Optimized distributed scheduling for a fleet of heterogeneous unmanned maritime systems

Geert De Cubber
Department of Mechanics
Belgian Royal Military Academy
Av. De La Renaissance 30, 1000 Brussels, Belgium
geert.de.cubber@rma.ac.be

Rob Haelterman
Department of Mathematics
Belgian Royal Military Academy
Av. De La Renaissance 30, 1000 Brussels, Belgium
rob.haelterman@mil.be

***Abstract*— Due to the increase in embedded computing power, modern robotic systems are capable of running a wide range of perception and control algorithms simultaneously. This raises the question where to optimally allocate each robotic cognition process. In this paper, we present a concept for a novel load distribution approach. The proposed methodology adopts a decentralised approach towards the allocation of perception and control processes to different agents (unmanned vessels, fog or cloud services) based on an estimation of the communication parameters (bandwidth, latency, cost), the agent capabilities in terms of processing hardware (not only focusing on the CPU, but also taking into consideration the GPU, disk & memory speed and size) and the requirements in terms of timely delivery of quality output data. The presented approach is extensively validated in a simulation environment and shows promising properties.**

Keywords—heterogeneous systems, multi-robot systems, cloud robotics, load balancing

I. INTRODUCTION

As robotic systems are becoming more and more intelligent, their perception, modeling, reasoning and actuation control architecture relies more and more upon a complex network of tightly interrelated processes. Moreover, these robotic systems are also more and more being deployed not in isolation, but as a part of a heterogeneous team, working towards a common higher-level goal [1]. The robotic agents thus become "edge" devices in an Internet-Of-Things framework with supporting computational infrastructure at fog and cloud level [2]. In such a collaboration scenario, the robotic agents can not only communicate between them to share information or to offload processes, but also to fog or cloud services.

This evolution paves the way for new research questions, as the question arises for each perception & control process where to optimally allocate it: on the robot that generates the input data or requires the output data, on another robot, on a fog device or on a cloud service? This question of process load distribution has been studied for decades in computer science [3], however focusing heavily on optimizing the load distribution within computing stations in function of the processing time and the communication time. These kinds of approaches cannot be easily ported to the domain of mobile robotics and especially unmanned maritime systems due to the following reasons:

- The performance of robotic perception and control algorithms is not only dependent on the CPU, but more and more also on the GPU, disk & memory speed and size.
- For robotic perception and control processes both the update frequency and the latency are important parameters to ensure the usefulness of the data produced by a process.
- Mobile robotic systems move, meaning that existing communication paths will improve or degrade over time or may even totally disappear.
- Unmanned maritime systems cover great distances, switching between communication technologies as satellite, radio, Wi-Fi or even cellular, meaning that communication parameters vary widely and also that the cost of communication needs to be considered.

In response to all these challenges, we propose in this paper a methodology that aims to optimally distribute the load of different processes over the different agents that are present in a robotic cloud architecture. The following assumptions are made:

- The presence of at least one remote cloud architecture with very high computing capabilities
- The presence of at least one remote fog architecture with high computing capabilities
- The presence of at least two maritime robotic agents with moderate computing capabilities
- The user defines for each process a number of requirements in terms of minimum framerate and maximum latency.
- Unreliable communication means between the different assets. We consider six different communication technologies for data transfer, each with their own connection parameters with respect to

bandwidth, latency, connection distance, quality of service, etc.:

1. Satellite communication
2. Radio telecommunication
3. Mobile broadband (cellular)
4. Wi-Fi
5. Ethernet
6. Fiber-optic communication

The methodology proposed in this paper is based on a local optimization process where each agent communicates to the agents it is connected with in the network (which is a subset of all the agents) in order to find the best distribution of processes. As the agents only transfer data on a local level and no load-balancing data is transferred across multiple hops, the result is 'only' a local optimization of the process load, which may (very likely) still be sub-optimal in a global sense. With this regard, the presented approach differs from global load balancing techniques [4] that distribute the processing charges across a whole network, based upon an exact knowledge of all properties of all agents in the network. While capable of providing a more optimal solution, such a global approach towards load balancing is not convenient for the case studied in this research paper, being a fleet of heterogeneous unmanned systems. Indeed, as the unmanned systems are not all directly interconnected, it would require a considerable amount of networking overhead to share the information about all the agents with each other in some sort of central database, as performed by [5]. Moreover, in maritime operations one needs to consider that agents can get totally disconnected from the network for some time, which means that critical processes must be executed locally in any case and that on other occasions, when limited communication is possible, it will likely not be possible to connect to such a central database server. For these reasons, it was decided to develop the load balancing mechanism as presented in this paper as a decentralized approach, running independently on each agent in the network and only relying on existing agent-to-agent communication.

Compared to other decentralized multi-agent collaboration frameworks [6], the presented approach differs in terms of objectives. Whereas classical decentralized multi-agent collaboration frameworks generally aim to optimize the successful completion of some higher or lower-level task (while generally ignoring how these tasks are decomposed into processes that need to run on the different agents), the presented approach aims to optimize the successful 'outcome' of each process (providing quality data on time with minimal strain on computing infrastructure), while it ignores the higher or lower-level tasks these processes belong to. As a result, it is perfectly possible to combine the approach presented in this paper with a classical decentralized multi-agent collaboration framework in order to combine both optimization objectives simultaneously.

The research presented in this paper has been funded by the Belgian Royal High Institute for Defence (RHID) under contract number DAP19/08.

II. METHODOLOGY

A. Overview of the approach

The global strategy of the presented load balancing methodology is summarized by the pseudo-code of Algorithm 1.

The main idea is to interrogate each of the connected agents (other systems of the maritime fleet where the agent in subject is in contact with) whether they have computing resources available that would be better suited to offload one or more processes to. Performing such an interrogation and a transfer of processes is a procedure that consumes valuable computing resources and communication bandwidth by itself. Therefore, a primordial check is made whether it is at all necessary to apply any changes.

The presented algorithm seeks to optimize 9 different parameters simultaneously for each agent in the network:

- The relative load on the incoming communication channel, expressed as the current bandwidth usage compared to the maximum bandwidth capacity of the incoming communication channel. [%]
- The relative load on the outgoing communication channel, expressed as the current bandwidth usage compared to the maximum bandwidth capacity of the outgoing communication channel. [%]
- The framerate, expressed as the current framerate of the process compared to the required minimal framerate, as defined by the user. [%]
- The latency expressed as the current latency (in seconds) of the process compared to the required maximal latency, as defined by the user. [%]
- The communication cost in euro or dollar.
- The relative load on the Central Processing Unit (CPU), expressed as the current CPU usage compared to the total CPU capacity. [%]
- The relative load on the Graphics Processing Unit (GPU), expressed as the current GPU usage compared to the total GPU capacity. [%]
- The relative disk usage, expressed as the current disk usage compared to the total disk capacity. [%]
- The relative memory usage, expressed as the current RAM memory usage compared to the total RAM memory capacity. [%]

Note that all of the above-mentioned parameters need to be minimized, except for the framerate, that needs to be maximized.

In the following paragraphs, we will further discuss how these parameters are calculated and how they fit into the framework of Algorithm 1.

```
01 FOR each Agent
02   IF CheckProblems(Agent) = TRUE
03     FOR each Process
04       FOR each ConnectedAgent
05         L_Comm = Calculate_Relative_Load_On_Communication
06         L_FPS = 1 / Calculate_Projected_Framerate
07         L_LAT = Calculate_Projected_Latency
08         L_Cost  = Calculate_Projected_Communication_Cost
09         L_CPU = Calculate_ Projected_Relative_Load_On_CPU
10         L_GPU = Calculate_ Projected_Relative_Load_On_GPU
11         L_DISK = Calculate_ Projected_Relative_Load_On_Disk
12         L_RAM = Calculate_ Projected_Relative_Load_On_RAM
13         STORE ConnectedAgent with lowest L_Comm, L_FPS, L_LAT,
                              L_Cost, L_CPU, L_GPU, L_DISK, L_RAM
14       ENDFOR
15       TRANSFER Process to optimal ConnectedAgent
16     ENDFOR
17   ENDIF
18 ENDFOR
```

Algorithm 1. Pseudo-code for the multi-agent load-balancing algorithm

B. Detailed discussion

Lines 01-02:

For each agent in the fleet, a check is made whether there are any problems that would necessitate a transfer of a process running on that agent. Identified problems can be:

- The combined processes running on the agent demand more bandwidth from the incoming or outgoing communication channel than is available.
- The combined processes running on the agent demand more CPU, GPU, disk or memory capacity than available on the agent.
- The required minimal framerate, as defined by the user, is not reached. This can be for example due to a slow CPU that cannot process the data in real-time.
- The required maximal latency, as defined by the user, is not reached. This can be for example due to a slow CPU that cannot process the data in real-time, but also due to a communication channel that is too slow.

Lines 3-4:

For each process p_i running on the agent, check whether it is possible and interesting to transfer that process to one of the other systems a_j the agent is connected to.

For evaluating this, we calculate one by one the 9 optimization parameters defined above for a virtual situation where p_i is running on a_j. This calculation is performed in lines 5-12.

Line 5:

The relative load on the communication channels is calculated (both incoming and outgoing) by comparing the used bandwidth of the communication channel to the maximum bandwidth of that channel. Two problems need to be solved for enabling this calculation: the best communication channel needs to be chosen and the new bandwidth of this communication channel needs to be estimated.

The algorithm does this by evaluating for each of the 6 communication methodologies their availability (e.g. wired communications will not work between mobile agents, WiFi or cellular will likely not work well far at sea) and choosing the communication technology with the highest available bandwidth. As it concerns a virtual situation (process p_i is not yet actually running on a_j), these communication parameters need to be estimated using a model. In order to do this, we use:

- For satellite communications, a constant service model
- For radio communications (both generic radio and WiFi), the two-ray propagation model [7]. This model is known to be well-suited for maritime applications.
- For mobile broadband communications, we use the same two-ray propagation model, but then not between agents, but between the agent and the cell tower.
- For ethernet and fiber-optic communication, we use the standard wired communication models [8].

This calculation is made once for the incoming and once for the outgoing communication channel and the respective loads on the communication channels are summed to provide a final ‘cost’ for the transfer with respect to communication.

Line 06:

The projected framerate of process p_i running on agent a_j is estimated, based upon - the computational load of process p_i on the current agent and the relative performance between the compute capabilities of the current agent and agent a_j, as indicated in [9]. The transfer cost with respect to the framerate is then defined as the inverse of the estimated framerate divided by the minimum requested framerate. Taking the inverse is applied in order to come to a cost function to be minimized, whereas the framerate needs to be maximized.

Line 07:

The projected latency of process p_i running on agent a_j is estimated based upon:

- the estimation of the latency due to the delayed arrival of the data on the incoming communication channel (see line 05)
- the estimation of the latency due to the processing time taken to process the data, which is the inverse of the framerate (see line 06)
- the estimation of the latency due to the delayed transfer of the data on the outgoing communication channel (see line 05)

Line 08:

The projected communication cost is estimated using a fixed price per Megabyte accorded to each of the 6 communication channels. This may not always be a perfect representation, due to subscription plans that are somewhat bandwidth-independent, but in most usage scenarios, it provides a 'good enough' means for estimating the communication costs. The communication bandwidth is estimated as explained in line 05.

Line 09:

The projected relative load on the CPU is estimated by adding the relative CPU load of the process p_i to agent a_j and dividing this by the CPU capabilities of agent a_j. Obviously, this approach is only correct in a single-core and non-hyperthreading scenario. Modern multi-threaded CPU's do scale differently, but the differences are not so big that they make the use of this model impossible for estimating the CPU load of processes in the context of this load-balancing approach.

Line 10-12:

A similar approach to the CPU (line 09) is followed for the GPU load and the load on the disk and the RAM memory. Whereas this works quite well for the disk and RAM memory (which follow a linear model of filling up), this doesn't really hold true anymore for the GPU. Indeed, due to the highly parallelized architecture, GPU's scale very non-linearly under load. As it is very difficult to model this behavior [10], we have not developed this any further and have applied the same linear model, but we will see in the results section that this has some repercussions.

Line 13:

The connected agent with the lowest total cost is selected. As we are dealing here with a multi-objective optimization process, ideally an intelligent multiple criteria decision making algorithm should be adopted to come to an optimal solution. However, this is not the focus of this research paper. With this paper, we just want to show a proof-of-concept for the load balancing approach for a fleet of heterogeneous unmanned maritime systems. Therefore, we just adopted linear scalarization with unitary weights to select the agent with the lowest cost.

This is a rather simplistic (and presumably not the best) approach, but with this approach we wanted to test the applicability of the presented load-balancing technique, with the prospect of improving the optimization algorithm later on.

Line 15:

Finally, the transfer of the chosen process to the identified optimal agent is executed. This procedure involves not only the launch of the process on the remote agent, but also the termination of the process on the current agent and the setup of new connections between agents.

III. VALIDATION

A. Validation methodology

In order to validate the presented methodology, we have chosen to develop a simulation tool. This tool allows for the simulation of multiple interconnected cloud, fog and edge agents with a range of processes running on them.

In order to evaluate how the approach scales with the number of agents involved, we present two different validation tests: one with a relatively low number of agents and one with a relatively high number of agents.

For each of the experiments, a number of agents was released inside an area of 1000 x 1000 km. At each iteration, the mobile agents move, using a randomized motion pattern. This movement obviously also influences the communication parameters between the agents, which are distance-dependent.

At every few iterations (a random number), a new process is spawned, with random values and requirements with respect to each of the 9 above-mentioned parameters. As such, the load on the network gradually increases over time. For this paper, we consider simulations with 1000 iterations.

As there are quite some random parameters (all following a uniform distribution) in the simulation process in order to test all kinds of eventualities, it is required to perform a large amount of iterations to acquire consistent data about the load-

balancing performance. Therefore, the simulations are repeated typically 100 times.

In order to assess the performance of the presented load-balancing technique, a comparison is made to a standard situation, where this technique is not used.

B. *Results for a low number of agents*

Here we present results as they were obtained with the presented load-balancing technique for

- 1 cloud agent (a powerful on-shore cloud infrastructure),
- 2 fog agents (typically 2 motherships with important computing infrastructure)
- 4 edge agents (unmanned vessels acting as sensor platforms with important needs in terms of computing power, but with limited on-board computing power)

On all the figures, the blue line represents the result of the presented load-balancing technique, whereas the red line represents the result of the standard situation for benchmarking.

Note that all the figures represent results as relative (unitless) data, following the definition of the optimization parameters given above. Therefore, the Y-axis has been left blank for reasons of clarity of the figures.

The results that are displayed here below show averaged results over 100 simulations and over all agents in the simulation.

Fig. 1 shows the relative bandwidth usage both for the proposed and the baseline approach. As can be noticed, the proposed algorithm clearly succeeds in limiting the overall bandwidth usage considerably, notwithstanding the ever increasing number of processes. Over a few iterations, the bandwidth usage of the proposed approach is actually higher than with the baseline approach. This is due to the algorithmic communication overhead required for exchanging the data.

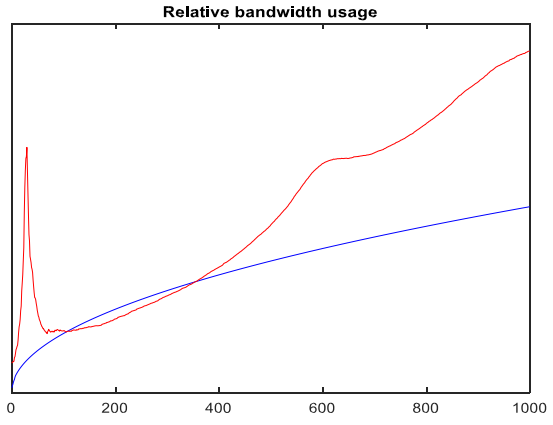

Fig. 1. Relative bandwidth usage for small number of agents (blue: proposed approach; red: baseline aproach)

Fig. 2 shows the relative latency compared to the user requirements both for the proposed and the baseline approach, whereas fig. 3 shows the evolution of the framerate across the iterations.

From Fig. 2, it can be noted that the presented approach obtains in a consistant manner a significantly lower latency, which is really important in real-time applications.

Fig. 3 shows that the proposed algorthm generally succeeds in achieving higher framerates than the baseline approach. The spikes in the red baseline curve on Fig. 3 are probably due to outliers values that are not sufficiently averaged out. This indicates that we should probably record even more than 100 simulations.

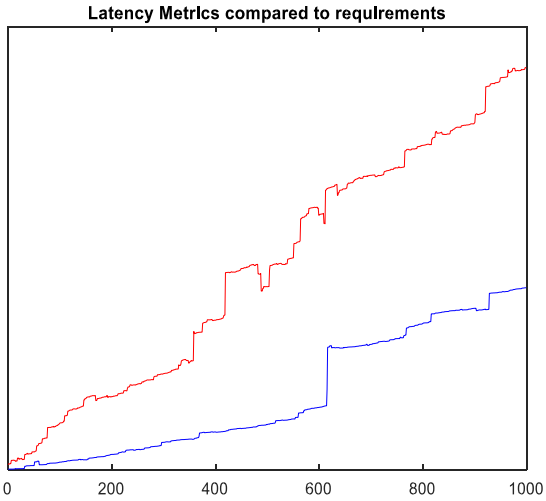

Fig. 2. Relative latency compared to user requirements for small number of agents (blue: proposed approach; red: baseline aproach)

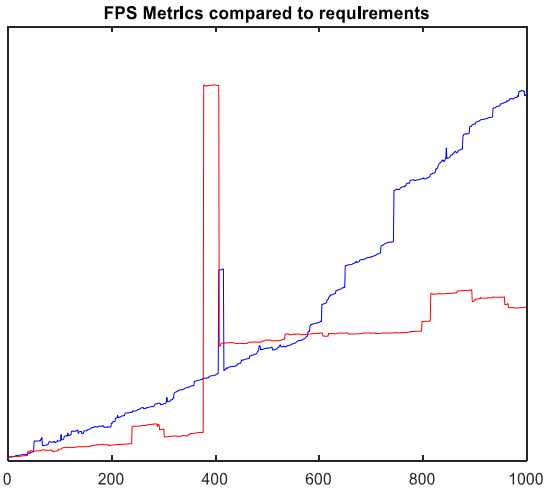

Fig. 3. Relative framerate compared to user requirements for small number of agents (blue: proposed approach; red: baseline aproach)

The connection cost for the presented approach is lower than the baseline connection cost, as indicated by Fig. 4. However, it is also clear that the connection cost for the proposed approach follows a more or less linear model, whereas the baseline approach shows a more asymptotic behavior, which would mean that the proposed approach would probably be costlier at higher iterations. The reason for this behavior is still under investigation, but it is probably due to the fact that the connection cost is currently no factor in deciding whether to transfer a process (in step 01-02).

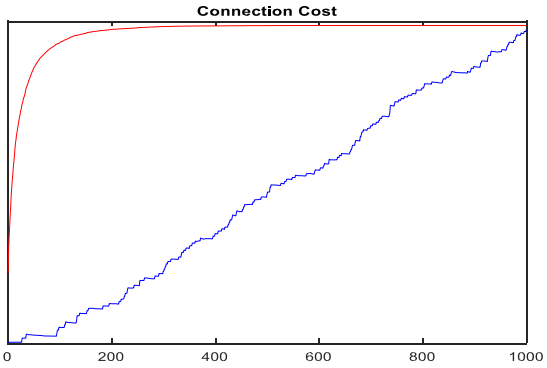

Fig. 4. Connection cost for small number of agents (blue: proposed approach; red: baseline aproach)

Fig. 5 compares the (averaged) relative load on the CPU, GPU, disk and memory. As it can be clearly noted, the proposed approach achieves better results in all cases, putting less cumulative strain on the computing hardware and still reaching a better performance in the end (as indicated by Fig. 3). However, it should be noted that for the GPU, the curves of the proposed and baseline approach are quite close to one another, indicating that the methodology doesn't really generate great benefits here. This is probably due to the fact that it is very difficult to model the scaling of the GPU load and performance (see discussion at line 10-12).

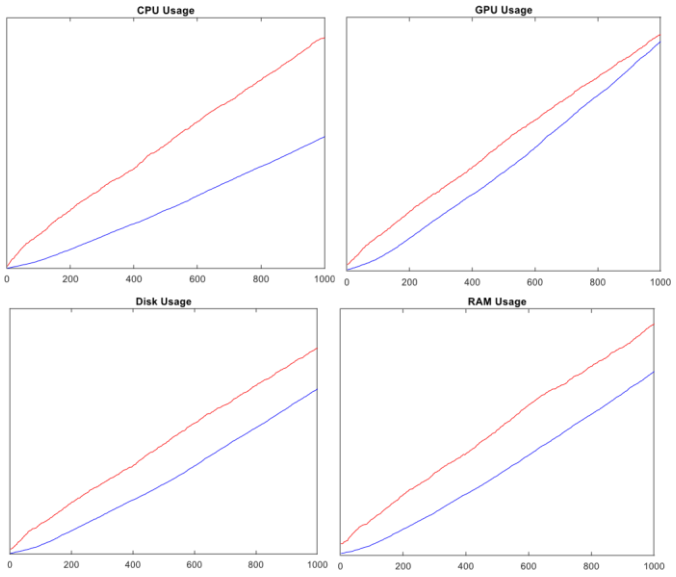

Fig. 5. Relative load on the CPU, GPU disk and memory for small number of agents (blue: proposed approach; red: baseline aproach)

C. Results for a high number of agents

Here we present results as they were obtained with the presented load-balancing technique for

- 2 cloud agents (powerful on-shore cloud infrastructures)
- 5 fog agents (typically motherships with important computing infrastructure)
- 50 edge agents (unmanned vessels acting as sensor platforms with important needs in terms of computing power, but with limited on-board computing power)

Fig. 6 shows the same graphs as Fig. 1-4, but now for this much higher amount of agents. Compared to the previous results with a low number of agents, the relative bandwidth usage is now consistently lower with the proposed method, while also the latency is lower and the framerate is higher. The connection cost now shows for both instances an asymptotic behavior, with only a marginally lower cost for the proposed method.

The results of Fig. 6 show that the proposed methodology scales very well with the number of agents, as the three most important parameters (bandwidth usage, average relative latency and average relative framerate) are clearly better with the proposed approach. It may be noted that the relative bandwidth usage is significantly lower, while the connection cost changes only moderately. This implies that the algorithm does seem to favor more powerful, but also costlier, connection means.

In terms of processing time, that the simulation presented in section C with 57 agents is about 30 times slower than the one in section B with 7 agents, indicating that the order of the algorithm is around $\frac{1}{2}O(n^2)$.

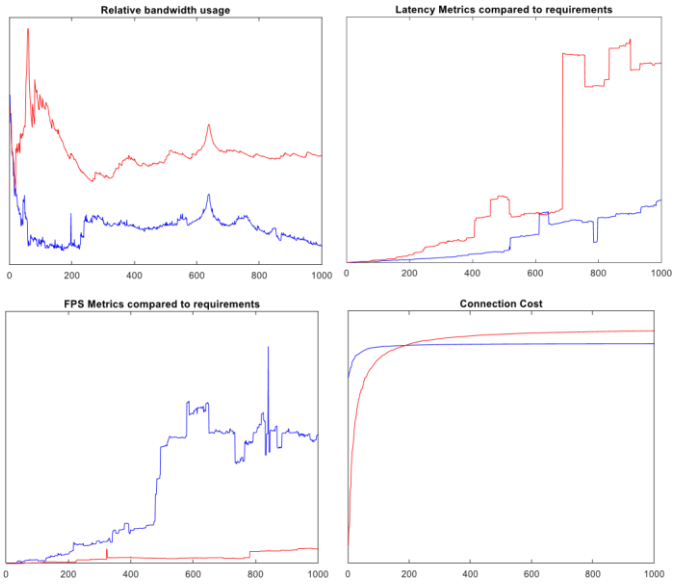

Fig. 6. Top Left: Relative bandwidth usage; Top Right: Relative latency compared to user requirements; Bottom Left: Relative framerate compared to user requirements; Bottom Right: Connection cost. All results given for a high number of agents (blue: proposed approach; red: baseline aproach)

Fig. 7 shows the same graphs as Fig. 5, but for the high amount of agents. As can be noted, Fig. 5 and Fig. 7 show very similar behavior for the relative CPU, memory and disk load. However, for the relative GPU load, we now see a worse behavior with the proposed approach after 400 iterations. The reason for this lies again in the fact that our linear load estimation model is not adequate for the GPU and that we thus fail to optimize this parameter.

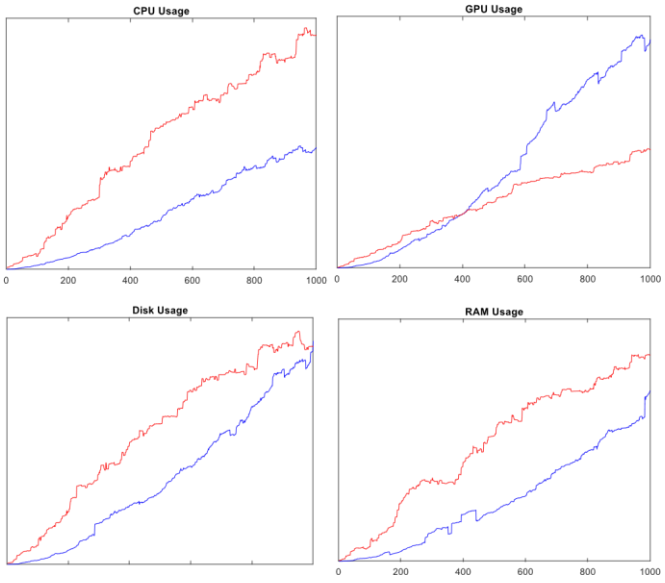

Fig. 7. Relative load on the CPU, GPU disk and RAM memory for high number of agents (blue: proposed approach; red: baseline aproach)

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a new methodology towards load balancing. The method is specifically geared towards managing processes across a fleet of heterogeneous unmanned maritime systems. The algorithm employs a decentralized approach where each agent in the system communicates with its connected agents to determine which process could best be offloaded to what other agent, in terms of available computing resources, but also in terms of network capabilities, connection cost and requirements in terms of latency and framerate imposed by the end user.

The proposed methodology has been validated in a simulation environment, first with a small number of agents to understand the dynamics of the system and then with a larger amount of agents to validate the capability of the algorithm to upscale towards more complex problems with dozens of interconnected agents. These simulations have shown that the presented approach achieves significantly better results than a baseline approach without load scheduling.

The presented paper and results must be seen as a proof of concept for the methodology. Based on the conclusions of this work, future work is now identified to improve the algorithm further.

In the first place, it was shown that a better methodology is required for modeling the load of processes on a GPU. Secondly, the naive multi-objective optimization function that was used for this work should be clearly replaced by a better methodology that allows for more fine-tuning and a better optimization. Finally, it is the idea to implement this methodology on a real fleet of unmanned maritime systems, such that experiments in a realistic environment can be performed.

REFERENCES

[1] M. Marques, R. Parreira, V. Lobo, A. Martins, A. Matos, N. Cruz, J. Almeida, J. Alves, E. Silva, J. Będkowski, K. Majek, M. Pełka, P. Musialik, H. Ferreira, A. Dias, B. Ferreira, G. Amaral, A. Figueiredo, R. Almeida, F. Silva, D. Serrano, G. Moreno, G. De Cubber, H. Balta, H. Beglerović, S. Govindaraj, J. Sanchez, M. Tosa, "Use of multi-domain robots in search and rescue operations – Contributions of the ICARUS team to the euRathlon 2015 challenge", OCEANS, China (2016)

[2] P. Simoens, M. Dragone, A. Saffiotti, "The Internet of Robotic Things: A review of concept, added value and applications", Int. Journal of Advanced Robotic Systems, Vol 15, (2018)

[3] D.G. Feitelson, "Workload Modeling for Computer Systems Performance Evaluation", Cambridge University Press (2015)

[4] A.M. Alakeel, "A Guide to Dynamic Load Balancing in Distributed Computer Systems", IJCSNS International Journal of Computer Science and Network Security, Vol.10 No.6, 2010

[5] V. Thakur, S. Kumar, "Load Balancing Approaches: Recent Computing Trends", International Journal of Computer Applications, Vol. 131 – No.14, 2015

[6] F. Ali1, R. Z. Khan, "The study on load balancing strateges in distributed computing system", International Journal of Computer Science & Engineering Survey (IJCSES) Vol.3, No.2, 2012

[7] P. Srinivasa Rao, V.P.C. Rao, A. Govardhan, "Dynamic Load Balancing With Central Monitoring of Distributed Job Processing System", International Journal of Computer Applications, Vol. 65– No.21, 2013

[8] D. Doroftei, G. De Cubber, E. Colon, Y. Baudoin, "Behavior based control for an outdoor crisis management robot", Proceedings of the IARP International Workshop on Robotics for Risky Interventions and Environmental Surveillance, 2009

[9] D. Serrano López, G. Moreno, J. Cordero, J. Sanchez, S. Govindaraj, M. M. Marques, V. Lobo, S. Fioravanti, A. Grati, K. Rudin, M. Tosa, A. Matos, A. Dias, A. Martins, J. Bedkowski, H. Balta, G. De Cubber, "Interoperability in a heterogeneous team of search and rescue robots", In: Search and Rescue Robotics-From Theory to Practice, InTech Open, 2017

[10] H. Balta, J. Bedkowski, S. Govindaraj, K. Majek, P. Musialik, D. Serrano, K. Alexis, R. Siegwart, G. De Cubber, "Integrated Data Management for a Fleet of Search - and - rescue Robots", In: Journal of Field Robotics, Vol. 34, No. 3, pp. 539-582, 2017

[11] A. Habib, S. Moh, "Wireless Channel Models for Over-the-Sea Communication: A Comparative Study", A Applied Science, Vol. 9, No. 443, 2019

[12] D. Hulens, T. Goedemé, J. Verbeke, "How to choose the best embedded processing platform for on-board UAV image processing?", in: proceedings International conference on computer vision theory and applications - VISAPP 2015.

[13] M.T. Amaris Gonzalez, "Performance prediction of applications executed on GPU's using simple analytical model and machine learning techniques", PhD. Thesis, University of Sao Paulo, 2018