Supporting Goods Movement in Warehouses using Decentralized Swarm Robotics – Integrated with Blockchain and Smart Contract

Bao Hoang Chuong Nguyen, Lam-Son Lê , Bich Hien Vo
Faculty of Engineering, Vietnamese-German University, Binh Duong, Vietnam
chuongnguyenbaohoang@gmail.com, son.ll@vgu.edu.vn, hien.vb@vgu.edu.vn

***Abstract*—This paper demonstrates an approach to integrating blockchain and Ethereum smart contracts into swarm robotics to perform distributed task allocation and task completion validation. The proposed system leverages an Ethereum-based smart contract to act as a high-level abstracted centralized co-ordinator despite the underlying operations being decentralized. The system follows a data-driven design, where the robots act based on identical data, which is the data stored in blockchain ledgers. This paper also proposed a distributed task complete validation method based on cross-checking within the swarm. The findings indicate that the design has 100 percent accuracy in smart contract processes, ensuring robust communication between robots and distributed task allocation. Based on a high successful rate of communication, 75 percent of the tasks are successfully done, and all the failed tasks are reported. The reason for failure mostly lies in the physical interference between robots while performing tasks. For the validation system, false positive judgements are not found, but false negatives could happen due to a logic problem, though it could be improved in future works. The findings indicate the approach successfully offers decentralized task allocation and validation with high accuracy and performance. However, further developments are needed to enhance efficiency, performance speed, and validation methods by sophisticating the control of smart contracts over the swarm. In addition, more hardware work is needed to improve decentralization and bring the system closer to real-world situations.**

***Index Terms*—swarm robotics, decentralized system, blockchain, smart contract, task allocation, task complete validation**

I. Introduction

Swarm robotics is a subfield of robotics that studies large groups of robots to work collaboratively towards a common goal. The field shows potential applications in areas such as search and rescue, environmental monitoring, and warehouse automation. However, ensuring the efficient coordination of a decentralized swarm system is still challenging, particularly in terms of task allocation and security. Blockchain technology is an emerging field of study known for its decentralized, tamper-proof ledgers. Integrating these new technologies offers a promising solution for a secure, transparent, trustless environment and robust global control with smart contracts for swarm robotics.

The integration of blockchain technology into decentralized swarm robotics attracts interest from researchers. Some benefits of blockchain to decentralized swarm robotics are security, trust within the swarm, an approach to enhance global knowledge, and distributed decision making (Ferrer, 2020).

The smart contract is a powerful tool in blockchain technology. Smart contracts are programs whose scripts, once deployed on the blockchain network, are tamper-proof and resilient to cyberattacks. This program runs on an Ethereum Virtual Machine, which is operated on all full nodes in the blockchain network. The program has functions invoked by transactions and uses a set of identical data from the blockchain ledger shared with all the nodes. Although execution is distributed to nodes throughout the network, the similarity in source code, invocations (transactions), and stored data (blockchain ledger) not only supports the consensus of the network, but also shows the potential to perform global knowledge synthesis and distributed decision-making.

There are approaches to exploiting the power of smart contracts in prior research. In Alexandre Pacheco's work in 2020, a smart contract is used to receive and synthesize result data reported by robots. This smart contract recognizes and rules out outliers in the reported values, and then rewards the accepted reports with 'cryptotokens', a currency that is required to send reports. In addition, in 2020, Grey proposed a more sophisticated method of task allocation and task completion judgment. The system includes robots with different roles of issuing tasks, performing tasks, and judging task completion.

This project develops a straightforward, data-driven decentralized system based on blockchain, where a smart contract has almost an equivalent responsibility and capability to a server in centralized systems. In addition, this project explores a different method to validate the completion of the work, which is based on distributed cross-checking within a swarm of homogenous robots. The goal of this system is to decentralize task allocation, task complete validation and real-time decision-making while maintaining high accuracy and performance.

II. Method

A. Experiment Scenario

The system in this experiment is dedicated to supporting the movement of goods in warehouses. The system is tasked with a list of goods movement tasks. A task is to move a goods 'G' from a location 'A' to a location 'B'. There are four locations in this simulation with a random number of unique goods there. And, the swarm contains three Turtlebots. The goods in this experiment are virtual. The lists of goods and locations are stored on a cloud database. The robots modify the goods list in the database instead of loading/unloading physical goods.

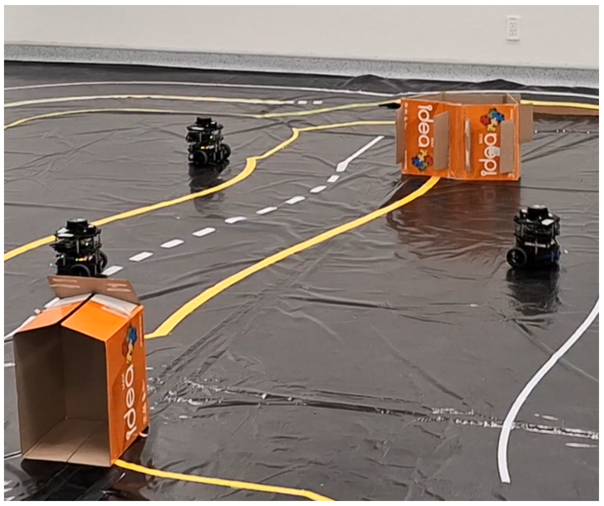


Fig. 1. Environment Setup.

The two main goals of this simulation are to allocate tasks to robots, and verify the completion of tasks. The target of task allocation is to delegate unassigned tasks to eligible robots in real time. This implies that, at a specific time, either there is no eligible robot or unassigned task, or the system is performing task allocation (1).

For task verification, the first requirement is that the validators are unpredictable until they report the inventory. The second one is that without external interference, any robot would report the correct state of the inventory. Assuming the above requirements are satisfied, the system ensures no false positive and a low chance of false negative. Finally, the credit point reward and deduction are correct based on the judgment. These goals ensure the verification system is decentralized, correct and fair. In this way, the system aims to quickly disable any malicious robots attempting to harm the system.

B. Task Allocation

The core of task allocation is the ’assign’ function in the smart contract. ’assign’ is an internal function, which means it can only be invoked by other functions of the smart contract. Therefore, the task allocation process is entirely under the control of the smart contract logic and is protected from any external assaults.

There are two functions in the smart contract that can trigger task allocation. They are ’addTask’, the only case of creating an unassigned task, and ’updateTaskStatus’, the only case of making a busy robot free. Adding the two facts that the system starts with no unassigned task, and after task allocation, either there is no unassigned task or no free robot, the above goal statement of task allocation (1) can be reached by this design.

Additionally, round-robin is used as a minimal method to reach equal task distribution. The result of the round-robin in this system can be stated as follows: ”When distributing a set of tasks to a set of free robots, the last robot to be assigned a task has the lowest priority”. In terms of tasks, the first task to be added is the first task to be assigned to robots.

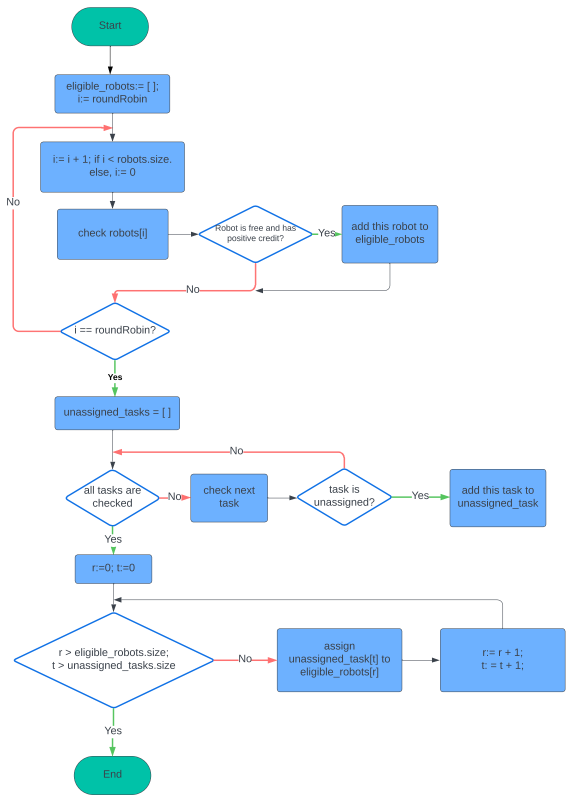
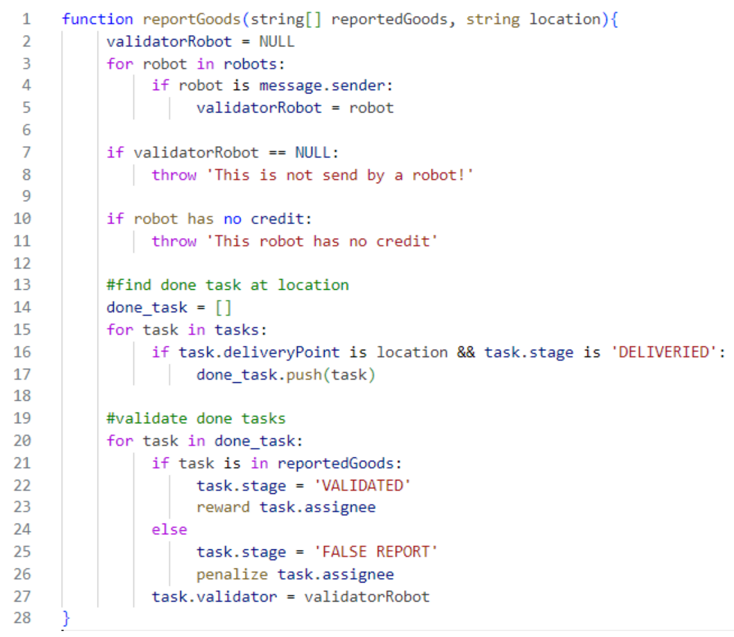

Start
eligible_robots:= [ ];
i:= roundRobin
i:= i + 1; if i < robots.size.
else, i:= 0
No
check robots[i]
Robot is free and has positive credit?
Yes
add this robot to eligible_robots
No
i == roundRobin?
Yes
unassigned_tasks = [ ]
No
all tasks are checked
No
check next task
task is unassigned?
Yes
add this task to unassigned_task
Yes
r:=0; t:=0
r > eligible_robots.size;
t > unassigned_tasks.size
No
assign unassigned_task[t] to eligible_robots[r]
r:= r + 1;
t: = t + 1;
Yes
End

Fig. 2. Flowchart of ’assign’ function.

C. Task Completion Validation

The process of task completion validation is triggered by any robot when it arrives at a location by invoking the function ’reportGoods’ in the smart contract. This function’s inputs are the location and the list of goods in there reported by the robot. For easier mentioning, the robot that calls the function ’reportGoods’ will then be called the validator. The function uses the validator’s input to check against the tasks which are reported by their assignees to be completed at the validator’s location. If the assignee had reported the task was completed but the corresponding goods item is not found in the validator’s report, that task would be marked as a ’false report’, and the assignee would be penalized with 2 credit points. Otherwise, if the two reports match, the assignee is rewarded with 2 credit points.

The credit point is an integer stored within the information of each robot in the smart contract. A robot needs a positive credit point to be assigned a task. And when a robot is assigned a task, 1 credit point is taken from that robot. If the robot is validated to have completed a task, it earns 2 credit points. If it submits a false report and that is detected, it is penalized with 2 points.



```
function reportGoods(string[] reportedGoods, string location){
    validatorRobot = NULL
    for robot in robots:
        if robot is message.sender:
            validatorRobot = robot

    if validatorRobot == NULL:
        throw 'This is not send by a robot!'

    if robot has no credit:
        throw 'This robot has no credit'

    #find done task at location
    done_task = []
    for task in tasks:
        if task.deliveryPoint is location && task.stage is 'DELIVERIED':
            done_task.push(task)

    #validate done tasks
    for task in done_task:
        if task is in reportedGoods:
            task.stage = 'VALIDATED'
            reward task.assignee
        else
            task.stage = 'FALSE REPORT'
            penalize task.assignee
        task.validator = validatorRobot
}
```

Fig. 3. Pseudo code of 'reportGoods' function.

The system is designed this way so the robot reports the goods in its location without being aware of which task it is judging and which robot is affected by its report. Moreover, the validator of a task is not predictable before it performs the validation because it depends on a lot of factors: task allocation algorithm, the order of the tasks, the time when the tasks are issued, the locations of robots at a certain time, etc. And some of them are unpredictable. This design tries to make the selection of validators become stochastic and random, aiming for the ideal distributed task validation. This design enhances the objectivity of the judgments and minimizes the chance that one malicious robot can corrupt the system.

D. Other functions

Some other important functions of the smart contract to complete the flow of the system are:

1. 'addTask' – this function is used by human admin to add a new task to the system.

2. 'getOwnTask' – this function is called by any robot to get its current task.

3. 'updateTaskStatus' – any robot uses this function to update its task's status when it has completed a certain part of its task.

4. 'readTasks' and 'readRobots' are functions used in the admin interface to view the current state of the system.

The 'addTask' function is an external one that takes input from the user interface. The input includes three vital parameters: a goods ID, a pickup location, and a delivery location. The function uses that input to create a Task object and add to the 'tasks' variable of the smart contract. As a variable in the smart contract is modified, a transaction and a block are created.

The 'getOwnTask' function is an external one used by the robots. This function takes no input, but it uses the information of the robot, which is called the function (the message sender), to return the task which is currently assigned to that robot.

The 'updateTaskStatus' function is an external function used by robots as assignees of a task, to report when it has done a stage of that task. In this project, a task has 8 stages: unassigned, assigned, started, received (goods), delivered (goods), (task) validated, false report, and error. However, the 'updateTaskStatus' only capable of switching to 4 stages, which are started, received, delivered, and error. In case of an error reported, smart contract uses its global knowledge to best identify the error before switching to stage 'error'. This 'updateTaskStatus' function's capability is designed to restrict the access of assignees to its task.

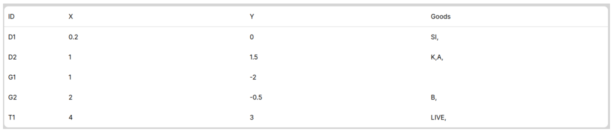
The 'readTasks' and 'readRobots' are used exclusively by the admin to monitor the current state of the system.

III. Findings

Firstly, a comparison between the performances of a single robot and a swarm of three robots is made. The average time for a single robot to complete three tasks in a row is 233 seconds, with an average of 78 seconds for each task. Meanwhile, the three robots swarm complete three tasks in parallel on an average of 77.5 seconds with a standard deviation of 4.034 (seconds). This means the swarm runs 3.02 times faster than one robot system. This result of 3.006 logically may show a little bias caused by the difference in the start point of the robots and the tasks' positions because the number of tests is small (ten times for each). As the number of tests increases, the ratio should fluctuate closer to 3 as the three robot systems in the three task tests are simply three single robots running one task each. However, when the number of tasks increases, the average time for each task of the single robot remains stable, while the average time of a set of three tasks for the swarm increases by 9 seconds. This is because after submitting delivery time to the smart contract, the robots move away from the position, and this action takes approximately 8 seconds for each robot (this number is measured by an external timer). This collision avoidance action is indeed not needed for a single robot system. So, in the long run, the speed ratio between the three robots swarm and the single robot is approximately 2.693.

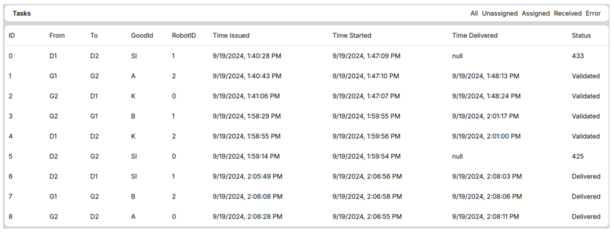
Secondly, the success rate of tasks is discussed. The action of adding new tasks to the network has a success rate remaining 100% until the time of this writing. The same success rate is shown when assigning tasks to eligible robots and when receiving and starting the assigned tasks. However, the rate of the tasks that are started to be completed is only guaranteed at more than 75%. The main reasons for failed tasks are timeout (80%) and exceeding the path planning retry limit (20%). The root of those errors comes from the physical interaction of robots, which is largely unpredictable, even though a lot of effort is made to prevent runtime interference between robots.

The validation of complete tasks and the accounting of robots' credit witness great accuracy with a zero percent of false positives, which means no false completion report is accepted, and the robot responsible for the false information is penalized with the correct credit point. False negatives, on



ID	X	Y	Goods
D1	0.2	0	SI,
D2	1	1.5	K,A,
G1	1	-2	
G2	2	-0.5	B,
T1	4	3	LIVE,

Fig. 4. The location view in admin interface.



Tasks

ID	From	To	GoodId	RobotID	Time Issued	Time Started	Time Delivered	Status
0	D1	D2	SI	1	9/19/2024, 1:40:28 PM	9/19/2024, 1:47:09 PM	null	433
1	G1	G2	A	2	9/19/2024, 1:40:43 PM	9/19/2024, 1:47:10 PM	9/19/2024, 1:48:13 PM	Validated
2	G2	D1	K	0	9/19/2024, 1:41:06 PM	9/19/2024, 1:47:07 PM	9/19/2024, 1:48:24 PM	Validated
3	G2	G1	B	1	9/19/2024, 1:58:29 PM	9/19/2024, 1:59:55 PM	9/19/2024, 2:01:17 PM	Validated
4	D1	D2	K	2	9/19/2024, 1:58:55 PM	9/19/2024, 1:59:56 PM	9/19/2024, 2:01:00 PM	Validated
5	D2	G2	SI	0	9/19/2024, 1:59:14 PM	9/19/2024, 1:59:54 PM	null	425
6	D2	D1	SI	1	9/19/2024, 2:05:49 PM	9/19/2024, 2:06:56 PM	9/19/2024, 2:08:03 PM	Delivered
7	G1	G2	B	2	9/19/2024, 2:06:08 PM	9/19/2024, 2:06:58 PM	9/19/2024, 2:08:06 PM	Delivered
8	G2	D2	A	0	9/19/2024, 2:06:28 PM	9/19/2024, 2:06:55 PM	9/19/2024, 2:08:11 PM	Delivered

Fig. 5. The task view in admin interface.

the other hand, have never happened, though it is logically possible. This is a logical failure in the verification system that allows false negatives. The issue is found late in the development, so even if the solution is known, the limited time prevents its implementation.

The scenario that allows false negatives is as follows:

Robot 1 delivers goods A to G1. Then, before any other robots reach G1, robot number 1 is assigned with the task of delivering goods A from G1 to D1. It comes to G1 and reports the inventory to the smart contract, but the smart contract does not allow robot number 1 to validate its own task. Afterwards, Robot 1 still picks up the goods A and brings it to D1. After that, Robot 2 comes to G1 and reports that good A is nowhere to be found in the inventory. This leads to the smart contract accusing Robot 1 of reporting false information and deducting its credit.

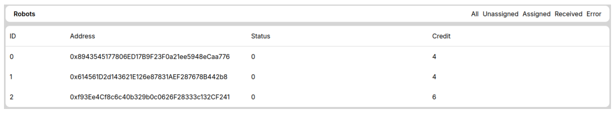
The solution to this could be to prevent a robot from being assigned to a task involving goods from its previous non-verified tasks. This algorithm could be implemented in the smart contract

IV. Discussion

The findings on robot performance proves one of the core goals of a swarm robotics system, which is to increase the performing speed by increasing the size of swarm, can be reached with this design. However, an experiment on a larger swarm should be carried out to develop a performance formulation, which depends on the size of the swarm as the variable.

The findings indicate a data-driven decentralized system with a smart contract being abstracted as a centralized controller performs accurately and efficiently. This path of design has the potential to be explored in further research, which may increase the swarm size or bring swarm robotics closer to practical usage.

This validation scheme shows possibilities in its early form as it works truly to the expected characteristics of objectivity, correctness and the unpredictability that brings decentralization. However, many flaws must be tackled before it can feature in a trustworthy, fully autonomous swarm robotics



Robots

ID	Address	Status	Credit
0	0x8943545177806ED17B9F23F0a21ee5948eCaa776	0	4
1	0x614561D2d143621E126e87831AEF287678B442b8	0	4
2	0xf93Ee4Cf8c6c40b329b0c0626F28333c132CF241	0	6

Fig. 6. The robot view in admin interface.

system. In this design, it is largely possible for an attack to cause chaos or to disable the swarm by spamming false reports saying there are no goods in every location. Therefore, new strategies should be developed to recognize, punish and prevent mass or random false judgements by malicious robots as validators. Experiments with predesigned numbers or percentages of malicious against various swarm sizes are needed to further prove the method.

V. Limitations and Future Research

There is plenty to be done to develop a completely distributed system of task allocation and task validation for swarm robotics.

For the hardware part, this system relies on campus Wi-Fi as a communication medium. That is not optimal because the availability of the system relies on the availability of the Wi-Fi router. Different decentralized means of communication should be developed. Installing a wireless access point on each robot to generate local networks can be a solution. This issue was mentioned in Pacheco's research in 2020, and his approach to this is worth referring to.

Regarding software, there is a huge space to improve the smart contract algorithm. The performance of the system can be boosted by assigning tasks to the nearest robot to the pickup location. The smart contract can be programmed to support collisions avoidance of robots. Moreover, the failed tasks can be reassigned to other robots a few times before being considered undoable. Additionally, the validation method can be improved to be more reliable.

A deeper research into hardware can be conducted so that the system can interact with physical objects by adding robot hands into the system. Interactions between Turtlebots and robot hands can be processed by smart contracts and blockchain as well. An upgrade in software is also needed to put the system into practice. Each robot can be equipped with a camera, and its software can be upgraded to scan QR codes on goods and identify them.

VI. Conclusion

This study demonstrates the potential of integrating blockchain technology and smart contracts with swarm robotics to support goods movement in warehouses. The proposed decentralized system achieved its goal of distributed task allocation and accuracy in distributed task completion validation, both through a data-driven approach. Despite these successes, there are challenges ahead, such as the low task completion rate because of physical interference between robots, limitations in communication infrastructure, and the incompleteness of the task validation method. In order to address these issues, future research should focus on enhancing the smart contract's algorithm, developing decentralized communication methods, and exploring new validation strategies to counter malicious behaviour by validators. Additionally, increasing the swarm size and conducting experiments with pre-defined numbers of malicious robots will provide further insights into the robustness and scalability of the system. With these improvements, the system has the potential to advance towards a fully autonomous and reliable swarm robotics solution suitable for real-world applications.

References

[1] Eduardo Castelló Ferrer. The Blockchain: A New Framework for Robotic Swarm Systems. In *Proceedings of the Future Technologies Conference (FTC) 2018*, pages 1037–1058. Springer International Publishing, 2019.

[2] Jonathan Grey, Isuru Godage, and Oshani Seneviratne. Swarm Contracts: Smart Contracts in Robotic Swarms with Varying Agent Behavior. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 265–272, October 2020.

[3] Alexandre Pacheco, Volker Strobel, and Marco Dorigo. A Blockchain-Controlled Physical Robot Swarm Communicating via an Ad-Hoc Network. pages 3–15. October 2020.

[1] [2] [3]