COMP 4560 Undergraduate Industrial Project

Developing Smart Contracts for Task Matching in Distributed System on Hyperledger Fabric

Hualong Qiao & Xiao Peng

7797627 & 7813579

1 Abstract

The work is about formulating task matching problem as smart contracts on Hyperledger Fabric which is a blockchain. Currently, we've formulated task matching problem on fabric, solvable by conventional heuristics and explored a meta-heuristic algorithm PSO to find solutions to task matching problem.

2 Introduction

Blockchain technology has been developing rapidly since the introduction of its peer-to-peer characteristic. It eliminates a number of vulnerabilities that a centralized ledger has, by storing data across its peer-to-peer network which has made secretly changing data in a blockchain practically impossible [6][5]. Apart from being used to underpin cryptocurrencies like bitcoin, blockchain technology could also be used to solve many problems. Such as Hyperledger Fabric, it is private-permission blockchain which would be used to solve task matching problem. For task matching problem, we have a set of tasks T = t1, t2, ..., tn, and compute resources R = r1, r2, . . . , rm where each task must be assigned to run on a specific resource or limited tasks could be assigned to resources based on availability. We introduce a Hyperledger Fabric network with the purpose of studying Task Matching algorithm on various network peers [3].

3 Background preparation

Hyperledger Fabric is an open source GitHub project, using Golang. Our experience of previously designing software in software engineering using Java, together with course level experience with the C programming language made learning Golang not a difficult task. Our previous work level experience in computer network related field could also help me exploring algorithms to formulate task matching problems as smart contracts on Hyperledger Fabric. Beside what we learned ourselves, we also got help from Dr. Thulasriam and PhD student Mr. Eid about heuristics implementation.

4 Related work

For a peer on Hyperledger Fabric network to find a solution to a task matching matrix stored on the blockchain, different algorithms need to be used. What had been done was using min-min, max-min using task driven approach and simulated annealing to find solutions to the task matching problem [3]. Compare with min-min and max-min, simulated annealing is a meta-heuristic which is computationally intensive [2].

5 Problem statement

The goal of this project is to formulate task matching problem as smart contracts on blockchain, Hyperledger Fabric. Hyperledger Fabric is an open-source project that runs chaincode [4], which is a smart contract that can view and update the current status of the ledger where all of the data on the blockchain is stored.

6 Methodology

Smart contract is about enforcing negotiation of a contract, the demand part of the contract could be viewed as the task in task matching problem, and the supply part in the contract could be viewed as the compute resource in the task matching problem. We are focusing on two types of task matching approach: task driven, and resource driven. Three algorithms had been used to find solutions to task matching problem on blockchain: min-min, max-min and min-max. We proposed to study and explore a meta-heuristic algorithm, particle swarm optimization (PSO) for this purpose. PSO has been well studied for task matching problem [2] in the recent past.

6.1 Hyperledger Fabric

Blockchain is a new application mode of distributed data storage, point-to-point transmission, consensus mechanism, encryption algorithm and other computer technologies. Blockchain is a distributed ledger technology. It is decentralized, open and transparent, so that everyone can participate in the database records.

[4] Hyperledger Fabric is a platform for distributed ledger solutions. Hyperledger Fabric is supported by a modular architecture with excellent confidentiality, scalability, flexibility, and extensibility. Hyperledger Fabric is designed to support direct unpluggable enablement of different modular components and to accommodate the intricacies of scenarios within an economic ecosystem. Hyperledger Fabric provides a unique scalable and extensible architecture, which is a significant difference between Hyperledger Fabric and other blockchain solutions.

6.2 Smart Contract

The smart contract model is a computer program that runs on a replicated, shared ledger that processes information, receives, stores, and sends digital assets. Smart contract is more than just a computer program that can be executed automatically. It is itself a system actor. It can respond to incoming information, receive and store value, and send information.

The process is like a person who can be trusted to temporarily hold assets, always following prior rules. Intelligent contract is equivalent to a contract that can be executed directly by the computer network. As long as the preconditions are met, the contract will be triggered. There is no need to establish a trust relationship between the two parties.

Smart contract is self-verification, self-execution and tamper-proof. Smart contract can transform legal obligations into automated processes, ensuring a higher degree of security, less reliance on intermediaries and lower transaction costs. Blockchain and smart contract have the potential to transform many traditional industries.

6.3 Project Structure

The project contains 7 organizations and each organization has one anchor peer and one leader peer. The anchor peer is a peer node on a channel that all other peers can discover and communicate with. Peers belonging to an organization can query this peer to discover all peers belonging to other organizations in the channel. When an ordering service node must send a block to the peers in the channel, it sends the block to each of the leading peer associated with the organizations. The leading peer in turn distributes this block to other peers belonging to the same organization. Distribution is done using gossip protocol.

Besides, there is an ordering service which coordinates transactions for a network; it creates blocks containing transactions in a well-defined sequence originating from all the different applications connected to the network. It is the central part of Hyperledger Fabric and responsible to create consensus in the blockchain. The CLI node is allowed the script to send comments directly to the network.

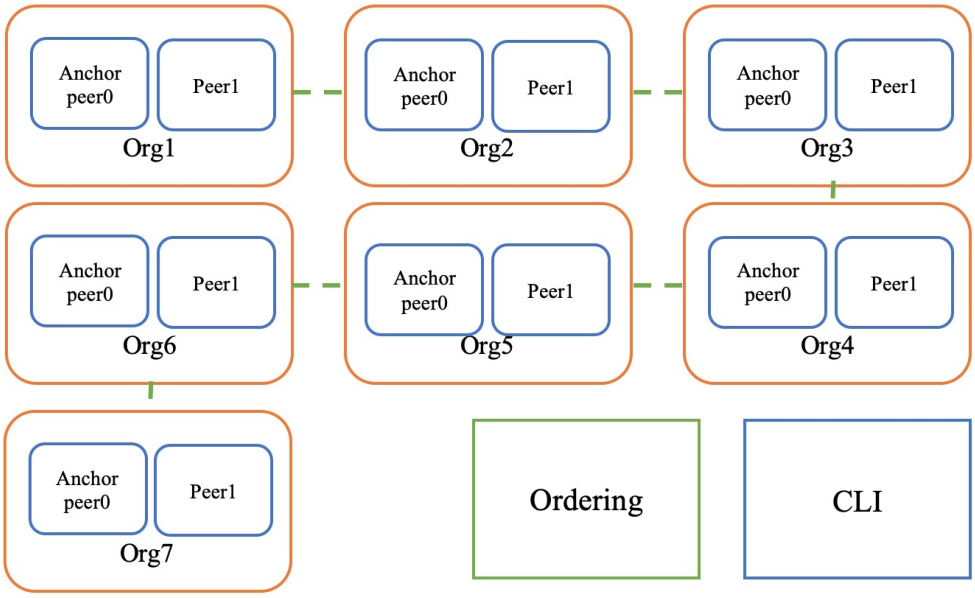


Figure 1: Project Structure

6.4 Workflow

In this project, we mapped task matching problem as a smart contract which can generate task matching problem randomly, assign the problem to various algorithms, compare the outcome and pick the optimal solution. Here is how the system works below.

- 1. When running startNetwork.sh, the script drives the createChannel command against the supplied channel name which is task_matching and uses the channel.tx file for channel configuration. The output of createChannel is a genesis block which gets stored on the peers' file systems and contains the channel configuration specified from channel.tx.

- 2. Then the joinChannel command is exercised for all peers, which takes as input the previously generated genesis block. This command instructs the peers to join task_matching channel and create a chain starting with task_matching block.
- 3. Now we have a channel consisting of peers with related organizations. These relationships are defined through the crypto-config.yaml and the MSP path is specified in our docker compose. The anchor peers are then updated. We do this by passing the OrgMSPanchors.tx artifacts to the ordering service along with the task_matching channel.
- 4. The task matching chaincode is packaged and installed on all peers. The chaincode is then separately approved by all organizations, and then committed on the channel.
- 5. The chaincode Init is then called which starts the container for the target peer, and initializes the key value pairs associated with the chaincode.
- 6. A query is sent to "work" peer to generate a task matching matrix.
- 7. Queries are sent to from peer 1 to peer 6 to get their default status.
- 8. Queries are sent to calculate task matching problem on each peer. MIN-MIN on peer 1, MIN-MAX-TASK on peer 2, MAX-MIN-TASK on peer 3, MIN-MAX-RESOURCE on peer 4, MAX-MIN-RESOURCE on peer 5, and PSO on peer 6 respectively.
- 9. Queries are sent to from peer 1 to peer 6 to get their current status.
- 10. A query is sent to pick the optimal solution and write to the ledger.
- 11. Get the optimal solution.

7 Algorithm Comparison

We use ETC (expected time to complete) matrix to represent task matching problem, for an ETC matrix, the size of the ETC matrix is defined by number of tasks and resources for a task matching problem. The estimated execution times of a task on different compute resources are on the same row of ETC matrix, and the execution time of a resource for different tasks are in the same column of ETC matrix, so for ETC [A, B], it stands for the execution time of task A on resource B.

In our experiment, ETC matrices we used have more number of tasks ($|T|$ = n) than number of compute resources ($|R|$ = m), shown as m >n. There are also two other circumstances: $|T| = |R|$ which could be solved using the same algorithm, and for scenario: $|T| < |R|$, task T_i could be assigned to R_i which has the minimum expected time to complete T_i .

Two approaches for task matching problem are: task-driven approach and resource-driven approach. Task driven approach using min-min algorithm is to:

- 1. Choose the min of each row i_1, i_2, i_3, ... i_n
- 2. Select the min of i_1, i_2, i_3, ... i_n and get i_j which is the smallest among them
- 3. Remove the row contains i_j, and add i_j to the same column of other rows
- 4. Repeat until the number of column is 1

While for min-max heuristic using task-driven approach is still choosing the minimum of each row but choose i_j which is the biggest among i_1, i_2, i_3, ... i_n in this case. For step 3, and 4, same as in min-min heuristic using task-driven approach.

For max-min heuristic using task-driven approach is to choose maximum among rows, but choose minimum among the maximum values chosen and step 3, and 4 are the same as in min-min heuristic shown above.

Resource driven approach is similar to task driven approach but first choose among column, then choose among the results extracted from each columns, following step is still the same as shown in min-min resource driven approach.

Two approaches: task-driven approach and resource driven approach with three conventional algorithms: Min-Min, Min-Max, and Max-Min are implemented to solve task matching problem. How we calculated the run time for each resource R_1, R_2, R_3, ... R_m for each algorithm:

- 1. Use the task-resource assignment returned from heuristics h: T_aR_b, T_cR_d, T_eR_f, ... T_iR_j to get the corresponding run time from the ETC matrix M_A.
- 2. Calculate the run time for each compute resource R_1, R_2, R_3, ... R_m
- 3. Among R_1, R_2, R_3, ... R_m, select the maximum run time assigned to resource Ri

The run time for R_i which has the maximum runtime among all compute resource would be the makespan for M_A using heuristic h. After that, we implemented meta-heuristic PSO and compared it's result with the other three algorithms using two approaches which is six in total.

Particle swarm optimization (PSO), formulated by Kennedy and Eberhart, the original intent was to graphically simulate the graceful but unpredictable choreography of a bird flock[1]. In PSO, there is a swarm of particles, where each particle represents a potential solution. Each particle are initialized with random position with a randomized velocity. Particles records its personal best solution achieved so far (*pbest*) by using the fitness function. For each problem, the fitness function could be different. The fitness function determines how "well" the solution inside the particle is. Among all the personal best (*pbest*) of all particles in the swarm, the best of *pbest* is called *gbest*.

With pbest of each particle and the gbest for the swarm, in each iteration of the PSO algorithm, each particle would change their velocity towards pbest and gbest. The tendency toward pbest and gbest are determined by c_1*r_1 and c_2*r_2. The bigger c_1*r_1 is, particle p_i would have bigger tendency of moving toward *pbest*. And the bigger c_2*r_2 is, particle p_i would have bigger tendency to move toward *gbest*, where r_1 and r_2 here, are the two random number. We use formula below to update the velocity of a particle p_i after each iteration:

$$V_{i+1} = w*V_i + c_1*r_1*(pbest - p_{curr}) + c_2*r_2*(gbest - p_{curr})$$

We use this formula to update velocity of each particle and to prevent the speed from getting too fast, we multiple the previous velocity by a parameter w, which is inertia. If inertia is not included here, the velocity would get too big, and lead to the circumstance that particles move too much for a single step, and might pass the optimal position, where the optimal position could yield a better fitness.

After updating the velocity for particle pi, We use below formula to update its position:

$$p_{i+1} = p_i + V_{i+1}$$

Where p_{i+1} stands for the updated position, and p_i stands for the position for the last round. After done updating velocity and position for all particles in the swarm, we then calculate each of their *pbest* and get *gbest* among *pbest*s

The condition of terminating the loop could also vary, some examples including: running for a certain amount of iterations, or if *gbest* does not change after a certain amount of iterations, etc.

In the task matching problem, how we applied PSO algorithm is like this:

- First step, we initialized certain amount of particles, for those particles, we randomly initialized their velocity and position.
- Second step, we find their *pbest* and update *gbest*, as it is the check after the initialization, *pbest* would be their current position.
- Third step, enter a loop and terminate based on conditions. (loop for a certain number of times, *gbest* have not changed for certain number of iterations, etc.)
- Fourth step, change particles' velocity and position using formulas given above.
- Fifth step, check their position, update *pbest* for each particle, and select *gbest* among *pbest*s

After the fifth step is done, loop back to fourth step until termination condition in step three is met.

We initialized position vector of particle pi with the length of number of tasks in the task matching problem, it is equivalent to a 2-d array of size 1 by number of tasks, that is with number of row equal to 1 and number of columns equal to number of tasks in the task matching problem. Example of such a position vector is:

T1	T2	T3	T4
1.233	3.44	10.99	−0.1

We do not have any position vector with values less than 0 in it for the initialized position vector, but for velocity of a particle, initialized velocity vector for a particle could have negative vector in it.

After a certain amount of iterations had passed, or any termination condition had met, we get *gbest* and it will be the solution returned by PSO algorithm, an example of such solution is shown below:

T1	T2	T3	T4
2.689	7.56	8.7	−10.2

Given the above result, for a task matching problem of three compute resources, we can get from the above vector that, task 1(T1) is assigned to compute resource 2 (R2) by casting 2.689 to integer, task 2 is assigned to compute resource 3 by casting any float numbers that is more than number of

tasks to the last compute resource ($R_{|R|}$), in this case: R3. Assign task 3 (T3) to compute resource 3 (R3), and assign task 4 (T4) to compute resource 1 (R1) as for any floating point number less than 1, case them to 1.

Given the fact that PSO algorithm is a random algorithm, for a small task matching problem, we sometimes might get two optimal solutions of same makespan but with different assignment (different *gbest* returned).

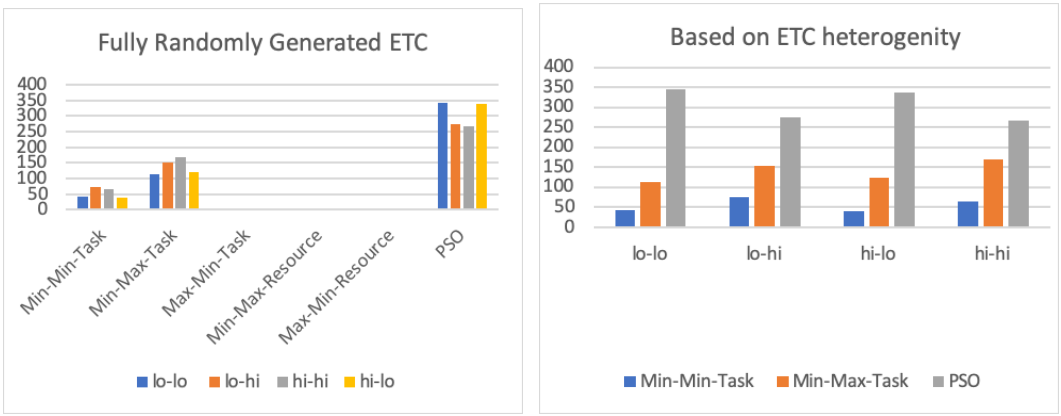
For the comparison of PSO and other conventional heuristics, we have three types of ETC matrix generated: fully randomly generated matrix, randomly generated matrix with increasing run time in each row, and randomly generated matrix with decreasing run time within each row.

For each type of ETC matrix generated, we divide them into four sub-categories as:

- Low variance between run time within each row, and low variance between each column, which is low-low sub-category
- Low variance between run time within each row, and high variance between each column, which is low-high sub-category
- High variance between run time within each row, and low variance between each column, which is high-low sub-category
- High variance between run time within each row, and high variance between each column, which is high-high sub-category

With twelve types of ETC matrix generated, we compared five conventional heuristics with meta-heuristic PSO, as min-min heuristic for task-driven approach is the same as min-min for resource-driven approach, we combined them together. With 500 randomly generated matrix for each sub-category of ETC matrix, with size of 12 tasks and 5 resources, we got result like this:

7.1 Experiments with fully randomly generated ETC matrix



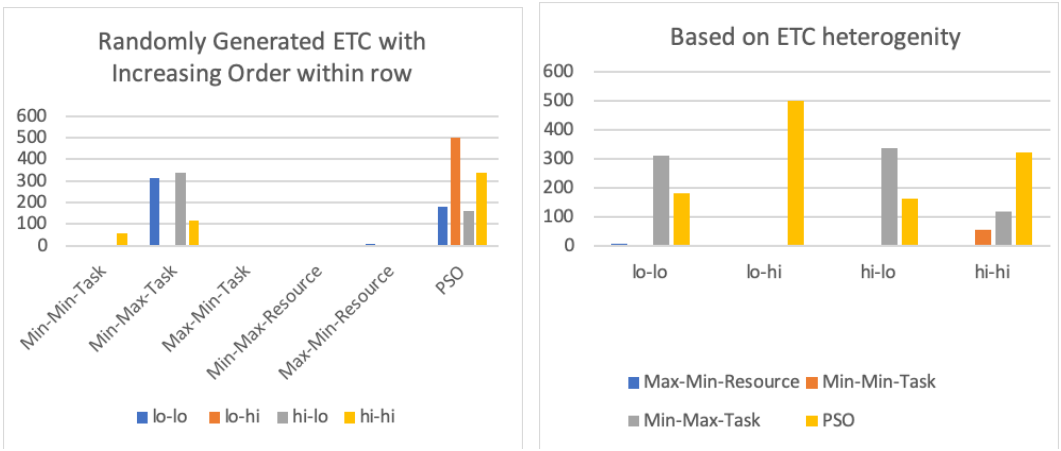
(a) (b)

Figure 2: Fully randomly generated ETC matrix

How did we choose an algorithm as the best solution is that, if an algorithm A get the shortest makespan among all other algorithms, then A is selected as the best, but if there is another algorithm B which yields the same makespan, but with shorter runtime, then algorithm B is set to be the best solution.

From the diagram, we can see within 500 experiments, for all sub-categories of ETC matrix generated, PSO performs the best among all heuristics.

7.2 Experiments with randomly generated ETC matrix with increasing Order within rows

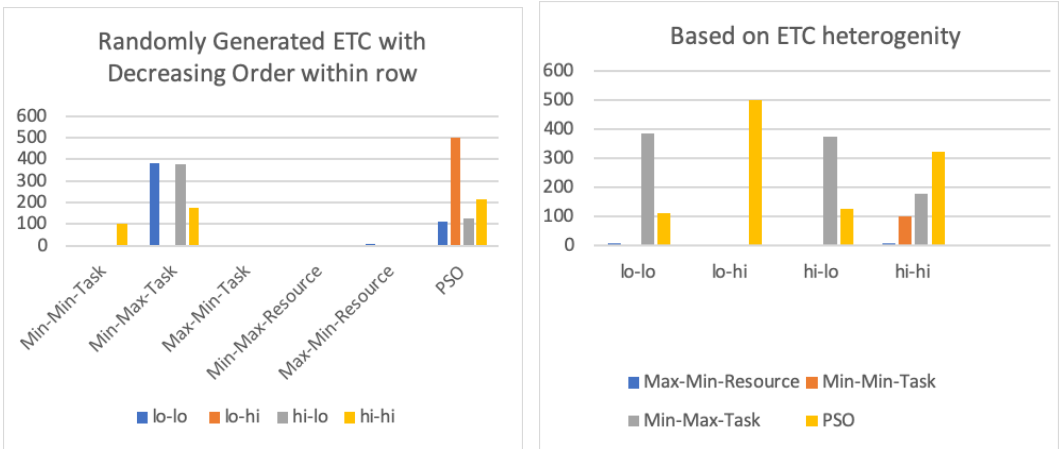


(a) (b)

Figure 3: Randomly generated ETC matrix with increasing Order within rows

From the two diagrams above, we can see in this case, randomly generated ETC matrix with increasing order in each row does not comply with the previous experiment, in this scenario, for low-low task and resource heterogeneity and high-low task and resource heterogeneity, min-max-task driven approach performs better than PSO algorithm, as for same solution returned by heuristics, min-max task driven approach uses the lease amount of time for the most of the time.

7.3 Experiments with randomly generated ETC matrix with decreasing Order within rows



(a) (b)

Figure 4: Randomly generated ETC matrix with increasing Order within rows

From the two diagrams above, we can see in this case, randomly generated ETC matrix with decreasing order in each row does comply with the previous experiment, in this scenario, for low-low task and resource heterogeneity and high-low task and resource heterogeneity, min-max-task driven approach performs better than PSO algorithm, as for same solution returned by heuristics, min-max task driven approach uses the lease amount of time for the most of the time.

In all three categories of ETC matrix generated, specifically when the task matching problem is small, when the heterogeneity is high-high for task and resource, PSO always performs the best, and randomly generated with increasing or decreasing order, min-max task driven approach performs better than min-min task driven approach and PSO.

But for larger scale task-matching problem, PSO heuristic would be best for most of the time and close to 100%, and if we exclude PSO for comparison, min-min task driven approach heuristic has the best performance among all conventional heuristics for task matching problem.

In terms of time consumption, for all conventional heuristics, they consumed much less time when compared with meta-heuristic PSO. To calculate 500 small scale task matching problem using conventional heuristics, among all heuristics, the biggest makespan is less than 0.02 seconds, while PSO algorithm would take more than 4000 seconds to calculate all 500 same problems. The more number of tasks and resources in the task matching problem, the bigger difference between conventional algorithm and PSO would be. Although for large scale task matching problem, PSO have much higher probability of returning the best scheduling scheme.

8 Infrastructure and facilities and expert personnel required

The project finished using our personal computers, no extra facilities was needed. With the help from Dr. Thulasriam and Mr. Eid Albalawi, a PhD student in the Cloud Computing lab, we met our goal for our project.

9 Outcome and deliverables

By the end of this project, our goal which is to formulate task matching on blockchain has met, and we solved task matching problem on fabric using different conventional heuristics with two approaches that are in the written smart contract, and we also compared the performance of conventional and PSO algorithm.

References

[1] Russell C. Eberhart and Yuhui Shi. Particle swarm optimization:developments, applications and resources. *Proc. of the IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 81—-86, 2001.

[2] Parimala Thulasiraman Eid Albalawi and Ruppa K. Thulasiram. Hglpso: Hybrid genetic learning pso and its applications to task matching on large-scale systems. *Proc. of the IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 997–1004, 2018.

[3] Ruppa K. Thulasiram Griffin McLennan, Eid Albalawi. Deploying a task matching network on hyperledger fabric blockchain. *Algorithmica*, 2019.

[4] hyperledger. hyperledger documentation. `https://www.hyperledger.org/projects/fabric`. Accessed Oct 4, 2019.

[5] Matthew Weber. Blockchain explained. `http://graphics.reuters.com/TECHNOLOGY-BLOCKCHAIN/010070P11GN/index.html`. Accessed Jul 15, 2018.

[6] Wikipedia. Blockchain decentralization. `https://en.wikipedia.org/wiki/Blockchain#Decentralization`. Accessed Oct 4, 2019.