Comparison of Market-based and DQN methods for Multi-Robot processing Task Allocation (MRpTA)

Paul Gautier, Laurent D. Johann, Jean-Philippe Diguet

Comparison of Market-based and DQN methods for Multi-Robot processing Task Allocation (MRpTA)

Paul Gautier, Johann Laurent, Jean-Philippe Diguet
Lab-STICC UMR6285, CNRS, Université de Bretagne Sud, Lorient, France
Preprint - Paper Accepted for presentation at IEEE International Conference on Robotic Computing (IRC) 2020
Contact author: paul.gautier@univ-ubs.fr

***Abstract*—Multi-robot task allocation (MRTA) problems require that robots take complex choices based on their understanding of a dynamic and uncertain environment. As a distributed computing system, the Multi-Robot System (MRS) must handle and distribute processing tasks (MRpTA). Each robot must contribute to the overall efficiency of the system based solely on a limited knowledge of its environment. Market-based methods are a natural candidate to deal processing tasks over a MRS but recent and numerous developments in reinforcement learning and especially Deep Q-Networks (DQN) provide new opportunities to solve the problem. In this paper we propose a new DQN-based method so that robots can learn directly from experience, and compare it with Market-based approaches as well with centralized and purely local solutions. Our study shows the relevancy of learning-based methods and also highlight research challenges to solve the processing load-balancing problem in MRS.**

***Index Terms*—Multi-robot tasks allocation, distributed system, reinforcement learning, deep Q-learning**

I. Introduction

The robust and flexible nature of multi-robot systems (MRS) makes them particularly suitable for critical tasks such as security or rescue missions. However, effectively coordinating the robots requires to accurately distribute the work within the system which leads to the multi-robot task allocation (MRTA) problem. This problem has been extensively studied and many approaches have been proposed to solve it [1].

But autonomy, safety and accuracy requirements come with a high computing cost due to a significant increase of sensor infrastructure and of algorithm complexity developed to sense, analyse and decide. A direct consequence is that processing tasks, within the MRS, become a major impediment to their development. In fact, despite the improvement of embedded-systems processing and storage capacities, computing resources are intrinsically limited. Cloud-robotics [2] have been introduced as an alternative solution, but it is applicable only when reliable and high bandwidth connections are available and when the processing latency, namely the response time, is not critical.

The concept of Robotic Cluster has also been introduced to speed up computationally hard tasks such as SLAM, which is detailed in [3]. This work demonstrates the possibility to get benefit from multiple processing resources distributed over a cluster of interconnected robots to execute a parallel version (e.g. multithreaded) of a complex application task. If we extend this work to multiple independent tasks, then sharing resources allows to improve the processing capacity of each single robot with multi-tasking capabilities. Considering a set of processing tasks to be executed by the MRS, the new question to solve becomes a task allocation problem with constraints such as execution time and priorities. This question is a variant of the MRTA problem focused on processing tasks (MRpTA).

Considering the MRS context, different options are possible. Firstly, the method can be centralized or decentralized. On the one hand, a centralized approach solves most of the difficulties by allowing a decision making based on a perfect knowledge of the state of each processing resource. But on the other hand, it brings two major drawbacks. The first one is the unreliability since the central agent introduces a single point of failure. The second one is the inherent communication overhead due to the aggregation of the all system knowledge in a single robot. This communication cost strongly limits the system scalability and makes it unpractical for MRS. Therefore, we only focus on distributed systems. Regardless the architecture the system uses, the problem may take many forms from travelling salesman problem (TSP) to job scheduling. The latter is more frequently studied in the field of multiprocessors or computer clusters than in robotics. It also raises new challenges related to the rapid development of autonomous systems. Indeed, the last few years have seen the emergence of new heavy processing methods in several areas such as computer vision, sensor fusion and especially machine (deep) learning.

In this paper, we investigate the task allocation problem within a MRS according to their computational and memory costs, with the joint objectives of task completion and fair load balancing.

A market-based approach is a quite straightforward possible candidate, which has been used in conventional MRTA problem in robotics and that can be adapted to the specificity of processing tasks as detailed in Sec. II. But a MRS is also composed of increasingly complex embedded systems, including multiple sensors, multi-core architecture with GPU, which are almost impossible to accurately model. Moreover the robots evolve within an uncertain environment and execute applications accordingly. The dynamic nature of the whole system requires a high degree of adaptability to cope with the lack of information (data availability, communication errors) and rapid changes (failure, object detection).

Based on these observations we consider the opportunity to use machine learning approaches and more specifically

Reinforcement Learning (RL), which has been successfully applied in many fields, such as energy management, communication optimisation, job scheduling, etc. In RL, the agent progressively learns to improve the quality of its decisions according to the experience it acquires by means of a reward that reflects the efficiency of the chosen actions. Recently, the combination of deep neural networks and RL has been introduced (Deep Q-Learning) to deal with the scalability issue of RL. Thus, instead of storing values in tables that grow with the environment dimensions, a neural network is used to approximate the policy function and find the best action according to the environment. This method has become very popular with the success of the Deep Q-Learning on Atari games [4].

In this article, we explore the viability of Deep Q-Learning to solve the MRpTA problem and compare it with a multi-robot task allocation problem with the following questions:

- Can robots of a fully decentralized system learn to efficiently manage task allocation on their own?
- What is the performance of a method based on Deep Q-Learning compared to a more traditional approach like the market-based one?
- What does the use of learning imply for real-life applications?

The rest of this paper is organized as follows: Sec. II discusses relevant works on the MRTA problem and RL. Sec. III explains the modelling of our problem and the approach used to solve it. Sec. IV describes our experimentation set-up. Experimental results are discussed in Sec. V. Finally, we conclude and introduce future work based on this study.

II. Related work

A. Multi-robot tasks allocation

We briefly introduce the key points of our MRTA problem, the reader can refer to cited references for detailed surveys and complete formulations.

a) MRTA architecture: The literature provides various instances of the MRTA problem, in order to offer a broader and more theoretical view, Gerkey and Mataric [1] have proposed a taxonomy for those problems. Based on their definition, our architecture is a multi-task type, single robot and instantaneous assignment (MT-SR-IA). In this decentralized context, we compare the efficiency of two methods: a market-based approach and an approach using reinforcement learning.

b) Market-based systems: A market-based approach enhances the efficiency of the overall system by maximizing individual profits. Several surveys address this subject [1], [5]. In our comparison cases, overloaded robots sell tasks to others in order to maximize the system's task completion rate. The sale is made by auction where an auctioneer (the seller) offers a task for sale. Each participating robot submits an auction whose valuation depends on its ability to perform the task. The best bidder wins the task and runs it. The model used for auctions is sequential single item mechanism, which is both efficient and inexpensive [6]. Further details on the auctions can be found in [7], [8].

Although it is inexpensive in communication or processing, this process raises the question of bid estimation. It is indeed difficult to correctly estimate a bid when several parameters are involved and especially, when their value depends on the environment's state. To overcome this problem, we propose another approach based on reinforcement learning.

B. Reinforcement learning

One of the main goals of our RL approach is to remove the auction system. Agents must then be able to correctly estimate the relevance of a transfer without exchanging information about their status.

1) Principles: In RL, the agent observes at each time step, the environment's state s_t and chooses an action a_t. This action modifies the environment, which then proceeds to the next state s_{t+1}. Then, the agent receives a reward r_t according to the quality of its choice. The learning aim of the agent is to maximize the cumulative value of future rewards. To operate, this method requires that the state transitions are stochastic and have the properties of a Markov Decision Process (MDP). It means that the rewards r_t and states' transitions s_{t+1} must depend only on the environment s_t and the action a_t. Fig 1 illustrates the principle of reinforcement learning

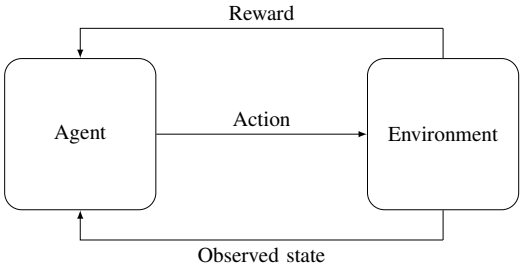

Fig. 1. Reinforcement Learning Diagram

One of the most popular reinforcement learning methods is Q-Learning, which chooses its actions based on Q-values. Q-Learning uses a table to store all Q-values of all possible {state, action} pairs. This Q-table is updated using the Bellman equation (eq. 1). The action selection is usually done with an ϵ-greedy policy. The Q value can be calculated using the following formula and definitions:

$$Q_\pi(s_t, a_t) = r(s_t, a_t) + \gamma max Q_\pi(s_{t+1}, a_{t+1}) \qquad (1)$$

- $r(s_t, a_t)$ returns the reward of action a_t in the state s_t
- γ in [0,1] is the discount factor witch control the value of future rewards
- $Q_\pi(s_{t+1}, a_{t+1})$ returns the optimal possible Q-value of the next state

Although effective, the Q-Learning method has some limitations. Indeed, it does not scale with the number of states since the number of pairs {state,action} increases exponentially leading to a very large Q-table and so requiring a large amount of memory. To overcome this problem, the Deep Q Network (DQN) method has been introduced, it combines principles of Q-Learning and a deep neural network (DNN) [4].

a) Deep Q Network: The limitation can be solved using a DNN like a function approximator. An approximation is possible since an agent must take similar actions for 'close-by' states. Fig.2 shows a Deep Q Network who uses parameter fitting to construct a function able to predict Q-values.

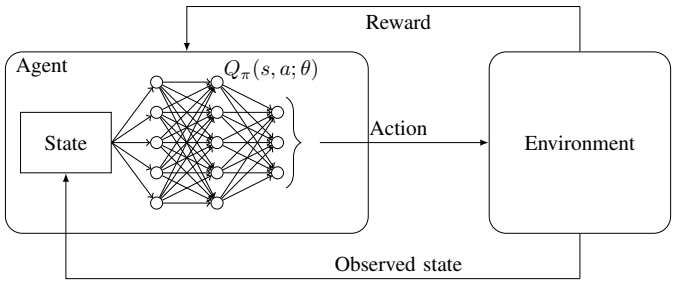

Fig. 2. DQN diagram

First, we construct a loss function using the mean square deviation to define the target function (eq. 2). Then we update weights using the Adam optimiser [9], which is computationally efficient.

$$L_i(\theta_i) = E[(y_i - Q_\pi(s_t, a_t, ; \theta_i))^2] \quad (2)$$

with: $y_i = r(s_t, a_t) + \gamma max Q_\pi(s_{t+1}, a_{t+1}; \theta_{i-1})$
However, by replacing the estimating function Q-values with a DNN the algorithm becomes unstable. The main reason is that there is a strong correlation between continuous states and action inputs. Each small update of the Q value of an action causes the modification of the set of network weights, which affects the Q value of each action in the other states. This strongly impacts the distribution of sampling data. To avoid instability we use *experience replay* developed by [10]. With this mechanism, each experiment {state, action, reward} is stored in a database. The agent learns from samples selected randomly and consistently in its database. Thus, all correlations are broken, and learning is accelerated.

b) Distributed reinforcement learning: Our approach being decentralized, each robot has its own learning system. Many distributed approaches take advantage of the multiplicity of local learning to accelerate the learning of the system [11]. Indeed, it is possible to exchange experiences between agents. Thus, each agent learns from others which allows it to converge more quickly towards an optimal solution as shown by [12]. We do not consider this mechanism that requires a large data exchange which is incompatible with a real deployment of mobile robots. Hereafter we describe the method used to compare these two approaches.

III. Method Definition

The two compared approaches share the same definition of the environment. They differ by their decision making process, in particular for the load balancing mechanism.

A. Definitions shared by both approaches

1) Context and objectives:

a) MRS objectives: the system pursues two objectives. The first one is the completion of a maximum of tasks assigned to it. It means that it must distribute the load over the entire MRS. The second objective is to favour high priority tasks. But the environment constraint and evolution make impossible to fully respect both objectives. The system must make choices.

For each task assigned to it, the agent must choose from one of three possible actions: run, postpone and transfer.

1) *run* : if enough resources are available, it executes the task until completion.
2) *postpone* : postpones the task until the next allocation cycle or fails the task if its deadline is reached.
3) *transfer* : transfers the task to another agent (see III.B and III.C for details). This transfer involves a penalty in order to model the overhead generated by the communications. The CPU and memory resources equivalent to the consumption of the transferred task during a cycle are blocked on the receiving agent while the transfer occurs. The task runs only in the next cycle.

b) MRS communications: the system being totally decentralized, robots communicate only during a transfer. There is no other information's exchange. Each robot relies solely on its local information to schedule its tasks. This low communication's level ensures a high-level of scalability. To get closer to a real deployment, we consider an environment composed of multiple areas where communications can only be established between robots of a same area.

c) Further information:

- The presence of a robot in a given area is not known by other robots.
- Transfer occurs via broadcast.
- Communications take place smoothly.
- Time is modelled as discrete timesteps.
- Robots are homogeneous and evolve in parallel
- Tasks are assigned to robots in a sporadic and uniformly random manner.

2) Task definition: In our model tasks are independent and have a fixed priority (but no task is imperative). Furthermore, there is no task preemption except case given in III.B. Similarly to prior work [3], we assume that task characteristics is known upon its arrival in the robot queue. The main task features are the following:

- CPU usage: processing resource needed to run the task.
- Memory usage: memory space needed to run the task.
- Execution time: the number of iterations required to complete the task.
- Priority: upon its creation, each task is assigned to a fixed priority level. There are three levels, high, medium, and low corresponding to values 1, 2, and 3 respectively. This value is randomly selected in a uniform manner.
- Laxity: maximum time allowed before the task starts. After this delay, the task is considered failed if not started.

Since pending tasks are stored in a queue, agents cannot choose which task to deal with. To solve this problem, the tasks in the queue are automatically sorted by priority (1 to 3) and by earliest deadline. Then the decision process depends on the approach used.

B. Market-based approaches

We consider two approaches, with or without preemption. They share the same decision making process, only the scope of the transfer mechanism differs as explained in III-B2.

1) Decision process: First, each agent tries to run local tasks as long as resources permit it. Then, it tries to transfer the non-executed tasks to other robots. And finally, it postpones the remaining tasks that have not found a buyer. A complete diagram of the decision process of the different algorithms used in this article can be found in Fig. 4. The transfer action uses an auction system to select the best receiver.

2) The auction system: When an agent wants to transfer a job, it uses the auction system mechanism. It then becomes an auctioneer and proposes selling a task to all other agents in its area. Any agent receiving the offer submits a bid whose evaluation process is described in Algo. 1. The preemptive method allows agents to stop the execution of certain tasks to accept a transfer. Preemption can only be done to the detriment of lower priority tasks. Stopped tasks can be restarted by the agent but will start from scratch (as penalty) and must respect their initial deadline. In the preemptive method the auctioneer can participate to its own auction which consequently allows local preemption. It is worth mentioning that the centralized nature of auctions does not contradict the decentralized nature of the system. Indeed, this centralization is temporary and any agent can become an auctioneer. Therefore, there is not introduction of a single point of failure (SPOF).

C. DQN approaches

Unlike market-based approaches, the system has complete freedom over the action it chooses. The only limit being the validity of the action, the system cannot perform an impossible action like allocating more resources than owning. The decision process depends on the Q-values calculated during the learning. Moreover, no preemption is allowed for this approach so the system must learn to refrain from performing low priority tasks to ensure high-priority task execution.

1) Decision process: As mentioned earlier, our RL method relies on a DNN to predict Q-values. These Q-values determine which of the three actions (run, transfer or postpone) to choose in the current state. The architecture of our network comprises an input layer of 9 neurons, two hidden layers of 32 neurons each and an output layer of 3 neurons as described by Fig. 3. The input layer brings two types of information. The first one is the state of the agent and consists of four neurons (CPU load, memory load, current area and tasks remaining in the queue). The second one, composed of five neurons, delivers information (CPU requirement, memory requirement, priority, deadline, transfer) about the task to be allocated.

In this approach, there is no explicit system for selecting the receiver robot for the transferred task. As a result, the process differs from the auction approach.

2) DQN and transfer: The transfer action causes the release of a transfer proposal to other agents in the area. Each concerned agent uses its DQN to choose whether to accept the task or not. If only one agent responds favourably, it will execute the task at the next cycle with the penalties due to the transfer. If several favourable answers exist, the agent is randomly selected regardless of the relevance of this choice. Agents must learn to properly evaluate the appropriate response to a transfer request. Finally, the task is postponed if no favourable response is sent to the auctioneer.

Another key factor of a DQN method is its reward system, which fully conditions the learning quality and its relevance.

3) The rewards' system: Our system has two objectives: the completion of a maximum of tasks and the respect of priorities. Moreover, the total freedom of this approach implies a third objective: the limitation of the use of the transfer in order to avoid over-communication between the agents. Our system

Algorithm 1: Bid valuation algorithm
(In red, steps specific to the pre-emptive approach)

Data: T, the auctioned task
A, the list of active tasks on the robot
A_i, the list of active tasks with a priority i
R_{cpu}, the amount of free CPU on the robot
R_{mem}, the amount of free memory on the robot
Function : $cpu(i)$, returns the CPU required by task i
$mem(i)$, returns the memory required by task i
Result: Bid, The bid valuation
if $R_{cpu} \geqslant cpu(T)$ *and* $R_{mem} \geqslant mem(T)$ **then**
  $Bid \leftarrow R_{cpu} + R_{mem}$;
else if $priority(T) = 1$ *and*
  $R_{cpu} + \sum_{i \in A \setminus A_1} cpu(i) \geqslant cpu(T)$ *and*
  $R_{mem} + \sum_{i \in A \setminus A_1} mem(i) \geqslant mem(T)$ **then**
  $Bid \leftarrow R_{cpu} + R_{mem} - \sum_{i \in A_1} (cpu(i) + mem(i))$
else if $priority(T) = 2$ *and*
  $R_{cpu} + \sum_{i \in A_3} cpu(i) \geqslant cpu(T)$ *and*
  $R_{mem} + \sum_{i \in A_3} mem(i) \geqslant mem(T)$ **then**
  $Bid \leftarrow R_{cpu} + R_{mem} - \sum_{i \in A \setminus A_3} (cpu(i) + mem(i))$
else
  $Bid = -2$;
end

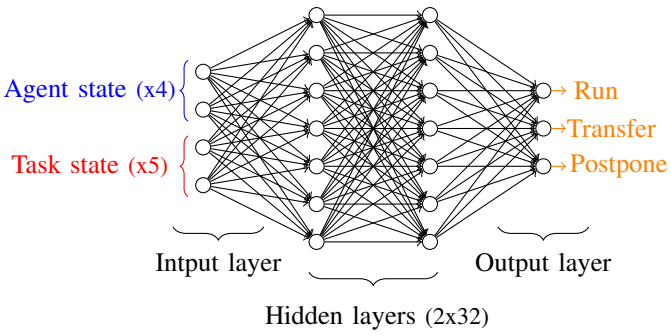

Fig. 3. Schematic view of our DQN architecture

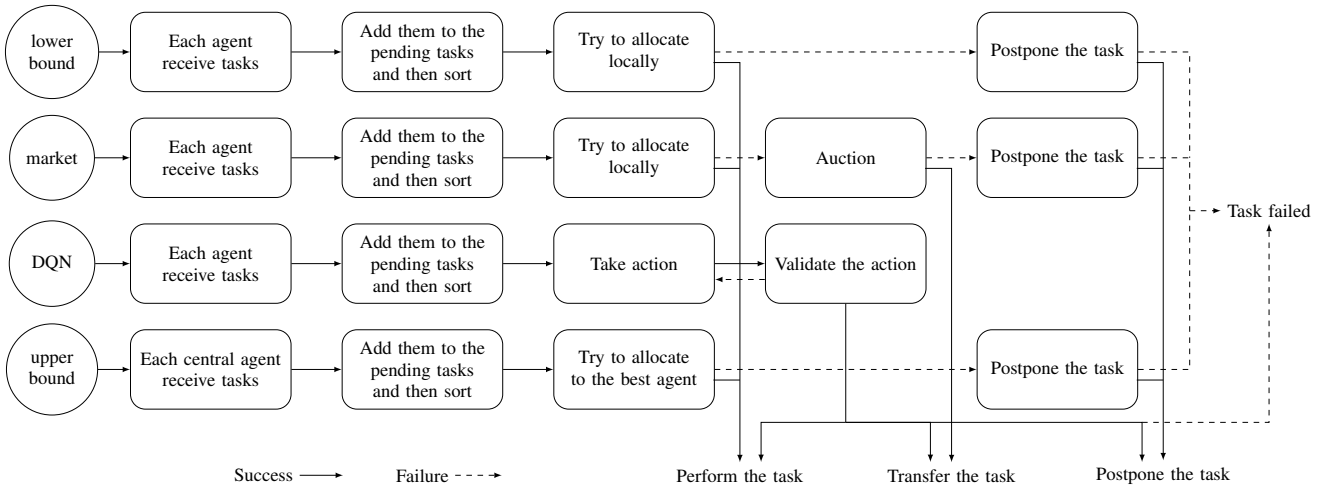

Fig. 4. Decision process diagram for the different methods

rewards differently if the action leads to performing a task (action 0 or action 1 successfully) or not.

$$r(s_t, a_t) = \begin{cases} V_i, & \text{if performed.} \\ \dfrac{j \times (3-i)}{L} + \beta\delta_{0\tau}, & \text{otherwise.} \end{cases} \quad (3)$$

with:

- i, j, L : the priority, the number of postponement and the laxity of the task, respectively. These values are obtained from s_t and s_{t+1}.
- $V = [\alpha_1, \alpha_2, \alpha_3]$: reward based on priority
- τ : takes the value 0 if the transfer is a success, 1 otherwise
- β : the transfer's penalty

The values of α_1, α_2, α_3, β have been determined by successive simulations and are $1, 0.3, 0.05, -0.2$, respectively. We will now present the other parameters of our experiments.

IV. Experimental conditions

A. Experimentation set up

Before deploying our algorithms on mobile robots, we test our comparison on an emulator to evaluate its relevance.

1) Emulator set up: Learning methods based on DNN require intensive computing. We have, therefore, tested the learning and inference times on an embedded architecture adapted to mobile robots. Our emulator is coded in python 3.7.3. The learning and deployment part of DNN rely on the widely used Tensorflow (with Keras), open source ML framework.

2) Viability for robot deployment: The Jetson TX2 is a power-efficient ($< 15W$) computing device suitable for Embedded AI. Since it is very popular on mobile robot, it will serve as a reference. As shown in Table I, results attest our reinforcement approach is fully compatible with the computing power available on a robotic architecture.

TABLE I
Comparison of Emulator and Embedded architectures

Set up	Emulator	Nvidia Jetson TX2
Components	2 Intel Xeon Silver 4114 Nvidia GTX 1080 TI	ARM Cortex-A57 NVIDIA Denver2
Hardware	20 cores at 2.20 GHz 64 GB of DDR4 3584 CUDA cores 11 GB GDDR5X	4 cores at 2 GHz + 2 cores at 2 GHz 8GB 128-bit LPDDR4 256 CUDA cores
Inference	$0.002s$	$0.0045s$
Experience replay (Batch of 24)	$0.21s$	$0.52s$

B. Reference performances

We introduce a lower and upper bounds in order to get comparison points for our three methods: DQN, market based ('Auction') and market-based with preemption ('P-Auction').

1) Lower bound: A purely decentralized and local approach acts as a lower bound ('L-Bound'). There is no communication between the agents. Each robot acts independently of the others. All other approaches must be superior because otherwise they are counter-productive.

2) Upper bound: As an upper bound ('U-Bound'), we logically chose a centralized approach with preemption. Indeed, because of its overall knowledge of the state of the system, the centralized vision offers optimal performance. All decentralized systems must aim to achieve such performances. Since our system is spread over multiple areas, our centralization is done by area. Each of them has a central agent independent of the MRS and equivalent to a local cloud. At each iteration, the new tasks to be assigned to the area agents are, instead, assigned to the central agent of the corresponding area. Then, the central agent distributes the tasks by favouring the load balancing among all available agents. At each iteration, all tasks are assigned to allow agents moving between areas ('carrying' their tasks with them). The surplus tasks are distributed uniformly between the robots and reclaim, at the beginning of each new iteration, to be redistributed.

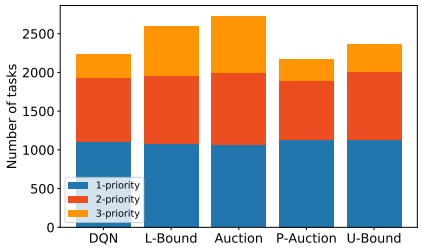

Fig. 5. Tasks completed

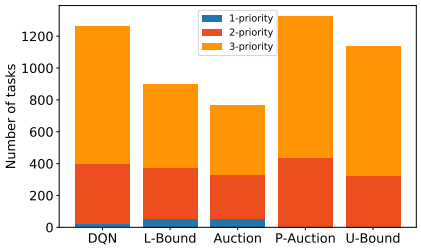

Fig. 6. Tasks failed

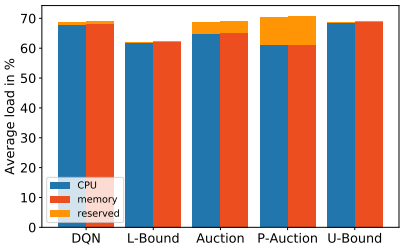

Fig. 7. Average load

C. Experimentation configurations

Because of space limitations, we restrict our study to a representative set of configurations.

Environment Settings:

- Number of robots: 7
- Number of areas: 3
- Maximum postpone value: 5
- Task execution time : 4

Learning parameters:

- $\gamma = 0.5$
- $\epsilon_{min} = 0.01$
- Epsilon decay $= 0.995$
- Learning rate $\alpha = 0.001$
- Batch size: 24

1) Tasks assignment: Tasks appear over time in an aperiodic way and are randomly assigned to an agent. To simulate activity peaks, the number of new tasks assigned to the systems is of serrated type from two to twelve (and then twelve to two) and so on. The average number of tasks assigned per iteration is therefore seven, which, according to the experiments and configurations, seems to be the sweet spots between the overload and the complete system saturation. During the experiments, the approaches evolve in parallel in a strictly identical environment (the assignment of tasks is identical).

2) Area assignment: Our experiment includes three areas ($area_1$, $area_2$ and $area_3$) and all agents start in the $area_1$. Every ten iterations, each robot is assigned to a zone that can be the same as the current one. The assignment probabilities for each area are as follows: $P(area_1) = \frac{1}{2}$, $P(area_2) = \frac{1}{3}$ and $P(area_3) = \frac{1}{6}$. The differences in probabilities introduce an imbalance to which the DQN approach must fit. As with tasks assignment, this assignment is identical for each studied approach.

We now present the results of our experiments obtained with dataset described in Table II. We have generated a balanced synthetic task set where no preference exists for CPU or memory usage. So the task set is built with the following property: if a task requesting x CPU and y memory (MEM) exists, then a mirrored task requiring y CPU and x MEM exists as well. In addition, there is also a balanced task that requires x CPU and x MEM. These results are those of an experiment taking place over 800 iterations. Since the DQN approach requires about 300 iterations to be effective, we discard the data out of the first 300 iterations to only analyse the last 500 iterations.

TABLE II
TASKS' DATASET

ID	CPU (%)	MEM (%)	ID	CPU (%)	MEM (%)	ID	CPU (%)	MEM (%)
T1	5	30	T2	5	5	T3	10	15
T4	10	40	T5	10	10	T6	15	10
T7	15	25	T8	15	15	T9	20	35
T10	20	30	T11	20	20	T12	25	15
T13	25	35	T14	25	25	T15	30	5
T16	30	20	T17	30	30	T18	35	20
T19	35	25	T20	35	35	T21	40	10
T22	40	40						

V. RESULTS

A. Overall performances

These simulations are intended to answer questions raised by the use of these approaches. The first question is *how do these strategies affect the system's capacity to complete its tasks?*

1) Completion rate: Fig. 5, 6 show the number of tasks completed and missed respectively, according to their priority for each approach. Pre-emptive approaches (P-Auction and U-Bound) and to a lesser extent the DQN approach provide a much lower completion rate. This is explained by the penalty caused by the eviction of low priority tasks which forces the execution to be repeated from the beginning. The Auction approach takes advantage of the transfer to offer the best completion rate regardless of priorities.

Approaches using pre-emption or learning have a lower completion rate. *Are they able to show better performance in respect of critical tasks?*

2) Criticality management: Pre-emptive approaches are effective in fulfilling criticality with a zero failure rates for *1-priority* tasks, as shown in Fig. 6. The DQN approach is likely to provide better results than non-pre-emptive approaches. This shows that this solution refrains from allocating as soon as possible and retains some scope for high priority tasks. However, it fails to ensure all *1-priority* tasks.

We see a need for compromise. The system cannot both complete a maximum of tasks and effectively maximize the completion of high priority tasks. The relevance of an approach therefore depends on the weighting given to the two objectives. In this situation, *how to evaluate the performance of the different solutions?*

3) Load balancing: The ability of a solution to maximize the use of available resources is a good performance indicator. Fig. 7 shows the average load distribution of an agent during

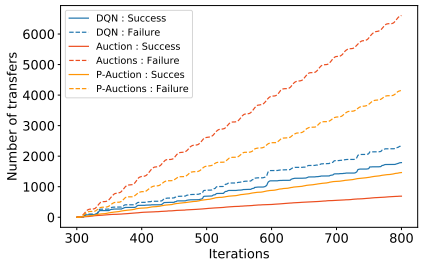

Fig. 8. Transfer quality

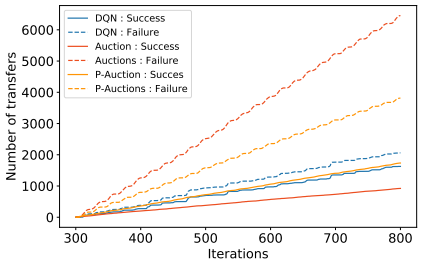

Fig. 9. Transfer quality with full range

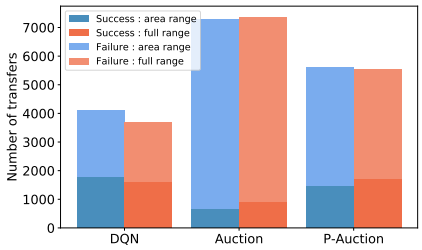

Fig. 10. Transfers' comparison

500 iterations. The orange amount represents the resources spent on processing the transfer overhead. Unsurprisingly, the U-Bound offers the best resource utilization available thanks to its centralized vision. On the other side of the spectrum, the absence of a mechanism to transfer the load makes L-bound the worst method for load distribution. Of the three distributed approaches, the DQN is the most performing with a greater payload and few resources spent on penalties. Although having a large average load, the P-Auction spends too much in penalties and therefore offers a payload barely over the lower bound.

Distributed systems manage to distribute the load by using the transfer mechanism. *How often and effectively do market-based and DQN approaches use the transfer mechanism?*

B. Quality of the distribution

1) Transfer and communication: Fig. 8 shows the number of successful and failed transfer queries over time. It appears that the DQN-type approach has more control over the transfer mechanism. Indeed, the success/failure rate is much more advantageous than for market-based approaches. We also find that the number of successful transfers is close to that of the P-Auction. However, we note in Fig. 7 that the resources used to pay the transfer penalties are significantly less important in the DQN method. This solution therefore adopts a task transfer strategy requiring few resources. The results regarding the transfer are therefore mixed. On the one hand, we manage to limit the number of transfers via our reward model as desired. On the other hand, we fail to eliminate unsuccessful transfer requests. This is explained by the fact that the success of the transfer action does not depend solely on the transmitting agent.

The transfer's efficiency depends directly on the number of potential recipients.*What happens if we increase the transfer range so that it can reach all areas?*

2) Impact of areas: Fig. 9 and 10 show the evolution of the transfer requests over time when the range is maximum and the comparison of the results according to transfer range respectively. As expected, increasing the transfer range allows agents to find new sales opportunities. This results in an increase in the number of successful transfers as well as a decrease in failures for market-based methods. More surprisingly, the DQN approach does not take advantage of these new opportunities to transfer more. On the contrary, the number of transfers decreases sharply leading to greater control over this mechanism. This reluctance of the system to transfer tasks is explained by the penalty incurred. Indeed, this solution transfers only high-priority tasks where the non-completion appears to the system as a greater penalty than the overhead generated. Once again, it behaves to minimize the overhead. This was already present with the selection of inexpensive tasks for the transfer (thus generating the least penalty). The greater number of transfers when fewer opportunities are present is due to the uncertainty of success. Indeed, the DQN approach does not seek to transfer more tasks, but to ensure its transfers (to avoid greater penalties). To do so, the only choice is to increase its number of requests.

So far, the results attest the great adaptability of the DQN approach. In order to evaluate the scalability and robustness of the method to the experiment parameters, we must explore different configurations. Due to the exploration space and space limitation we focus on two important features of an MRS namely the scalability and the behaviour in case of imbalance.

C. Scalability and imbalance

1) Scalability: To test the scalability of our simulations, we increase the number of robots from 7 to 20 as well as the average number of tasks per iteration from 7 to 20. The number of areas remains unchanged and has the same transitions' probabilities. Fig. 11, 12, 13 show the number of tasks completed by priority, the number of tasks failed by priority and the average load of an agent respectively. The observed results are identical to those obtained for a system three times smaller. It appears therefore that our conclusions remain relevant on a larger scale.

2) Imbalance: In order to model an imbalance, we propose the experimentation of a new set of tasks favouring one of the metrics, the CPU (the results are identical if we choose to favour the memory). This new set presented in Table III is a refined version of the previous set where tasks requiring more MEM than CPU have been removed.

TABLE III
TASKS' DATASET WITH IMBALANCE

ID	CPU (%)	MEM (%)	ID	CPU (%)	MEM (%)	ID	CPU (%)	MEM (%)
T1	5	5	T2	10	10	T3	15	10
T4	15	15	T5	20	20	T6	25	15
T7	25	25	T8	30	5	T19	30	20
T10	30	30	T11	30	25	T12	35	25
T13	35	35	T14	40	10	T15	40	40

Fig. 14, 15, 16 represent the completion rate by priority, failures by priority and the average load of the agents. Completion rates remain close to those observed for a balanced

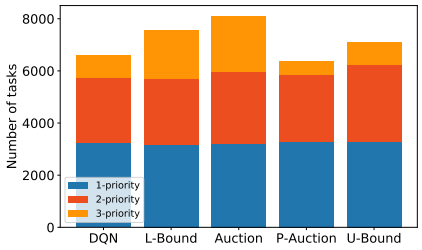

Fig. 11. Tasks completed for 20 robots

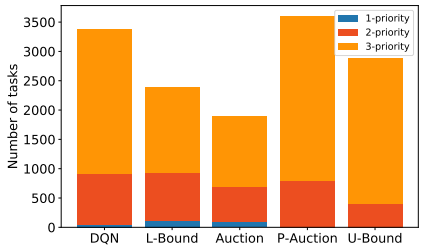

Fig. 12. Tasks failed for 20 robots

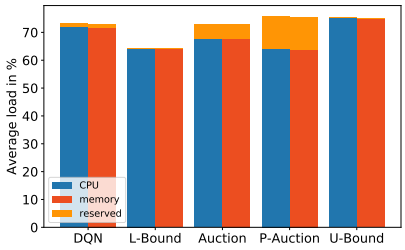

Fig. 13. Average load for 20 robots

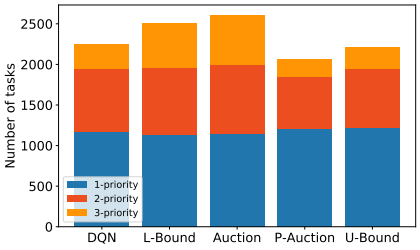

Fig. 14. Tasks completed with imbalance

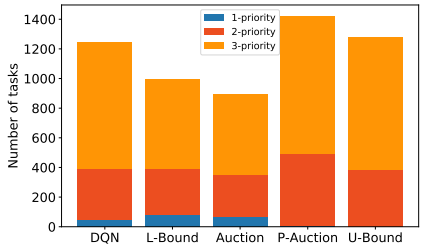

Fig. 15. Tasks failed with imbalance

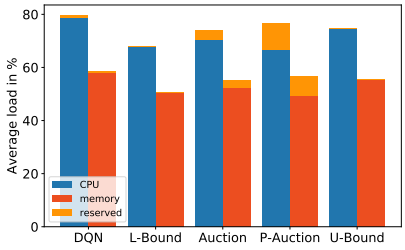

Fig. 16. Average load with imbalance

game, but resource balancing performance differs significantly. Indeed, the DQN is more efficient than the U-Bound and manages to make greater use of system's resources (77% CPU + 1% reserved vs 75% CPU and 57% MEM + $<$ 1% reserved vs 56% MEM) , that reflects a profound difference between studied approaches. On one side, we observe classical approaches (without learning) trapped by their static nature. Indeed, they are defined for a specific model and therefore cannot adapt to a different problem. Here, we find the problem of auctions' valuation. On the other hand, we note the adaptive nature of models based on RL that can learn by themselves the correct valuation of a metric. The result is a functional system that performs well without any *a priori* information.

VI. Conclusion and Future Work

In this article, we propose a comparative analysis of two types of approaches to solve the MRpTA problem that emerge to take advantage of shared computing resources within a MRS. In a distributed context, agents must spread the computational load through the transfer mechanism to compensate for local overheads. To complicate the problem, the transfer has a range limit and generates a penalty (20%).

As the complexity of the system is increasing, we have imposed a simple auction valuation algorithm to represent the impossibility of correctly valuing the bids of a real system. As far as RL is concerned, we have limited the size of the neural network architecture to make it compatible with mobile robot computing capabilities. In addition, we impose a simple reward scheme for the same reasons as auction valuation. Despite these restrictions, the use of learning is relevant. Indeed, the tested solution is able to:

- Respect a priorities system
- Schedule tasks effectively
- Develop a transfer strategy
- Adapt to a dynamic distributed context

There are however some expected limitations:

- While complying to the priorities system, we could not guarantee the execution of all *1-priority* tasks
- The distributed nature of the system does not allow to sufficiently reduce the number of unsuccessful transfers

Overall our conclusion to this study is that DQN appears as very relevant alternative.This first comparative approach to this MRpTA problem paves the way for the following future works:

- Introduce a more accurate model for heterogeneous MP-SOC (multi-core processors, GPU), a concrete simulation of robot movements and the use of real task sets.
- Study the relevance of other RL methods such as those based on the policy gradient, or other architectures such as recursive neural networks.

References

[1] B. P. Gerkey and M. J. Matarić, "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, Sep. 2004. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364904045564

[2] J. Kuffner, "Cloud-enabled humanoid robots," in *IEEE-RAS Int. Conf. Humanoid Robots*, Nashville, USA, 2010.

[3] R. Grandl *et al.*, "Multi-resource packing for cluster schedulers," in *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14*. Chicago, Illinois, USA: ACM Press, 2014, pp. 455–466. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2619239.2626334

[4] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," *arXiv:1312.5602 [cs]*, Dec. 2013, arXiv: 1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602

[5] X. Jia and M. Q. Meng, "A survey and analysis of task allocation algorithms in multi-robot systems," in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dec. 2013, pp. 2280–2285.

[6] S. Koenig *et al.*, "The power of sequential single-item auctions for agent coordination," in *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'06. AAAI Press, 2006, pp. 1625–1629. [Online]. Available: http://dl.acm.org/citation.cfm?id=1597348.1597457

[7] M. Otte, M. Kuhlman, and D. Sofge, "Multi-robot task allocation with auctions in harsh communication environments," in *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, Dec. 2017.

[8] Z. Wang *et al.*, "A task allocation algorithm based on market mechanism for multiple robot systems," in *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, Jun. 2016, pp. 150–155.

[9] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Dec. 2014, arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980

[10] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://www.nature.com/articles/nature14236

[11] A. Yahya *et al.*, "Collective robot reinforcement learning with distributed asynchronous guided policy search," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vancouver, BC: IEEE, Sep. 2017, pp. 79–86. [Online]. Available: http://ieeexplore.ieee.org/document/8202141/

[12] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv:1602.01783 [cs]*, Feb. 2016, arXiv: 1602.01783. [Online]. Available: http://arxiv.org/abs/1602.01783