization problem in (i) State Variables, (ii) Decision/Actions/Controls (iii) Exogenous Information (probability distribution of physical system) .It addresses the issues that Discrete MDPs could not address i.e. High Dimension of decision or action variable. It has to overcome problem of multi-dimensional state variable as well as in large action space, both state and action increase the dimension of the problem. This has been referred as "curse of dimensionality".

3.3 Bellman Optimality and Residuals

Dynamic programing technique rests on very simple idea known as the principle of optimality.
"An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must contribute an optimal policy with regard to the state resulting from the first decision" [10]. That means for any state 'i' and 'i+1' somewhere in the middle of state space graph, expected value of state 'i+1' will depend upon state 'i'. This follows Markov property that future value is only dependent upon the present value. At every step in state space there will exist an action yielding better or equal reward for the next future state. The sequential set of these actions is an optimal policy.

For policy iteration algorithm, we need to define the maximum no. of iterations for which the algorithm should run and find an optimal or near optimal policy. This maximum number can be set in advance or determined by using *stopping rule*. By examining the Bellman residual during value iteration and stopping when it gets below a threshold as $\varepsilon`=\varepsilon(1-\gamma)/(2\gamma)$ where ε is the maximum difference between n^{th} and $(n-1)^{th}$ state value function as proposed by "Williams and Baird" [3].

ε-optimal policy is defined as where expected value of the state-action pair has a nominal difference with total cost discounted in the iteration. So when the condition is reached, iterations shall stop. After n iterations the estimated total cost function can differ from the optimal total cost function by no more than $2M\gamma^n / (1-\gamma)$ at any state. M is the maximum magnitude of instantaneous cost [Tseng],[3].

4. SYNCHRO-MODAL DISTRIBUTION AND SMART CONTRACT IMPLEMENTATION

4.1 Blockchain – Hyperledger Fabric Design

A smart contract in Hyperledger Fabric is a program called as chain code. Chain code can be written in Go Language or Java Script (node.js) in Hyperledger. Chaincode initializes and manages ledger by the transactions submitted by the applications using the REST interface.

A chaincode typically handles business logic that members of the network have agreed to. The state created by a chaincode is scoped exclusively to that chaincode and can't be accessed directly by another chaincode.

Transactions defined in the chaincode:

1. Upload/create asset for the Hyperledger Inventory.
2. Place order request for destinations.
3. Transfer asset from local warehouse to the destination upon demand arrival.
4. Transfer asset from central ware-house to local warehouse.

Transaction of upload asset is straight forward and a request to add a new part will be done. Whereas transaction like transfer of asset or part will be based upon availability of parts. These transactions could be more complex when implemented.

The query file contains the query to retrieve "assigned status" request from Couch DB and same can be stored in databases or flat files. In current implementation, we have used csv files to get the information and they are saved manually to the directory path for R server by the analyst.

4.2 Design of Policy Iteration

The proposed solution is inspired from a synchro-modal of allocation and supplies as described in [1]. The dynamic demand fulfilment as in [2] is based upon approximation of relative value function which addresses transit behaviour for stock out probability for any part supply. This paper is focused upon the planning part of vehicle fleet to meet the demand and hence have designed MDP process per vehicle. Same vehicle carries parts for delivery to multiple destinations with aim to incur minimum cost of logistics, penalty due to SLA breach which is assumed as certain percentage of the logistics cost for our model. There is a specific calculation provided in [2] for generating the final cost. This *cost matrix/reward* is calculated as per the rules of penalty for late delivery. Although the cost matrix will have state to state transfer cost and that includes time factor based cost of state transfer and since it is not a route optimization but a decision problem so we are not interested in point to point cost but we need to find the impact of traversing a path.

The main assumptions have been followed as in [2]. There are echelon model as central warehouse, local warehouse and destinations.

1. The central warehouse has ample stock. In real scenarios we often see that the central warehouse can obtain new items from multiple channels such as regular supplies, emergency suppliers etc.
2. Part request from each customer follow an independent Poisson process. This is a common assumption.
3. Inventories at local warehouse are controlled through continuous time based stock policies. Though as per the rule there will be a one-to-one replenishment from central warehouse. But there is a rule for part transfer from central warehouse to local warehouse based upon request from local warehouse or an auto transfer on the smart contract.
4. Service deadlines, penalty cost and delivery times are such that it is never beneficial to backorder demand.
5. Destinations are located in the same geographical area.

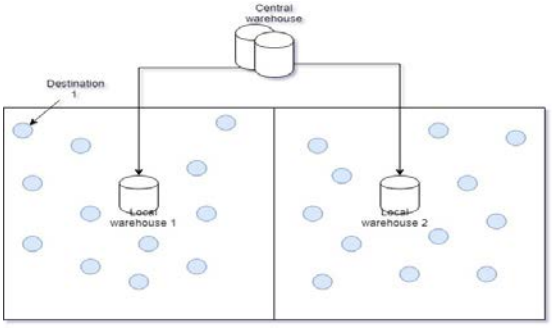

Fig 2. Distribution Layout

The layout defines a Central warehouse, two local warehouses and multiple destinations. These destinations request part from the LW. Demand overflow is not modelled in this paper. We solve the Bellman optimal solution using the MDPToolBox with ε-greedy approach. This is approximate dynamic programing to estimate an optimal solution of policies in a finite horizon condition. As described [1] in synchro-modal planning, it is possible to change the transportation plan i.e. the services and transfer needed to bring a freight from its origin to its destination at any point in time. Even though the planner might have a complete plan at a given moment, only a part of such a plan is implemented and rest can be re-estimated via a real time feed of exogenous information to calculate transition probability. That would mean that if a vehicle has a plan that maximizes the reward as {s1, s2, s3, s4, s5} and while the agent/vehicle is at {s3} and new transition probability is received or the reward has changed, a new policy can be rolled out with minimal time of processing.

Unlike [2], this paper suggests to prepare a plan with optimal policies and re-evaluate and publish as required.

First we design our MDP model by capturing the *probability of transition*. In the experiment setup it has been simulated using a *log-normal distribution*. Second, we evaluate the policy iteration algorithm with value iteration and LP model to derive the sequence of actions and compare the performance over system time and iterations.

4.3 MDPToolBox

This is an R package authored by Iadine Chades, Guillaume Chapron, Marie-Josee Cros, Frederick Garcia, and Regis Sabbadin. It provides functions like mdp_check ( ), mdp_policy_iteration ( ), mdp_value_iteration ( ), mdp_LP ( ) etc. to implement and solve MDP problems. In this setup we have used this package to solve the allocation problem.

Analyst from local warehouses will run a script to launch R server on Google cloud platform. This script will have start and stop functions available in a package called "googleComputeEngineR" as gce_vm_start ( ) and gce_vm_stop ( ). Once R server starts, program to run Policy Iteration algorithm can be invoked from R console or command prompt. In current implementation as we are using csv files in the directory path, so analyst can easily run program from R console.

4.4 Technology Stack

The technology stack used for the implementation is defined as:

- GCP Account
- Linux Machine with Ubuntu instance
- Hyperledger fabric
- NPM package
- Composer Playground
- Node-Red
- R server on n1-highmem-2 machine
- R studio on local machines for analyst

4.5 Experiment and Result

We have conducted two different experiments where we have tried Gamma values in the range of 0.65 to 0.99 with Experiment.1 as state=100 and available actions=100 so resulting into a 100x100x100 [state x action x state] data structure. The transition probability matrix is sparse and some of the reward values have been converted to negative.

Fig 3 shows the relation between the discount rate and CPU time to calculate a policy on x86_64 windows machine with INTEL core i-7 4600 CPU with R version of 3.5.1. We can observe that as the discount rate is close to 1 the CPU time as higher and it comes down to 0.25 sec with discount rate close to 0.75.

Experiment.2 uses a 1000x1000x100 problem where states=1000 and actions=100. So there are 1000 states available for the agent and it can perform 100 different actions per state.

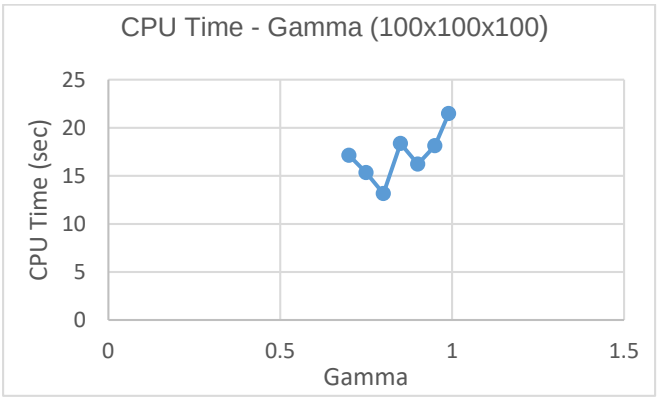

Fig 3. CPU time and Gamma (discount rate) for 100 x100x100 problem.

This problem requires a data structure that will consume 760 MB of RAM space. So if the actions increase to 1000 then space required will be 7.6 GB. Fig 4, shows that as discount rate is 0.99, CPU time is 21.48 secs and its lowest for discount rate 0.65 with 12.63 secs. Although it may not appear very

significant for single policy but when solving multiple MDP problems and sub-problems this factor multiplies and plays an important role in computation cost and model.

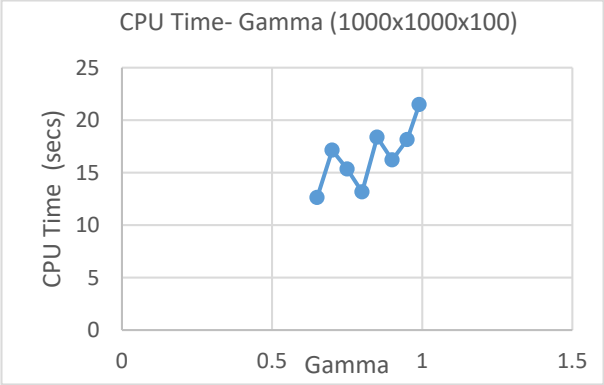

Fig 4. CPU Time- Gamma (1000x1000x100 problem)

When tried same experiment with LP model, it took more than 30 mins to solve the same problem.

CONCLUSIONS

With the combination of Policy Iteration over choice of destination and smart contract management OEMs can efficiently manage supply chains. The setup remains custom and complexity varies according to the organization. Smart contracts ensure part assignment is according to the rule and Policy Iteration makes sure that the state next chosen gives the best reward value and also the ability to solve the sub problem is advantage when real time information flows in. The computation costs is very low as the R server is launched only when needed and hence this kind of setup is viable option for logistics partners of OEMs.

ACKNOWLEDGMENTS

We would like to thank TIBCO Software Inc. for being an incubator of innovation and our colleague Ashish Rajput, an integration specialist for the assistance. Special thanks to IIMM (Indian Institute of Materials Management) for helping professionals with niche knowledge in SCM.

REFERENCES

[1]. Arturo Perez Riverra, Martjin Mes, "Service and Transfer selection for freights in a synchromodal network".WP504, 2016, Research School for Operations Management and Logistics, Eindhoven.

[2]. H.G.H Tiemessen, M. Fleischmann, G.J van Howtum, JAEE von Numen , E. Pratisini , "Dynamic demand fulfilment in spare parts networks with multiple customer classes".2012, Research School for Operations Management and Logistics , Eindhoven.

[3]. Michael L. Littman, Thomas L. Dean, Leselia Pack Kaelbling, "On the complexity of solving Markov Decision problems". Brown University.

[4]. Warren, B. Powell, "Approximate Dynamic Programming, chapter: Modelling", Wiley & Sons Inc. 2010.

[5]. Richard S. Sutton, Andrew G. Barto, "Reinforcement Learning: An Introduction", Second Edition, MIT Press, 2012.

[6]. Nils J. Nilsson, "Artificial Intelligence: A new synthesis", Stanford University, Harcourt Asia, 2000.

[7]. Tommi Jakkala, Satinder P. Singh, Michael L. Jordan, "Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems". Department of Brain and Cognitive Science, MIT.

[8]. Lucian Busoniu, Robert Babuska, Bart De Schutter , Damian Ernst, "Reinforcement Learning and dynamic programing using function approximators", http:www.dcsc.tudelft.nl/rlbook.

[9]. Csaba Szepesvari," Algorithm for reinforcement learnings", Synthesis lectures on Artificial Intelligence and Machine Learning, 2009, Morgan & Claypool Publishers.

[10]. Simon Haykin, "Neural Networks: A comprehensive foundation", Second edition, Pearson Press.

[11]. Imran Bashir, "Mastering Blockchain: Distributed Ledgers, decentralization and smart contracts explained", Packt Publishing Ltd, ISBN 978-1-78712-544-5.

[12]. White Paper, "Hyperledger Architecture: Vol 2 Smart Contracts", wiki.hyperledger.org/groups/architecture/architecture-wg