Real-Time Coordination of a Foraging Robot Swarm Using Blockchain Smart Contracts

Alexandre Pacheco(✉), Volker Strobel, Andreagiovanni Reina, and Marco Dorigo

IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
{alexandre.melo.pacheco,volker.strobel,andreagiovanni.reina, marco.dorigo}@ulb.be

Abstract. We present a novel control scheme for robot swarms that exploits the computation layer of a blockchain to coordinate the actions of individual robots in real-time. To accomplish this, we deploy a blockchain smart contract that acts as a "decentralized supervisor" during a swarm foraging task. Our results show that using blockchain-based global coordination rules can improve the foraging behavior of robot swarms, while maintaining a decentralized, scalable, and democratic system in which every robot contributes homogeneously to the decision-making process.

1 Introduction

The application of blockchain technology to robotic systems is a fast growing research topic. Particularly, in swarm robotics, the most noteworthy advancement was the recent introduction of a blockchain in order to achieve secure consensus in the presence of Byzantine agents: in [19,27], it was shown that blockchain-secured robot swarms can be deployed in situations where security against unauthorized agents is paramount.

The introduction of a *decentralized* and *secure* database such as the blockchain might have a strong impact on the field of swarm robotics. However, further research is required to understand the extent of this impact, as well as its potential drawbacks.

Ethereum [3] extended the application of blockchains from financial ledgers to decentralized computing platforms. This means that the participants in the Ethereum network can agree not only on the execution of financial transactions, but also on the execution of computer programs known as *smart contracts*.

In this paper, we argue and validate the claim that smart contracts can be very valuable when applied to the real-time coordination of robot swarms. In this context, a smart contract is control code that is executed in a decentralized manner by the swarm; that is, each robot executes the code independently and the swarm comes to an agreement on its output. On a micro perspective, the individual robots collect local information and deliver it to the smart contract by broadcasting local messages. On a macro perspective, the smart contract extends

M. Dorigo et al. (Eds.): ANTS 2022, LNCS 13491, pp. 196–208, 2022.
https://doi.org/10.1007/978-3-031-20176-9_16

the swarms' ability to self-organize by aggregating the input of the robots and returning action policies on which the robots can act in real-time.

To demonstrate this, we deploy a blockchain smart contract to act as a "decentralized supervisor" during a collective foraging task in which the swarm needs to collect resources spread in an unknown environment. The robots broadcast messages—known as *transactions*—that contain information the robots obtained from scouting the environment for resources and that should be included in new blocks of the blockchain. The information about the environment contained in these transactions is aggregated by the smart contract into a shared database of resource locations. The blockchain consensus protocol guarantees that these transactions are executed orderly and conflict-free, and that all robots reach an agreement on the most recent state of this database. Furthermore, the smart contract distributes the available robots (recruits) to the various resources, while (i) prioritizing resources with better quality; and (ii) limiting the number of foragers per resource. These simple rules are shown to increase the resource collection rate and energy efficiency during the task. As consensus protocol we use proof-of-authority [29], which we have shown in previous research [19] to be suitable for robot swarms since it requires low power and is robust to network partitioning and temporary unavailability of robots.

The rest of the paper is organized as follows. In Sect. 2, we review related works. In Sect. 3, we introduce the foraging task, the environment, and other methods relevant for the implementation of the experiments: the simulations software, the robot's model, the blockchain, and the robot controllers. In Sect. 4, we present and discuss the experimental results. In Sect. 5, we deliver the conclusions of this study.

2 Related Work

Cooperation in Foraging Robot Swarms. Foraging is one of the most studied behaviors in swarm robotics because it models a wide range of application scenarios, such as search and rescue, agriculture, mining, waste cleaning, and planetary exploration. It can be described as the combination of two sub-tasks: searching the environment for objects, and performing actions on those objects (e.g., transportation, consumption, destruction, ...). In this work we focus on central place foraging [13], where agents are tasked with finding and transporting objects back to a target location (called "nest").

Inspired by the foraging behavior of ants, which deposit pheromones along paths leading to objects [5], robot swarm algorithms are most frequently based on indirect communications (stigmergy). Researchers have attempted to mimic ant behavior by using chemicals [24]; augmented reality pheromones [10]; and virtual pheromones, which are advertised locally by robots with the role of pheromone beacons [4,12,17]. The main advantages of these methods are scalability and robustness; however, their implementation either requires specific equipment and infrastructure (e.g., a smart environment or sensors/actuators for chemicals) or reduces efficiency by allocating part of the swarm to play the role of beacons.

Additionally, it has recently been shown that stigmergy is particularly fragile when malicious agents are present [1].

For these reasons, some researchers have employed forms of explicit communication to coordinate the collective foraging behavior, inspired by the recruitment dances that honeybees perform to signal foraging locations to peers [2,25].

Pitonakova et al. [21,22] compare swarms where robots recruit other robots at the nest with swarms of individualist foragers. They show that when resources are scarce or difficult to find, nest-site recruitment can be helpful to maximize the total resources collected. Conversely, if resources are abundant, it may be more advantageous to forage individually as this might prevent both *physical interference* (when robots foraging for the same resources collide) and *informational interference* (when robots are misguided by incorrect social information). Despite this insight, no coordination strategy nor methodology to enable the validation of the information are proposed in order to limit or reduce interference.

Applications of Blockchain to Swarm Robotics. The application of blockchain technology to robot swarms was demonstrated for the first time in [26–28], where the authors presented a proof-of-concept (in simulation) showing how blockchain-based smart contracts can be used to neutralize the negative effects of Byzantine robots in a consensus problem. In [19] the authors presented the first implementation of a blockchain in a swarm of real robots using proof-of-authority consensus [3,29], which is shown to be suitable to robot swarms given that it is energy efficient and robust to network partitioning.

Although these studies showcase the promise of smart contracts to achieve generic swarm-wide agreements, it is not yet clear whether the network consensus delay is too large to allow a wider range of applications—particularly, real-time control. Some researchers have presented control architectures in which the blockchain is maintained outside of the robot's network [9]. This design is akin to using an external control element (albeit, a distributed one in this case), and does not grant the autonomy and fault-tolerance properties warranted in a robot swarm. In this paper, we present a decentralized and autonomous robot swarm that uses blockchain smart contracts for real-time coordination.

3 Methods

Task. The goal of the swarm is to retrieve resources from the environment and deposit them at the nest. Resources have various qualities that yield a different reward when deposited. The performance of the swarm is measured in terms of the total *reward* collected, and of the *scouting efficiency*, which is the ratio between the reward collected and the distance traveled by the robots while exploring the environment. Each experiment lasts 15 min.

Environment. The environment consists of a square arena with the nest located at the center. The size of the arena is a function of the number of robots (i.e., we maintain a constant robot density of 3 robots per m^2), and the nest occupies

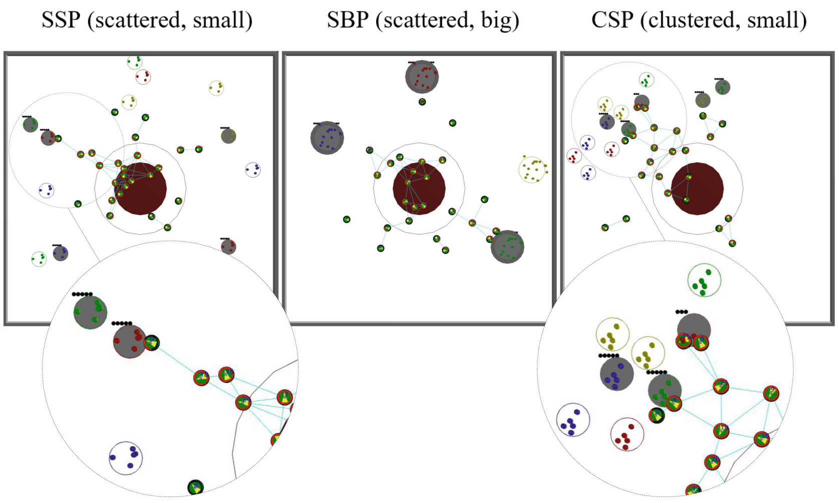

Fig. 1. A frame from a simulation run for each resource patch distribution: *SSP* (left), *SBP* (middle) and *CSP* (right). The patches are circles with items inside (the resources). A gray background means that the patch is included in the blockchain database, and the black dots above represent the remaining quantity of resources according to the blockchain database. The brown circle and annulus in the center are the nest and its deposit area, respectively. (Color figure online)

10% of the arena's total area. The nest is divided into two areas (Fig. 1): an external annulus, where robots can deposit resources; and an internal circle, where robots can idle. The nest broadcasts a homing signal which allows the robots to navigate to the nest from any location.

Resource *patches* are circular areas distributed randomly in an annulus centered on the nest and with radiuses 0.83 m and 1.44 m. *Resources* are individual items contained in a patch that the robots can collect and deposit at the nest. The patches can be of 4 different types (red, green, blue and yellow), and the resources collected from each type yield a different reward (2, 4, 6 and 8, respectively). Once a patch runs out of resources, an identical patch spawns elsewhere.

We consider three *distributions* of patches and resources in the environment. In all distributions, approximately 3% of the environment area is covered with patches, and there is an identical number of red, green, blue and yellow patches.

- **Scattered small patches (SSP)** The patches are distributed uniformly in the annulus, have a diameter of 16cm and contain 10 resources (Fig. 1, left).
- **Scattered big patches (SBP)** The patches are distributed uniformly in the annulus, have a diameter of 36cm and contain 15 resources (Fig. 1, middle).
- **Clustered small patches (CSP)** The patches are distributed according to a normal distribution that is biased towards the upper left quadrant of the arena, have a diameter of 16cm and contain 10 resources (Fig. 1, right).

Simulation Setup. The simulation setup consists of the swarm robotics simulation software ARGoS [20]; the blockchain software Ethereum [3]; and the virtualization software Docker [14]. The nodes of a custom Ethereum network are executed in Docker containers. Each ARGoS robot controller is associated with an Ethereum node, and can interact with the client application software (geth). In this way, ARGoS interacts with the client-side of Ethereum, while the maintenance of the blockchain is handled by the Docker containers.

We use Python wrappers for both ARGoS [11], and geth [8]. This allows the robot control routines and interactions with the blockchain client to be fully written in Python. Our code is available online [18].

Robot Model. The agent used in the simulations is a model of the *Pi-puck robot* [15]. In previous research, we showed that the Pi-pucks are capable of executing the blockchain software [19]. In order to perform the foraging task, the Pi-pucks use infrared sensors for obstacle avoidance; a range-and-bearing board for local peer discovery; a ground sensor for scouting resource patches; and two motors for locomotion. The manipulation of resources is not modeled.

Blockchain Protocol. For a thorough understanding of blockchain technology, we refer the readers to the papers on Ethereum [3] and Bitcoin [16]. Here we focus on the two components of blockchain technology which are most relevant for this work: *consensus protocols* and *smart contracts*.

The *consensus protocol* consists of the rules used by a blockchain network to agree on the addition of new blocks of information to the blockchain. In situations of conflict (known as blockchain *forks*), it also establishes the rule that defines what becomes the current accepted state of the blockchain. To accomplish this, proof-of-work, the original blockchain consensus protocol introduced with Bitcoin [16], requires the expenditure of computational resources. As such, it is often considered contraindicated for swarm robotics applications [23], which typically consider robots with limited capabilities and resources [6].

In our research, we have decided to use proof-of-authority [29] as an alternative to proof-of-work. Proof-of-authority keeps a core of authorized and accountable nodes which share the role of producing new blocks. In this protocol, anyone can create a block and propose it to be added to the chain, but in order to be considered a valid block three conditions must be met: (i) the difference between the timestamp of two consecutive blocks must be at least $t = T_b$ seconds (T_b is called the *block period*); (ii) the block must be correctly signed by an authorized node (known as a "sealer") using its private key; and (iii) a sealer can only sign one block every $\lfloor \frac{N}{2} \rfloor + 1$ blocks (N is the number of sealers).

Every node in the network checks if a proposed block meets these conditions. If this is the case, the node appends that block to its local copy of the blockchain. The consensus protocol establishes that the current version of the blockchain is the one which has the highest *cumulative difficulty*. Blocks which are signed *in-turn* (i.e., that are signed by an appointed preferred sealer for that block), contribute with a difficulty of 2; while other blocks contribute with 1.

When deploying a robot swarm it is important to consider that: (i) some robots may be unavailable when the network is partitioned; and (ii) some robots may join or leave the swarm during its operation. In the first situation, it is possible that the robots disconnected from the partition hosting the main blockchain (which has the highest cumulative difficulty) operate on a different version of the blockchain (i.e., a blockchain fork). Eventually, when the partitions reconnect, the main blockchain is established by consensus and the transactions included in the fork are rebroadcast. In the second situation, we note that the proof-of-authority consensus protocol allows current sealers to democratically elect or remove sealers, thus allowing for dynamic swarm sizes. In this paper, however, we maintain constant swarm sizes and every robot is a sealer throughout the duration of the experiment.

A *blockchain smart contract* is a computer program that is stored on the blockchain, and that encapsulates code (its functions) and data (its state). Network participants can execute its functions by broadcasting transactions to the smart contract address, which in turn will change its state. It is the role of the blockchain system to agree on the irrefutable execution of these state-altering transactions in a decentralized manner.

Our smart contract allows robots: (i) to store information regarding discovered resource patches; (ii) to enlist themselves as recruits in order to forage at a certain patch; and (iii) to query information about the known resource patches. Its programming code ensures that the information the robots provide is synchronized without conflicts; that the highest-reward resources are prioritized for foraging; and that there is a limit on the number of foragers per patch.

The robots can interact with functions by broadcasting *transactions* (to execute the function on the blockchain network), or by invoking *calls* (to execute the function locally and read its output). Our smart contract has 3 functions:

- `update_patches(patches[])` The input is a list of formatted strings which contain the relevant information about a patch: position, radius, quality, and quantity of resources. If the position is unique (within an error margin) a new resource is added to the database, otherwise an existing resource is updated.
- `assign_patch()` If there are available patches (i.e., patches with fewer foragers than the maximum number allowed), then the transacting robot is assigned as a forager to the highest quality patch.
- `query_patches()` Returns a database of resources, including the current foragers for each resource.

Robot Controller. The robots are controlled by a *finite-state machine*. At each simulation step, the robots perform a routine corresponding to their current state, as well as a *local peer discovery* routine.

The *finite-state machine* starts at the state `Scout` and is composed of 5 states:

- `Idle` Wait for 30 s; then, transition to `Scout`.
- `Scout` Perform a random-walk, with a duration sampled from $\mathcal{N}(\mu = 40$ s, $\sigma = 2$ s) and store the discovered patches in a list stored locally; then, broad-

cast a transaction to execute `update_patches(scouted_patches)`. Once the transaction is included in a block, delete the list and transition to `Plan`.

- `Plan` Return to the nest using the homing signal and invoke a call to `query_patches()`. If assigned to forage a resource, transition to `Search`; otherwise, broadcast a transaction to execute `assign_patch()`, and wait until it is included in a block. If the transaction fails (no resources available to forage), transition to `Idle`.
- `Search` Navigate from the nest towards the direction of the assigned patch and search its neighborhood for 10 s. If resources are found, transition to `Forage`; otherwise broadcast a transaction to execute `update_patches(depleted_patch)` and transition to `Scout`.
- `Forage` Collect a resource from the patch and navigate to the nest using the homing signal. Then, broadcast a transaction to execute `update_patches(current_patch)` to inform that one resource was removed. Once the transaction is included in a block, deposit the resource and, if there were more resources, transition to `Search`; otherwise, transition to `Scout`.

The *local peer discovery* routine enforces that all communications, including blockchain synchronization, occur locally (up to 30 cm). Within this range, the robots broadcast and receive IP addresses using infrared signals on the range-and-bearing board. After receiving an IP address, robots use TCP to share their *enode*—a unique URL used to identify and connect to nodes in the Ethereum network. If the infrared signal is lost, the robot disconnects from that peer on the blockchain network and deletes its IP address and enode from its local memory. This peering scheme serves two purposes: (i) to ensure that communications are only local and thus mimic a real-world swarm deployment where network partitioning can occur; and (ii) to provide an additional layer of security which prevents external agents from participating in the network (since the robots reject connections which are not accompanied by the short-range infrared greeting).

4 Results and Discussion

In general, our goal is to show that a blockchain can extend the swarm's ability to self-organize, and thus improve its collective performance, while maintaining the properties of a robot swarm: *decentralization*, *scalability* and *adaptability*.

The blockchain allows robots to agree on the state of the environment and on a coordination strategy, without the need for delegated supervisors (in contrast with centralized or hybrid control). Since the proof-of-authority consensus algorithm is robust to the unavailability of up to 50% of the network nodes, our blockchain-coordinated robot swarm does not have a singular point-of-failure and could be deployed in situations where a system that relies on information traveling to and from supervisors would fail (for example, in environments with limited or no communication infrastructure). In this sense, a blockchain enables a decentralized and democratic swarm, in which all robots contribute homogeneously to the decision-making process.

On the downside, it is important to analyze the impact of *consensus latency*, i.e., the time it takes for messages to be disseminated through the network and for robots to reach agreements in this democratic process—as well as the costs of *data storage*, since each robot keeps a local copy of the blockchain database. These aspects could raise scalability concerns in terms of communication and hardware requirements for robot swarms. In Sect. 4.1 we discuss these concerns, and show that they are manageable for swarms of different sizes.

In foraging, cooperation is not always an advantage [22]. Sharing information can lead to an increased rate of physical interference, for example, when the robots forage the same resources rather than finding a balance between exploitation and exploration. It may also lead to informational interference, which occurs when robots propagate incorrect or outdated information (e.g., if a resource patch becomes depleted during the time the information is being processed, or if the robots' sensors provide inaccurate positions).

The role of our smart contract supervisor is to improve the performance of the swarm (in terms of the *reward* collected and the *scouting efficiency*) by aggregating information about resource patches from the robot scouts, and assigning resource patches to robot recruits—thus minimizing the impact from both forms of interference. In Sect. 4.2 we report the performance results of a blockchain-coordinated robot swarm and we compare them to those obtained with a swarm of uncoordinated robots, which explore the environment and forage resources as they discover them individually, in environments with different resource distributions. In these experiments, we keep the swarm size constant (25 robots) and analyze how performance changes as the maximum number of foragers that the smart contract allows per patch increases.

4.1 Scalability

Consensus Latency. Figure 2 (left) shows the *Block Reception Delay*, which is the difference between the timestamp at the moment a robot receives a block and the timestamp at the moment the block was produced (in other words, the time it took for a block to be disseminated through the network from its producer to any other robot). Figure 2 (right) shows the *Block Production Delay*, which is the difference of the timestamps between two consecutive blocks on the final version of the blockchain. The first metric is calculated online by the robots, while the second is calculated offline after the experiment is finished.

The *block period* (T_b) parameter sets the minimum required difference between the timestamps of two consecutive blocks (see Sect. 3), and thus has a big effect on the information delay introduced by the blockchain: if it is too high, it reduces the possibility to employ the shared knowledge to perform time-critical tasks. Conversely, if it is too low, it increases the frequency of block production which leads to (i) higher costs of communication, computation, and data storage; and (ii) an increased rate of blockchain forks which contain redundant, or more dangerously, conflicting information. In Fig. 2 (left) we observe that a majority of blocks are received within 2 s. This observation justifies our choice of $T_b = 2\,\mathrm{s}$,

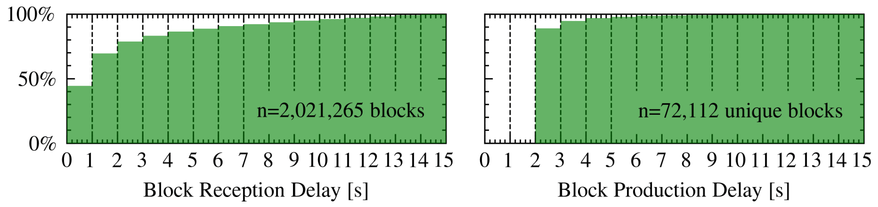

Fig. 2. The histograms represent cumulative probability distributions, and are generated from the combined data of all experiments performed in this study. Left: In 70% of the instances a robot received a block, that block was produced less than 2 s earlier; and in 100% of the instances, less than 15 s earlier. Right: The minimum and ideal production delay is equal to $T_b = 2\,\text{s}$. An additional delay occurs due to network delays (e.g., temporary unavailability of the preferred block producer). In our experiments, 90% of the blocks were produced within 2 s to 3 s, which means that the blockchain is operating as designed.

as there is a high chance that the previous block has been disseminated through the network before it is time to produce the next block.

Data Storage. In previous research [19], we set the block period to 15 s. With a block period of 2 s, we expect that the cost of storing the blockchain will be higher since the amount of data stored depends on the number of blocks created (as well as on the number of transactions performed by the robots).

Figure 3 (left) shows the data storage required by each robot, which is seen to increase linearly with the number of robots in the swarm. On average, each robot requires 8 MB for 15 min of operation, which we consider reasonable given current data storage technology. Furthermore, the robots in our experiments are *full blockchain nodes*, i.e., each robot stores the complete blockchain history. In a real deployment this might not be necessary, since only the most recent information is relevant for the robots' operations, and the task of storing the blockchain history can be delegated to external agents when connection is available, or it can be segmented and stored by the robots in a distributed manner. In this case, the hardware-limited robots would host *light blockchain nodes* [7], while remaining able to verify the status of the blockchain and of the transactions by leveraging cryptographic primitives. For these reasons, we do not expect data storage to pose a scalability problem in a real deployment.

Performance. Figure 3 (right) shows that the swarm is capable of maintaining performance (the total reward collected increases with the number of robots) as the environment size, number of robots and quantity of resources scale accordingly. However, we also observe decreasing performance returns (the total reward collected increases sublinearly with the number of robots). Rather than a limitation of our blockchain-coordinated approach, this seems to occur due to the layout of the environment, which is prone to interference at the centrally located nest when the swarm size increases.

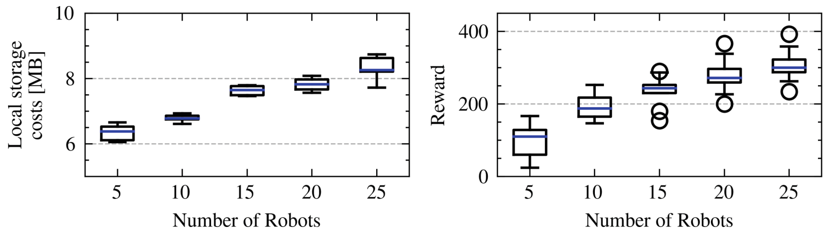

Fig. 3. Left: The storage space required for each robot grows linearly with the number of robots, at a rate of approximately 1 MB per 10 robots. Right: The collected reward grows sublineary with the number of robots, due to the increasing rate of physical interference at the centrally located nest. These experiments were repeated 25 times using the SSP distribution.

4.2 Performance in Different Distributions

SSP Distribution. In this environment there is a large number of patches randomly spread on the map. Previous research [22,30] indicates that individualist foragers tend to perform well, or even better than cooperating robots (when the benefits of cooperation do not overcome the negative effects of interference). In Fig. 4 (left) the total reward collected saturates at 2 foragers per patch, but is consistently higher than the non-collaborating swarm ('NC' in the x-axis). The scouting efficiency can be significantly higher but also has a high spread. This happens because cooperating robots, when lucky, will discover higher quality patches and better allocate foragers to those resources.

SBP Distribution. The blockchain-coordinated swarm is capable to retrieve 50% to 100% more reward, and to be twice more efficient in scouting for resources, as seen in Fig. 4 (middle). In this environment, the advantage of coordination is more pronounced since (i) the patches last longer as they contain more resources, and (ii) they are larger in size and there is therefore less interference.

CSP Distribution. The blockchain-coordinated swarm is capable to retrieve more than double of the reward and be 2 to 5 times more efficient during scouting, as seen in Fig. 4 (right) than non-collaborating swarms. This occurs because the scouting robots which move in the direction of the resource cluster are very successful, while others robots do not find any resources. The ability to aggregate and share information prevents unsuccessful robots from idling or wasting energy performing redundant exploration. Conversely, given the tight aggregation of resources, the foraging efficiency quickly drops as the number of recruits increases above 3 due to physical interference between robots.

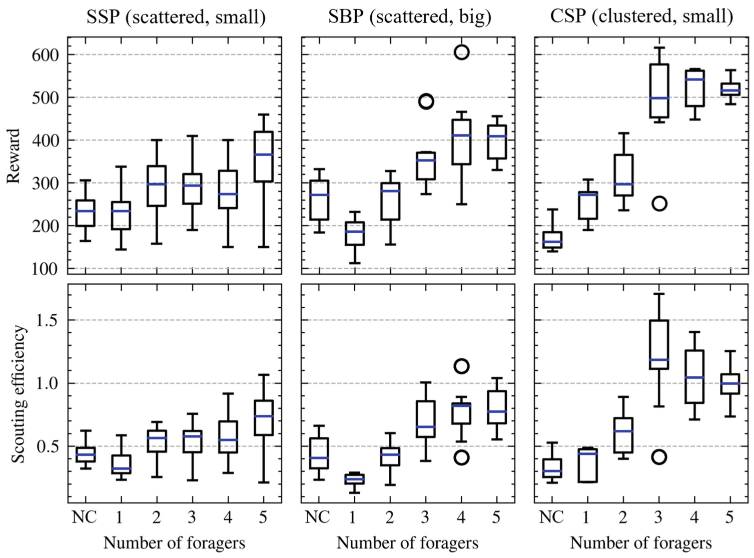

Fig. 4. Performance results for three distributions: SSP (left), SBP (middle) and CSP (right). The top row shows the reward collected by the swarm at the end of the experiment, and the bottom row the scouting efficiency. The x-axis (number of foragers) is a parameter in the smart contract which limits how many robots can be tasked as foragers for each resource patch. The uncoordinated robot swarm shows "NC". A swarm of 25 robots was used, and the experiments were repeated 10 times.

5 Conclusions

We showed that the coordination rules provided by a smart contract supervisor can improve the performance of the robot swarm during the foraging task, while keeping reasonable data storage costs and manageable delay in the control loop. These are positive results that showcase the potential of deploying blockchains for the real-time coordination of robot swarms in a wider range of scenarios.

The usage of a blockchain in a swarm robotics system enables a new class of distributed control algorithms that use explicit communication and coordination, while preserving decentralization and local exchanges of information. It is important to note the contrast between the macro perspective that is used when creating smart contract supervisors and the micro perspective that is more frequent in the design of robot swarm controllers. In our research, we present the two approaches as complementary since the behavior of individual robots emerges from local sensing and interactions, while the blockchain is regarded as an additional layer that is reserved for high-level decision making.

Acknowledgements. This work was partially supported by the program of Concerted Research Actions (ARC) of the Université libre de Bruxelles and by the Brussels-Capital Region via the Brussels International contract n. BI-MB-531-004021. A. Reina and M. Dorigo acknowledge support from the Belgian F.R.S.-FNRS, of which they are Chargé de Recherches and Research Director, respectively.

References

1. Aswale, A., López, A., Ammartayakun, A., Pinciroli, C.: Hacking the colony: on the disruptive effect of misleading pheromone and how to defend against it. In: Proceedings of the 21st International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2022), pp. 27–34. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2022)
2. Biesmeijer, J.C., de Vries, H.: Exploration and exploitation of food sources by social insect colonies: a revision of the scout-recruit concept. Behav. Ecol. Sociobiol. **49**(2), 89–99 (2001). https://doi.org/10.1007/s002650000289
3. Buterin, V.: A next-generation smart contract and decentralized application platform. Technical report, Ethereum Foundation (2014). https://github.com/ethereum/wiki/wiki/White-Paper. Accessed 18 July 2019
4. Campo, A., et al.: Artificial pheromone for path selection by a foraging swarm of robots. Biol. Cybern. **103**(5), 339–352 (2010). https://doi.org/10.1007/s00422-010-0402-x
5. Deneubourg, J.L., Aron, S., Goss, S., Pasteels, J.M.: The self-organizing exploratory pattern of the argentine ant. J. Insect Behav. **3**(2), 159–168 (1990). https://doi.org/10.1007/BF01417909
6. Dorigo, M., Birattari, M., Brambilla, M.: Swarm robotics. Scholarpedia **9**(1), 1463 (2014)
7. Ethereum Foundation: Ethereum project (2017). https://ethereum.org
8. Ethereum Foundation: ethereum/web3.py: A Python interface for interacting with the Ethereum blockchain and ecosystem (2022). https://github.com/ethereum/web3.py
9. Fernandes, M., Alexandre, L.A.: Robotchain: using tezos technology for robot event management. Ledger **4** (2019). https://doi.org/10.5195/ledger.2019.175. https://www.ledgerjournal.org/ojs/ledger/article/view/175
10. Font Llenas, A., Talamali, M.S., Xu, X., Marshall, J.A.R., Reina, A.: Quality-sensitive foraging by a robot swarm through virtual pheromone trails. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Reina, A., Trianni, V. (eds.) ANTS 2018. LNCS, vol. 11172, pp. 135–149. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00533-7_11
11. Hasselmann, K., Parravicini, A., Pacheco, A., Strobel, V.: KenN7/argos-python: python wrapper for ARGoS3 simulator (2022). https://github.com/KenN7/argos-python
12. Hoff, N., Wood, R., Nagpal, R.: Distributed colony-level algorithm switching for robot swarm foraging. In: Martinoli, A., et al. (eds.) Distributed Autonomous Robotic Systems, vol. 83, pp. 417–430. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-32723-0_30
13. Houston, A.I., McNamara, J.M.: A general theory of central place foraging for single-prey loaders. Theor. Popul. Biol. **28**(3), 233–262 (1985). https://doi.org/10.1016/0040-5809(85)90029-2
14. Merkel, D.: Docker: lightweight Linux containers for consistent development and deployment. Linux J. **2014**(239) (2014)

15. Mondada, F., et al.: The e-puck, a robot designed for education in engineering. In: Gonçalves, P.J.S., Torres, P.J.D., Alves, C.M.O. (eds.) Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, vol. 1, pp. 59–65. IPCB: Instituto Politécnico de Castelo Branco (2009)
16. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
17. Nouyan, S., Groß, R., Bonani, M., Mondada, F., Dorigo, M.: Teamwork in self-organized robot colonies. IEEE Trans. Evol. Comput. **13**(4), 695–711 (2009). https://doi.org/10.1109/TEVC.2008.2011746
18. Pacheco, A., Strobel, V.: teksander/geth-argos at ANTS2022. https://github.com/teksander/geth-argos
19. Pacheco, A., Strobel, V., Dorigo, M.: A blockchain-controlled physical robot swarm communicating via an ad-hoc network. In: Dorigo, M., et al. (eds.) ANTS 2020. LNCS, vol. 12421, pp. 3–15. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60376-2_1
20. Pinciroli, C., et al.: ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intell. **6**(4), 271–295 (2012). https://doi.org/10.1007/s11721-012-0072-5
21. Pitonakova, L., Crowder, R., Bullock, S.: Understanding the role of recruitment in collective robot foraging. In: Proceedings of the 14th International Conference on the Synthesis and Simulation of Living Systems (ALIFE 2014), pp. 264–271 (2014). https://doi.org/10.7551/978-0-262-32621-6-ch043
22. Pitonakova, L., Crowder, R., Bullock, S.: The information-cost-reward framework for understanding robot swarm foraging. Swarm Intell. **12**(1), 71–96 (2017). https://doi.org/10.1007/s11721-017-0148-3
23. Reina, A.: Robot teams stay safe with blockchains. Nat. Mach. Intell. **2**, 240–241 (2020). https://doi.org/10.1038/s42256-020-0178-1
24. Salman, M., Garzón Ramos, D., Hasselmann, K., Birattari, M.: Phormica: photochromic pheromone release and detection system for stigmergic coordination in robot swarms. Front. Robot. AI **7** (2020). https://www.frontiersin.org/article/10.3389/frobt.2020.591402
25. Seeley, T.D.: Division of labor between scouts and recruits in honeybee foraging. Behav. Ecol. Sociobiol. **12**(3), 253–259 (1983). https://www.jstor.org/stable/4599586
26. Strobel, V., Castelló Ferrer, E., Dorigo, M.: Managing Byzantine robots via blockchain technology in a swarm robotics collective decision making scenario. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2018), pp. 541–549. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2018)
27. Strobel, V., Castelló Ferrer, E., Dorigo, M.: Blockchain technology secures robot swarms: a comparison of consensus protocols and their resilience to Byzantine robots. Front. Robot. AI **7**, 54 (2020). https://doi.org/10.3389/frobt.2020.00054
28. Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Reina, A., Trianni, V. (eds.): ANTS 2018. LNCS, vol. 11172. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00533-7
29. Szilágyi, P.: EIP 225: clique proof-of-authority consensus protocol (2017). https://github.com/ethereum/EIPs/issues/225. Accessed 10 May 2020
30. Wilson, E.O.: Sociobiology: The New Synthesis, Twenty-Fifth Anniversary Edition. Harvard University Press, Cambridge (2000)