Distributed Task Allocation in Network of Agents Based on Ant Colony Foraging Behavior

Dorian Minarolli
dminarolli@fti.edu.al
Polytechnic University of Tirana
Tirana, Albania

ABSTRACT

Task allocation is an important problem that is encountered in different distributed computing environments such as grid and cloud computing. In this paper we take a multi-agent learning approach for task allocation in distributed systems composed of network of nodes where each node is connected to a fixed number of neighbor nodes. We developed an approach for distributed task allocation based on principles of ant colony foraging behavior. Ant colony systems as part of more general swarm intelligence systems are notorious for their self-organization, scalability, adaptability and robustness. Because of these properties these systems are very suitable for our distributed task allocation problem. We compared our approach with other heuristic based algorithms. Simulation results on different tasks lengths, load levels, node types and random network graphs show improved performance of our algorithm compared to other approaches. Especially, combining the ant colony based algorithm with an heuristic approach gives the best performance.

CCS CONCEPTS

• **Computing methodologies → Distributed computing methodologies**; **Bio-inspired approaches**.

KEYWORDS

distributed task allocation, multi-agent learning, task scheduling, cloud computing, ant colony optimization

ACM Reference Format:
Dorian Minarolli. 2023. Distributed Task Allocation in Network of Agents Based on Ant Colony Foraging Behavior. In *International Conference on Computer Systems and Technologies 2023 (CompSysTech '23), June 16–17, 2023, Ruse, Bulgaria.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3606305.3606324

1 INTRODUCTION

Task scheduling in distributed systems [10] is an important and well studied problem . Especially in todays cloud systems, scheduling algorithms [13] are important since the strategy to allocate task to different nodes of the system determine its performance. Of special interest is the allocation of tasks in a network of nodes where each node is connected to a fixed number of neighbor nodes and for each task that arrives, a decision has to be made on which node to allocate it, in order to optimize the system performance. This is inherently a distributed task allocation problem that can be encountered in a variety of real world distributed systems. For example in one possible scenario, geographically distributed computing clusters [15] are connected to each other in a graph structure forming a grid computing system. This scenario can be represented by a network of nodes where each node is a computing cluster and tasks arriving at each cluster are scheduled locally or transfered to a neighbor cluster if the local one is overloaded. In this scenarios there is a need for the allocation and distribution of tasks on different computing clusters in oder to optimize the system performance.

In this paper we view the distributed task allocation in a network of nodes as a multi-agent learning problem where each agent makes allocation decisions autonomously and independently of other agents. For every task that arrives at a node, the node agent, having only a partial view of the network, has to decide whether to allocate the task locally or forward it to a neighbor node. In order to learn agent decision policies we apply principles of ant colony foraging behavior. Ant colony systems are part of swarm intelligence systems [4] in which global intelligent behavior emerges from local interactions of simple agents that do not posses any special intelligence. These systems are particularly suitable for tackling the distributed task allocation problem since they have some interesting properties such as: they are inherently distributed, self-organizing, robust to failures, and very adaptive to environmental changes. The basic idea of the proposed approach is the following. So called ant agents are periodically generated at each node and wander through the network for finding suitable nodes to allocate tasks to. They mark the path to suitable nodes with pheromone values proportional to the goodness of nodes measured by a reward value. Other ants follow probabilistically the paths with greater amount of pheromone, reinforcing paths toward good nodes. This reinforcement process is known as positive feedback mechanism. An interesting feature of this process is that ants cooperate with each other for finding good paths, by an indirect communication mechanism, known as stigmergy. The important result of this process is that it makes it easier for arriving tasks to find suitable nodes, even if the these nodes are far away from the task arriving node.

The main contribution of this paper is to propose a new approach of application of ant colony foraging behavior in distributed task allocation problems and show its validity and effectiveness. The simulation results for different task lengths, network graphs, load levels, number and type of nodes show improved performance of ant colony algorithm compared to other heuristic based algorithms. Especially, combining the ant colony based algorithm with an heuristic approach gives the best performance.

The rest of the paper is organized as follows. Related work is presented in section 2. In section 3, the distributed task allocation model used in our study is presented. Our main approach based on ant colony foraging behavior is presented in section 4, while some extensions of it for improved performance are presented in section 5. Section 6 presents experimental evaluation. Finally, section 7 concludes our paper.

2 RELATED WORK

There are several works that deal with task allocation in distributed systems such as grid and cloud computing. For example Dam *et al.* [5] propose an ant colony load balancing strategy in cloud computing. They focus on task allocation by lunching ant agents that search for underloaded VMs whenever there are no free VMs available. However they focus more on distance of the optimal path and its not clear how they take into account the quality of VMs, while we focus on quality of the nodes to allocate tasks by using a reward value. They also tackle a centralized task allocation problem where tasks arrive at a scheduler which makes all allocation decisions. Instead we tackle a completely distributed task allocation problem where tasks can arrive at any node and agents make allocation decisions independently. Ludwig and Moallem [11] propose a distributed approach based on ant colony for job load balancing in grid computing. When a job is submitted, the system generates an ant that searches through the network for light loaded nodes to allocate the job. In this process it stores pheromone information on each node to guide other ants for finding easily suitable nodes while it is guided in its search by pheromones left by other ants. Their work is distinct to ours in two respects. First, they take a reactive approach of lunching ants at the moment of job submission which is similar to Simple_Ant_Heuristic algorithm used in our experiments. We take a proactive approach by lunching ants periodically which adapts more quickly to load changes. Second, we tackle a slightly different problem domain which is that of distributed task allocation in network of nodes with fixed neighbor structure while their system can chose any neighbor node randomly. Another approach of using ant colony optimization for load balancing in cloud computing is proposed by Nishant *et al.*[12]. In their solution ants are generated from a single head node and traverse the network for finding underloaded and overloaded nodes while updating pheromone tables. This is different from our approach where ants are generated from every node in the network. They group the nodes in three classes low, medium and high load while we use a more fine grain quality assessment through a reward value.

There is another group of works [2, 15] that similar to our work, approach the distributed task allocation problem in a network of nodes from a multi-agent perspective. For instance, in [2], authors propose a multi-agent reinforcement learning approach for distributed task allocation problem. More specifically agents apply Weighted Policy Learner (WPL) algorithm [1] for learning stochastic decision making policies. However they focus more specifically on a self-organizing mechanism for adapting the network structure through the help of reinforcement learning process. While we assume a fixed underlying network structure and focus on learning task allocation decision policies through the use of ant colony foraging behavior. Zhang *et al.* [15] propose a multi agent reinforcement learning approach for online distributed resource allocation in a network of clusters. They decompose agents decisions into local allocation problem and task routing problem and apply what they call Fair Action Learning (FAL) algorithm. The difference with our work is as follows. First, they use an utility rate as optimization metric while we optimize node queue lengths. Second and most importantly they use reinforcement learning updating rule where Q-values depends on reward and maximum next state Q-value while we use an ant colony based pheromone updating rule for policy learning.

There is a group of works [3, 7, 9, 14] that apply multi-agent learning or ant colony optimization for routing in communication networks. Our approach is similar since we learn decisions for routing tasks through a network of agents. However we tackle a different problem where the final destination node is not known in advance like it is in the data routing problem and we think this makes the problem more challenging.

3 DISTRIBUTED TASK ALLOCATION MODEL

We use a simple model of a network of nodes where each node is connected to a limited number of neighbor nodes forming an undirected graph as shown in Fig 1. Tasks can arrive at any node and we formulate the task allocation problem as deciding on which node to allocate each arriving task in order to optimize some performance metric. Each node can execute one task at a time and has a local queue where arriving tasks wait in a first-in-first-out order before being executed.

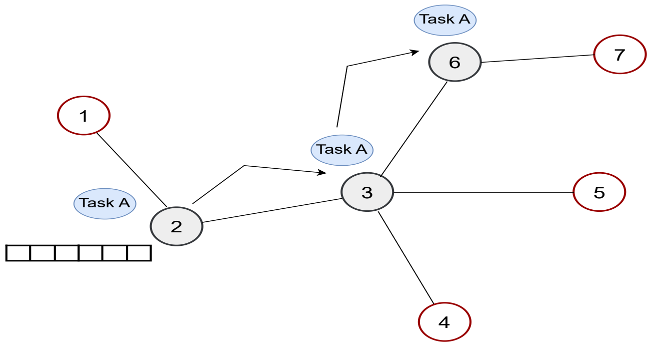

Figure 1: Distributed task allocation model

This a distributed task allocation problem since there is no central node that can make allocation decision for the whole system. A task allocation model based on a central scheduler would have a single point of failure and scalability problems in a large scale distributed system. For this reason we apply a multi-agent learning approach for the distributed task allocation problem. In this approach each agent situated in each node makes autonomous task allocation decisions based only on local view of the network which is its state and that of its neighbors. The decision made by each agent for each arriving task is whether to execute the task on local node or transfer it to one of the neighbor nodes in order to optimize a performance metric. For example in Fig 1, agent 2 has to make a decision for Task A whether to put on local queue or transfer to agent 1 or agent 3. If it is transfered to agent 3, this agent also will have to make a decision whether to put the task on the local queue or transfer to agents 4, 5, or 6. This process continues until the task is allocated to a suitable final node. By suitable we mean a node that has a small queue length as measured by the reward value. In the next section we show how node agents learn task allocation decision policies based on ant colony foraging behavior.

4 ANT COLONY FORAGING BEHAVIOR FOR TASK ALLOCATION DECISION LEARNING

In our system, each node agent makes task allocation decisions based on what we call a task routing table. Each entry of this table, corresponding to one neighbor node, stores a numerical value representing the goodness of transferring the task to that neighbor. One entry in this table corresponds to the local node. In multi-agent terms a decision action corresponds to choosing the next neighbor node to transfer a task. The goal for our multi agent system is to learn the task routing table values for an optimal decision policy. The system learns a stochastic decision policy, meaning that neighbor node corresponding to the entry with greater value will be chosen with greater probability.

For learning task routing table values we apply principles of ant colony foraging behavior of finding the shortest path to food source. When ants start foraging for food they wander randomly through the environment and lay down a substance called pheromone. After they found the food they turn back to the nest by following the previous pheromone path, again laying down an additional amount of pheromone. Other ants searching for food follow probabilistically the path with greater pheromone amount. Since more ants will pass more frequently along the shortest paths the amount of accumulated pheromone will be grater along these paths. This will attract more ants to the same paths so shorter paths will be reinforced more and after some time most of the ants would be following the shortest path to the food source.

We apply approximately the same principle as explained in the following. The task routing table entries represent pheromone values. These values are learned by sending through the network what we call ant agents. These ant agents sample and collect the network state for finding regions of network with optimal nodes to transfer tasks to. Each node periodically generates what we call a forward ant, which is sent to one of the neighbor nodes chosen probabilistically according to the task routing table. In the beginning all task routing table entries are initialize to a small value. The rule of choosing the next neighbor node is given as follows [6]. First we normalize each table entry $v[i]$ by converting them to $a[i]$ values according to the formula:

$$a[i] = \frac{v[i] * h[i]}{\sum_{i=0}^{number\_of\_neighbors} v[i] * h[i]} \tag{1}$$

where $h[i]$ is a heuristic value used in an extended version of the algorithm explained in the next section, but in the basic algorithm it is set to 1. The probability of choosing entry i is given as:

$$p[i] = \frac{a[i]}{\sum_{i=0}^{number\_of\_neighbors} a[i]} \tag{2}$$

Finally, the next neighbor node i to transfer the forward ant is chosen according to the formula:

$$i = \begin{cases} according\ \ to\ \ probability\ p[i] & \text{if } p < p_0 \\ \underset{i}{\operatorname{argmax}}(a[i]) & otherwise \end{cases} \tag{3}$$

where p is a uniformly distributed random variable and p_0 is the exploration probability. For ants we chose $p_0 = 0.9$ in order to give more weight to exploration of other entries than to exploitation of the maximum value entry. The next neighbor agent after receiving the forward ant decides according to its routing table where to transfer it. This process continues until the forward ant is transfered to a local node or some maximum number of hops is reached (which is set to 4 in our case). This process is shown in Fig 2. After reaching the final node the forward ant estimates the goodness of this node as measured by a reward value that is inversely proportional to the node queue length. The queue length is the sum of lengths of all task waiting in the local queue. Task length represents the running time of the task on a reference machine. If the queue is empty the reward takes a maximum value of 2. More specifically the reward is given below:

$$reward = \frac{c}{queue\_length}, \quad with\ c = 2 \tag{4}$$

After reward estimation the final node sends back to the source node what we call a backward ant that follows the same path of the forward ant but in backward direction. On each node of this path, the backward ant updates, using the reward value of the final node, the task routing table entry corresponding to neighbor node from which it came from as shown in Fig 2. The updating formula for entry i is given below [6]:

$$v[i] = (1 - \alpha) * v[i] + \alpha * reward \quad , \quad 0 < \alpha < 1 \tag{5}$$

This updating rule is analogues to the process of laying down of pheromones by real ants along the backward path to the nest. By reducing the entry value $v[i]$ by a factor of $(1 - \alpha)$, implicitly it is emulated the evaporation process that takes place on real pheromones. The pheromone value $v[i]$ is updated proportional to the reward value. This means that, greater the reward value,

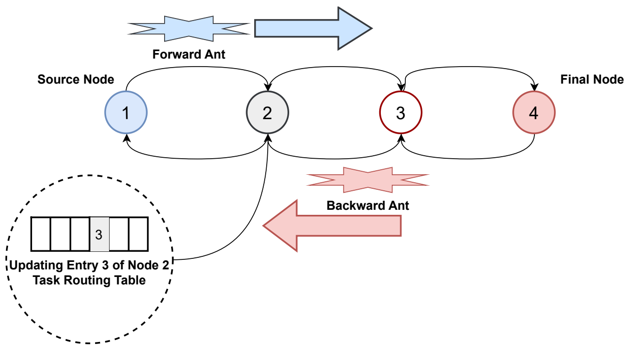

Figure 2: Ant foraging from node to node

greater the probability that the neighbor node corresponding to that entry is followed by other forward ants or tasks. In this way paths to regions of nodes with greater rewards will be reinforced more frequently analogues to the way real ants based on pheromone values reinforce the shortest path to food source. This cooperation principle that ants build their paths based on other ant paths has an interesting property for our task allocation problem. The system can find final optimal nodes for allocation of tasks that are many hops away from source nodes. We think this is the main reason why ant colony based algorithm achieves better results than other algorithms.

Whenever a task arrives at a node, whether form a neighbor node or submitted locally, the node agent makes a decision where to forward it using the same probability rule (3) as for routing of ants but with lower exploration probability $p_0 = 0.1$. This low exploration probability is chosen in order for tasks to exploit the learned pheromone values and chose most of the time the maximum value entry. If the local node is chosen then the task is put in local queue for execution, else the task is transfered to the chosen neighbor node. This process of forwarding the task from node to node continues until a local node is chosen or a maximum number of hops is reached, in which cases the task is put on local queue of the final node. In next section we present some extensions to the basic algorithm that are shown to improve the performance.

5 EXTENSIONS TO THE BASIC ANT COLONY FORAGING BEHAVIOR ALGORITHM

One extension we made is that every time the backward ant arrives at a node, after updating the task routing table, it compares the reward value that it carries, with the reward of the local node. If the reward of local node is greater, then it carries the reward of local node with it for updating the routing tables of other node on the backward path from this point onward. This means that the maximum reward along the backward path is used for task routing table updating of all nodes form the point of maximum reward node to the source node.

Another extension we made is that forward ants also are used to update, with the reward of the source node, the task routing tables of nodes along the path towards the final node. This process speeds up learning and improves the performance. Also in this case we apply the principle that the maximum reward is used to update task routing tables of all nodes along the forward path from the point of maximum reward node to the final node.

Another extension we made is that if a local node is chosen for a forward ant without reaching the maximum number of hops the ant is forwarded to next neighbor node with some survival probability that we set to 0.5. This enables the ant to explore a little further and it has shown to improve the performance compared to the case with survival probability of zero.

Last extension is the combination of the basic ant colony algorithm with an heuristic approach. This is done by combining the pheromone value $v[i]$ with a heuristic value $h[i]$ as shown in equation (1). The heuristic $h[i]$ is the reward value of the neighbor node corresponding to entry i. The heuristic values are measured for all neighbor nodes at the moment the node agent makes a routing decision. In the combined approach the heuristic value is taken into account for both ant and task forwarding decisions. With this combination the best performance is achieved compared to other approaches as shown by experimental results.

6 EXPERIMENTS

6.1 Simulation Setup

For evaluation we simulated a network of nodes using a discrete-event simulator build in python programming language. We tested for different network graphs randomly generated using Erdos Renyi model [8]. We evaluated the case where tasks are generated in only 30% of the nodes with task arrival times according to Poisson distribution with different mean inter-arrival times. The time values in the following represent simulation time units. Tasks lengths were taken from two different uniformly distributed intervals: low variability interval [60:70] and higher variability interval [50:150]. We simulated two network infrastructures: one with homogeneous nodes having the same computing speed and the other with heterogeneous nodes having different computing speeds. After some experimentation we found that setting the parameter $\alpha = 0.2$ in equation (5) gives the best results. Another parameter is the agent generation interval which we set to 10 time units as it is shown to give good results. We use the average of task waiting time in the queues of the system as performance metric. This performance metric is a fair metric for evaluation since our algorithms try to minimize directly the queue length through the reward value.

We compared 5 different algorithms summarized as follows. **1) Ant**: is the ant colony based algorithm with the extensions explained in section 4. **2) Ant_Heuristic**: is the combined algorithm of pheromone values with heuristics as explained in section 4. **3) Heuristic**: is the algorithm that uses only heuristic values in making decisions, where we set the value $v[i] = 1$, in equation (1). **4) Simple_Heuristic**: this algorithm works as follows. When a task arrives, the node generates an agent that follows a path in the network based on the **Heuristic** algorithm until a maximum number of hops is reached. The agent records the node with maximum reward that it encounters during path traversal and turns back this information to the source node. The task is then forward to this maximum reward node for allocation. **5) Simple_Ant_Heuristic**: is the same as **Simple_Heuristic** but the agent follows a path based on the decisions of the **Ant_Heuristic** combined algorithm. In all algorithms based only on heuristics, there are no ant agents generated since there is no need to learn pheromone values.

6.2 Simulation Results

We evaluated for different scenarios of network graphs, task length variability, task inter-arrival times, node speeds and node numbers given in shorthand form as Si=(Hom/Het,Gi,[min:max],t,n) for scenario i with Homogeneous/Heterogeneous nodes, graph Gi, task length in the interval [min:max], task average inter-arrival time t and number of nodes n. Here we summarize the different scenarios: **S1**=(Hom,G1,[60:70],60,50), **S2**=(Hom,G1,[50:150],60,100), **S3**=(Het,G1,[50:150],60,100), **S4**=(Hom,G1,[50:150],100,100).

For each scenario we executed 10 simulation runs and measured the average performance. Each simulation run is fed with a different random seed in order to take into account the randomness in the execution. Also we evaluated a modified version of scenario **S2**,

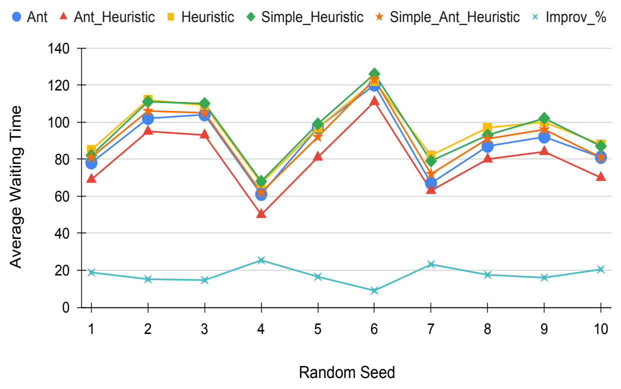

Figure 3: Performance for scenario S2

where we kept the random seed constant but executed 10 simulation runs with different network graph each run.

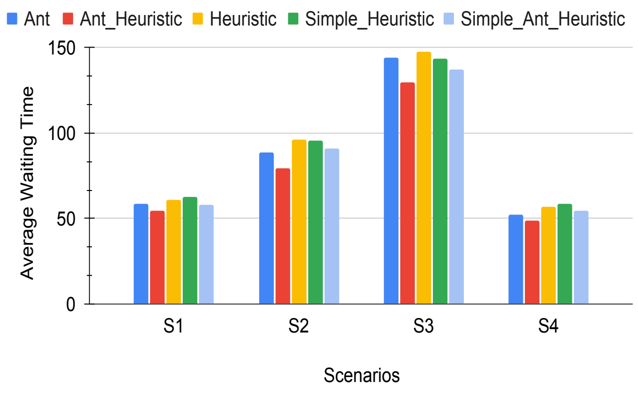

Figure 4: Average performance for different scenarios

In Fig. 3 is shown the average waiting time for each random seed for scenario *S*2. As we can see for each seed, the performance of **Ant_Heuristic** algorithm is better compared to other algorithms. Also we can see that **Ant** algorithm although is near in performance to **Heuristic** and **Simple_Heuristic** algorithms there are certain seeds such as seed 2, 7 and 8 where it is much better. Overall we can observe that all ant colony based algorithms show improved performance compared to all algorithms that are not ant based. This observation leads to the conclusion that the performance of the systems is improved if pheromone values learned pro-actively through ant agents are taken into account in decision making. In Fig 3 is shown also the percentage of performance improvement (Improv_%), given as the percentage of reduction in waiting time achieved with **Ant_Heuristic** algorithm compared to **Heuristic** algorithm. We see an improvement of near 20% and in certain seeds (seed 4 and 7) it is over 20%, showing clearly the advantage of the ant based algorithm.

To get an overall view of the behavior of algorithms, we show in Fig. 4 the average performance over 10 simulation runs of all algorithms for every scenario. We observe that for each scenario, **Ant_Heuristic** algorithm achieves better performance than all the others. Also here we can observer what we mentioned earlier that all ant colony based algorithms have better performance than non ant based algorithms, although with different degrees of improvements depending on the scenario. For instance, greater improvements of ant colony based algorithms are achieved in scenarios with higher task length variability and load levels such as in *S*2 and *S*3.

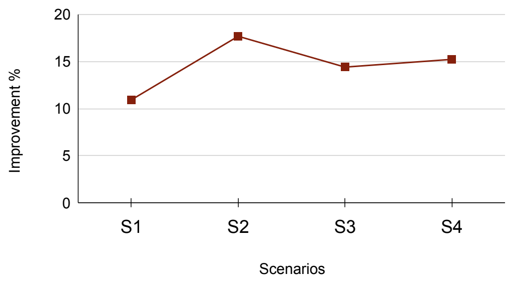

Figure 5: Improvement % of Ant_Heuristic over Heuristic

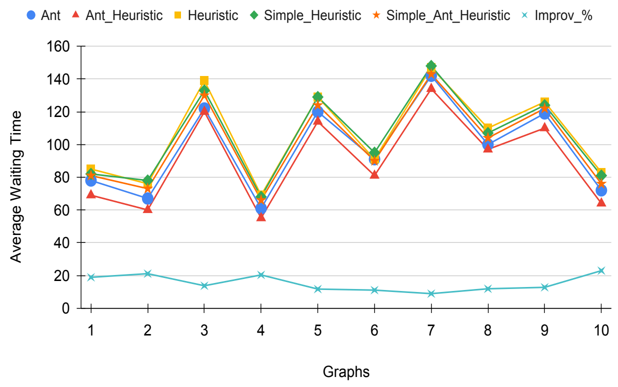

Figure 6: Performance for different network graphs

To get an overall view of performance improvement percentage, in Fig. 5 is shown the average percentage of reduction in waiting time achieved with **Ant_Heuristic** algorithm compared to **Heuristic** algorithm for every scenario. In general the average performance improvement is around 15%, with best results for *S*2 scenario with 17%. Here again we can observe that better results are achieved in high task length variability scenarios.

In Fig. 6 is shown the performance of all algorithms for 10 different network graphs generated randomly for the *S*2 scenario. We can observer here that for each network graph, **Ant_Heuristic** algorithm achieves better performance then other algorithms. In general all ant colony based algorithms are better than non ant based ones although with less noticeable difference then that achieved by **Ant_Heuristic**. To understand better the improvement we can see the percentage of performance improvement of **Ant_Heuristic** compared to **Heuristic** in the bottom the line of the graph. In general the improvement is about 15% but for certain network graphs

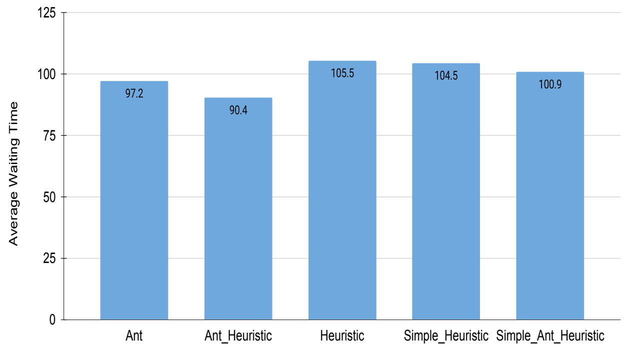

Figure 7: Average performance for different network graphs

it is even 20%. These results show the consistency of ant colony based algorithm in achieving good performance even for different network graphs. Fig. 7 shows the average performance over all network graphs for all algorithms. Here we can seen in a concise manner the superiority of ant base algorithms compared to other algorithms.

7 CONCLUSIONS

In this paper we tackle the problem of distributed task allocation in a network of nodes that arise in many real world scenarios such as grid and cloud computing systems. We propose a multi-agent learning approach based ant colony foraging behavior for optimizing task allocation decisions. Ant agents spread through the network build paths of pheromones toward suitable nodes to allocate task to, as measured by a reward value. Node agents independently of each other make task forwarding decisions by applying stochastic policies based on learned pheromone tables. Simulation results show that making task allocation decisions based on pheromone paths learned by ant agents lead to improved performance compared to other heuristic algorithms. We think that the main reason, why ant colony based algorithm achieve better results, is the ability of node agents for forwarding tasks to good nodes that are far away form the local node. Based on simulation results we can conclude that the proposed ant colony based approach is a viable and effective solution to the distributed task allocation problem.

One possible extension for future work is to take into account task network transfer time when learning paths toward suitable nodes, which in this paper it is assumed to be negligible. One straightforward way to do this is to include in the reward value not only the final node queue length but also the task data size and number of hops to the final node.

ACKNOWLEDGMENTS

The work presented in this paper has been supported by Faculty of Information Technology, Polytechnic University of Tirana and the Erasmus Mundus Programme.

REFERENCES

[1] Sherief Abdallah and Victor Lesser. 2006. Learning the Task Allocation Game. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06)*. Association for Computing Machinery, 850–857.

[2] Sherief Abdallah and Victor Lesser. 2007. Multiagent Reinforcement Learning and Self-Organization in a Network of Agents. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '07)*. ACM.

[3] Justin Boyan and Michael Littman. 1993. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In *Advances in Neural Information Processing Systems*, Vol. 6. Morgan-Kaufmann.

[4] David W. Corne, Alan Reynolds, and Eric Bonabeau. 2012. *Swarm Intelligence.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1599–1622.

[5] Santanu Dam, Gopa Mandal, Kousik Dasgupta, and Paramartha Dutta. 2014. An Ant Colony Based Load Balancing Strategy in Cloud Computing. In *Advanced Computing, Networking and Informatics- Volume 2*. Springer International Publishing, 403–413.

[6] Marco Dorigo, Gianni Di Caro, and Luca M. Gambardella. 1999. Ant Algorithms for Discrete Optimization. *Artificial Life* 5, 2 (1999), 137–172.

[7] Frederick Ducatelle, Gianni A. Di Caro, and Luca Maria Gambardella. 2010. Principles and applications of swarm intelligence for adaptive routing in telecommunications networks. *Swarm Intelligence* 4 (2010), 173–198.

[8] P. Erdos and A. Renyi. 1960. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.* 7 (1960), 17.

[9] Di Caro G. 1998. AntNet : Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research* 9 (1998), 317–365.

[10] Raquel V. Lopes and Daniel Menasce. 2016. A Taxonomy of Job Scheduling on Distributed Computing Systems. *IEEE Transactions on Parallel and Distributed Systems* 27, 12 (2016), 3412–3428.

[11] Simone A. Ludwig and Azin Moallem. 2011. Swarm Intelligence Approaches for Grid Load Balancing. *Journal of Grid Computing* 9, 3 (2011), 279–301.

[12] Kumar Nishant, Pratik Sharma, Vishal Krishna, Chhavi Gupta, Kuwar Pratap Singh, Nitin, and Ravi Rastogi. 2012. Load Balancing of Nodes in Cloud Using Ant Colony Optimization. In *2012 UKSim 14th International Conference on Computer Modelling and Simulation*. IEEE, 3–8.

[13] Shubham Suresh Pol and Avtar Singh. 2021. Task Scheduling Algorithms in Cloud Computing: A Survey. In *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*. IEEE, 244–249.

[14] Ruud Schoonderwoerd, Owen E. Holland, Janet L. Bruten, and Leon J. M. Rothkrantz. 1997. Ant-Based Load Balancing in Telecommunications Networks. *Adaptive Behavior* 5, 2 (1997), 169–207.

[15] Chongjie Zhang, Victor Lesser, and Prashant Shenoy. 2009. A Multi-Agent Learning Approach to Online Distributed Resource Allocation. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI '09)*. Morgan Kaufmann, 361–366.