Synthesized A* multi-robot path planning in an indoor smart lab using distributed Cloud computing

Morteza Kiadi[1], José R. Villar[1*], and QingTan[2]

[1] University of Oviedo, Oviedo, Spain
Mkiadi2002@yahoo.com, villarjose@uniovi.es
[2] Athabasca University, Athabasca, Canada
qingt@athabascau.ca

Abstract. Finding the shortest path for an autonomous robot in static environments has been studied for many years and many algorithms exist to solve that problem. While path finding in the static setting is very useful, it is very limiting in real world scenarios due to collisions with dynamic elements in an environment. As a result, many static path planning algorithms have been extended to cover dynamic settings, in which there are more than one moving objects in the environment. In this research, we propose a new implementation of multi agent path finding setting through A* that emphasizes on the path finding through a centralized meta-planner that operates on the base of Bag of Tasks (BoT), running on the distributed computing platforms on the cloud or fog infrastructures and avoiding dynamic obstacles during the planning. We also propose a model to offer a "Multi-Agent A* path planning as-a-Service" to abstract the details of the algorithm to make it more accessible.

Keywords: A* algorithm, Pathfinding, Fog Computing, Kubernetes

1 Introduction

Robot navigation is the process of finding and executing a path from the initial location towards a target position while avoiding obstacles [1]. Based on the availability and the knowledge of the environment, path planning is scoped at the local or global level [2]. While local level refers to modifications to a predefined path made by the robot based on information gathered from the available sensors [1], the global level is responsible for producing a valid path to each robot. When the obstacles are static and the start and goal cells are known beforehand, we calculate the path (based on an ideal criterion such as the "shortest" path), which is known as "global" path planning. However, this

method cannot help for scenarios where the obstacles are moving, or the goal is not fixed. As will be seen in the next section, there are many existing solutions thar combine local and global path planning [3].

This study analyses the hybridization of local and global planning in multirobot environments, considering collision avoidance. From one viewpoint, our solution is similar to the offline path planning as we use the prior environmental knowledge while we also would like to consider other agents' moves during the "planning" phase. Our solution proposes implementing a central "meta-planner" module, running ultimately on the cloud computing facilities to minimize the communication overhead between agents while taking advantage of abundant processing power in the cloud computing to apply the heuristic algorithms to avoid collisions. This research shows a proof of concept for the meta-planner implementation as well as introducing the architecture of the cloud-based solution. We propose a cloud native (CN) multi-agent (MA) A* path planner (A*PP), in short (CNMA-A* PP).

The structure of the study is as follows. The next Section copes with the related work. Sect. 3 completely describes the hybridized global multi-robot path planning. The experimentation design and results are introduced in Sect. 4, including some discussion. Finally, the conclusions are drawn.

2 Related Work on Multi-Robot Path Planning

Path planning is an important and essential issue for the navigation of autonomous robots among many other use cases[4] . The studies of path planning started in the late 60s and many different algorithms have been proposed [5]. In this path different frameworks to solve a multi-robot path planning have been proposed such as problem reduction through answer set programming [6] and SAT [7]. In the Optimal Multirobot Path Planning on Graphs[8], the problem of multi-robot path planning has been discussed over four minimization objectives: the makespan (last arrival time), the maximum distance (single-robot traveled), the total arrival time, and the total distance. These objectives are pairwise distinct and NP-hard to optimize, as a result we can see suggestions to solve optimal MPP by finding effective near-optimal algorithms[8]. The solution proposed in [8] to tackle such multi-objective minimization problem is to create a one-to-one mapping between MPP and that for a multicommodity network flow problem by translation of the MPP problem into an integer linear programming (ILP) model solvable using an ILP solver.

In operational research, these types of problems have been traditionally solved based on linear and/or integer programming. Such approaches inevitably place restrictions on the form of the objective function and/or the constraints by being linear. Other non-linear solutions rely on the differentiability of the problem in its nature. Furthermore, such traditional methods all assume deterministic quantities and have problems dealing with any stochastic effects [9]. In more complex settings, swarm intelligence optimization methods, such as Genetic Algorithms or Ant Colony Optimization among others, have been successfully to find the shortest path in more complex settings[10]. The ACO-A* is another proposal to use ACO metaheuristic to suggest the order of traveling and then use A* to find the shortest path pair-wise between two cells[11]. Our solution is similar to ACO-A* from this viewpoint that it has two phases however, we use heuristic than meta-heuristic in our method as in our setting we do not face with stochastic and

unknown environment. Collision avoidance through danger immune algorithm [12] has presented how different information systems such as global positioning system (GPS), automatic identification system (AIS) and Automatic Radar Plotting Aid (ARPA) is widely used in collision avoidance system on most merchant ships. Our proposal is different from this viewpoint that we do not have positional systems or real time sensors to report the location of robot in real time. The proposed solution in the Concurrent Goal Assignment and Collision-Free Trajectory Generation for Multiple Aerial Robots[13] is very similar to our proposed solution with the exception that we use robots in the indoor settings that aerial robots flying in different altitudes. The similarity comes from the fact that Benjamin Gravell and et al.[13] suggest Constrained Collision Detection Algorithm(CCDA) and Constrained Collision Detection Algorithm with Delay Times(CCDA-DT) to resolve the collision which are similar to our approach to find the collision by creating a matrix of time-moves and introducing the "wait" action to avoid collision.

By far, graph-based algorithms are the most widely used methods in global path planning [1] in order to find the shortest path. Examples of these algorithms include [14]: breadth-first and depth-first search, the Dijkstra's algorithm, the Bellman-Ford algorithm or the Floyd-Warshall algorithm. Nevertheless, one of the most competitive algorithms is the A*, which solves the single-source shortest path problem for nonnegative edge costs. Our solution has extended the implementation of the A* algorithm in the in the Artificial Intelligent book [15] and implemented by the simpleai [16]library. The idea of simultaneous task assignment and planning (STAP) problem [17] sounds a promising approach to extend our solution to a more dynamic and unpredictable approach with randomly assigned costs to each path in a graph route. However, the approach in STAP different from our solution provided that, each robot has a local reactive collision detector to avoid collision with dynamic obstacles. In our proposed solution we do not use local sensors and we do have dynamic assignment of the robots to destinations, like the way STAP works.

2.1 A* Algorithms

In the informed search algorithms such as A*, we rely on a function called the "heuristic function", to help the algorithm to pick the next cell to explore based on its "closeness" to the goal state in the entire path-finding process. A heuristic function is all about the trade-off between its accuracy and its speed [18]. One example is to try to estimate the "best heuristic" and then incorporate that into the A*. This method works if the search process is not time sensitive. For example, one can make use of the output generated by the backtracking techniques mentioned in [19] as heuristic values. The backtracking technique is useful when we do not have much knowledge about the topology of the environment, and we would like to find the state values by reinforcement learning and trials and errors. This technique is based on the optimization of the Markov decision-making process and tweaking the model's hyper-parameters. This technique approximates the real distance from the goal and as a result, the evaluation function produces the successors for the optimal path, obviating entirely the need for search[20].

A* algorithm pitfalls

The A* algorithm has the following shortcomings or limitations: 1) slow search in large scale path search. For example to get the optimal path in a 100*100 grid, at least

513 nodes need to be searched of [18] 2)The A* is only useful when there is some domain knowledge about the environment 3) Finding the right estimate for the heuristic function is tricky and it impacts the performance of the algorithm drastically 4) In large space searches, the algorithm needs lots of memory and 5) A* algorithm assumes one node is moving at a specific point in time. That is not a suitable algorithm for multi-node and dynamically changing environments.

The above issues, specifically the last one, motivates the researches to think about making a better version of A* algorithms, that it is the subject of the next section.

3 Solution Design Approach and Features

This study proposes utilizing the A* algorithm in a multi-agent setting in order to obtain a multi-robot path planning, that is, simultaneously obtaining a collision free path planning for each of the robots. We use the principle of Bag of Tasks (BoT) [21], where each agent runs the A* algorithm independently (the Agent's planning phase) and after all the agents are done with their planning, a module that we call it "meta-planner" starts modifying the results of independent tasks (the refinement phase) to create (synthesizing phase) a cohesive plan that works for all agents, in this case, a collision free path for each agent. Figure 1 shows the block diagram of this idea. Moreover, this procedure has been designed and implemented "as-a-service", finding a collision free path for multi-agent systems.

Fig. 1. The phases of a meta-planner to realize a multi-robot path with no collision.

Due to the need for a path planning for each robot, we need to perform A* for each of them and then analyze the results. Actually, the planning and the refine stages could be integrated by synchronizing the different A* running in parallel. To do so, several modifications to the A* algorithm are needed. Moreover, our implications to design a solution "as-a-service" suggested moving to a different solution path.

Alternatively, we opted to run each A* independently, merging their result and running the following stages afterwards This solution makes use of a distributed container scheduling open source project called Kubernetes, which has been successfully used in different initiatives like Cloud manufacturing [22] and distributed containerized serverless architectures[23]. Kubernetes is one of the well- adopted platforms when it comes to Cloud Native Applications (CNAs). It is the path to make a cloud based solution that is elastic, self-contained deployment, no lick-in to a cloud provider, cross platform, automated infrastructure management and containerization[24] We extend the idea of scheduling tasks in the fog computing by BoT [21] to run on Kubernetes and we suggest a new cloud based service for a multi-robot A* path planning based on the CNA principles. The categorization and taxonomy of distributed problem solving and planning detailed [25] have been considered in this research. Moreover, we have considered all the movements of the robots to take one slot of time and that all of them have the same speed. The basic movements can be configured to be the main cardinal

directions or extended with the main diagonal as well. The complete solution steps to run in the Kubernetes platform have been shown in Table 1.

Table 1. Multi-agent A* path planning as a service

1 **Containerize** the A* algorithm
2 **Setup** the YAML manifest files to describe the run-time environment
2-1 **Set** the maze configuration
2-2 **Set** the number of agents and their valid movements
3 **Launch** the Kubernetes components (Pods. ConfigMaps, …) with A* containers in Step 1
4 **For each** agent run one A* algorithm
4-1 **Run** A* algorithm for each agent in a Pod
4-2 **Store** the shortest path for each agent in the shared storage
5 **While** the meta-planner has not done:
5-1 **Unify** the path lengths
5-2 **Map** the **wait** and **time** factors to the produced A* paths
6 **For each** item in the map:
6-1 **Run** the meta-planner **Heuristics** by:
6-1-1 **Detecting** the collisions by comparing t_n:(x, y) of each agent
6-1-2 **Introduce** the wait action and shift the next moves accordingly whenever a collision is detected
6-1-3 **If** the path is blocked by another agent **then** set the blocking agent to a lower priority
7 **Return** the final paths

3.1 Refinement and Synthesize Phases

The first step in the refinement phase is to unify the length of the path to the same number. To do so the length of the longest path is determined, then padding the shorter paths with “no move” (step 5-1 in Table 1). The next step is to represent the paths in terms of cells and time slots. A path will be represented as a sequence of pairs like **time: (cell x, cell y)** (step 5-2 in Table 1). Without losing generalization, the robots are considered having the dimension of one cell. After all, in the synthesize phase the collision detection tries (steps 6-1-1 and 6-1-2 in Table 1) to detect cells included in more than one path at the same time units. In addition, in this phase, we detect the agents that block other agents' paths (step 6-1-3 in Table 1). Both actions in the synthesizing phases are achieved by the heuristic logic, implemented in the meta-planner module.

3.2 The heuristic of the meta-planner

To better understand how heuristics of the meta-planner works, we need to elaborate on the details of its internal functions. The meta-planner injects three new elements to the path produced by a pure A* algorithm: 1) the wait action 2) temporal element 3) path-blocker detector. These tools are used in the meta-planner heuristic as they are described in the following paragraphs:

- If a cell is going to be taken by more than one agent at the same time, one of them must wait. We introduce the "time-step" concept to the solution in the refinement phase to make sure such a goal is achievable in the synthesize phase, by making each "time unit" equal to each move. So, at timeslot 1 **(t1)**, we have n-move (where n is equal to the number of agents), and in **t2** we have another n-move, and so on. The agents need a different number of moves to reach their destinations (as they have different start and destination cells). When all paths are reported to the meta-planner, it unifies their sizes (practically by adding "no move" action to the end of shorter paths). The selection of the agent to wait in our solution is completely random but it could be based on a more advanced priority system.
- There is a possibility that the destination cell of an agent blocks other agents' paths. We do not manipulate or modify the decisions that are made by A*. The reason is A* already has proved itself as one of the most efficient path planners. We respect the A* quality in finding the shortest paths but we detect the blocking moves and delay those moves in favour of other agents that need those cells. So the path shapes in our solution are not changed.

3.3 Solution Architecture of CNMA-A* PP

The solution described in the previous subsections can be augmented using the Kubernetes platform by extending the meta-planner and A* executions to a cloud-based distributed service offering. We call this proposed solution as "CNMA-A*PP' to emphasize on its cloud-native nature, multi-agent A* path planning. The "CNMA-A*PP' converts a standalone A* single agent algorithm that works in static settings to a cloud-based, configurable, multi-agent A* global planner. To do so, each agent is mapped to one Kubernetes Pod to execute the A* algorithm independently (planning phase of the meta-planner). The Pods run in parallel and in a distributed manner, reducing the total service time. The results from each A* runs shall be saved in storage that is shared among the Pods. The meta-planner running the refinement and

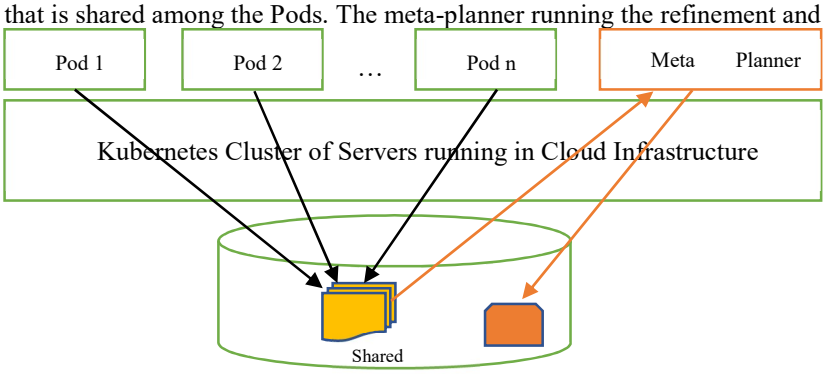

Fig. 3. A scheme of the Cloud Native-based design for a multi-robot path planner as-a-service

synthesizing processes also run in a Pod. Requesting such a service is realized entirely in the form of Kubernetes manifest YAML files describing the environment maze

setup, number of agents, allowable moves, cost of moves, etc. The solution should launch a set of infrastructure components such as Pod(s) or ConfigMaps to realize a "multi-agent A* path finding-as-a-service". The architecture of this solution is shown in Figure 3. The "CNMA-A*PP' agrees with the principle of composability that is about employing the same architecture to deploy self-managing service compositions or applications using the microservice architectural pattern [26]

4 Experiment and Results

In this section, we present a proof of concept (PoC) implementation of our idea, i.e. CNMA-A*PP. The PoC is based on three agents, starting in different start points and targeting to different end points. We have intentionally positioned the start and end points to increase the chance of conflict to test our meta-planner performance and we have purposefully set the end cell of one agent in the middle of another agent's path to block the path. Figure 2 shows the maze, as well as the initial and goal points. Three agents are placed on it, each one with its starting points (**o**, **a** and **c**) and corresponding endings (**x**, **b**, and **d**) points. The paths found for each robot using A* are shown in Figure 2. In the proposed path a few collisions exist.

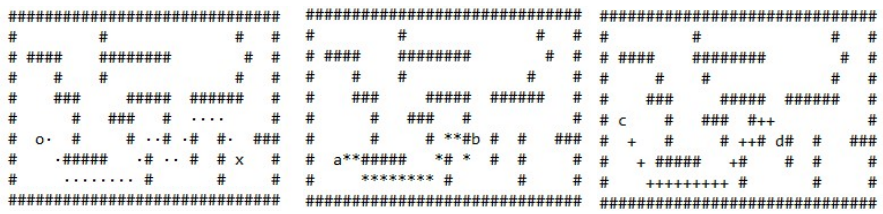

Fig. 2. Three agents generate their A* path independently in PoC.

The meta-planner instructs the priorities of agents if there is a blocker agent. As you can see in the following outputs, agent3 has been set to a "lower" priority by meta-planner due to the fact that the agent1 needs to pass through a cell that is the destination of the agent3 (that is where "d" is). Since the agent3 path has a lower number of moves (18 moves) it will reach the destination sooner than agent1, hence it will block agent1's move. To avoid this situation, the meta-planner suggests delaying its move.

```
The agent1 has no priority
The agent2 has no priority
The agent3 has low priority
---------------------------------
number of steps in path1 is: 23
number of steps in path2 is: 16
number of steps in path3 is: 18
```

The other meta-planner heuristic we have implemented is to detect the colliding cells. After executing the refinement and synthesizing phases of the meta-planner, the following moves are generated. As you can see in the following output, the agent1 has been set to wait(**w**) in t2 (**t2:w**) in favour of agent2 as both of them wanted to take cell (9,7) at t2.

```
Path for Agent 1 is:

t0:(7, 6) t1:(8, 6) t2:w t3:(9, 7) t4:(10, 8) t5:(11, 8) t6:(12, 8) t7:(13, 8) t8:(14, 8) t9:(15, 8) t10:(16, 8) t11:(17, 8) t12:(18, 7) t13:(19, 6) t14:(20, 6) t15:(21, 7) t16:(22, 7) t17:(23, 6) t18:(24, 5) t19:(25, 5) t20:(26, 5) t21:(27, 5) t22:(28, 6)

Path for Agent 2 is:

t0:(7, 7) t1:(8, 7) t2:(9, 7) t3:(10, 8) t4:(11, 8) t5:(12, 8) t6:(13, 8) t7:(14, 8) t8:(15, 8) t9:(16, 8) t10:(17, 8) t11:(18, 7) t12:(19, 6) t13:(20, 6) t14:(21, 7) t15:(22, 6) t16:0 t17:0 t18:0 t19:0 t20:0 t21:0 t22:0

Path for Agent 3 is:

t0:(6, 5) t1:(7, 6) t2:(8, 7) t3:(9, 8) t4:w t5:(10, 8) t6:(11, 8) t7:(12, 8) t8:(13, 8) t9:(14, 8) t10:(15, 8) t11:(16, 8) t12:(17, 8) t13:(18, 7) t14:(19, 6) t15:(20, 6) t16:(21, 5) t17:(22, 5) t18:(23, 6) t19:0 t20:0 t21:0 t22:0
```

5 Conclusions

This research is focused on collision avoidance multi-robot path planning. The aim of this study is to extend the outcome of A* with a simple heuristic to avoid the collisions, altogether designed and implemented in one of the latest state of the art distributed scheduling system in the cloud (i.e. Kubernetes) and adding meta-planner to augment A* to work in a multi-agent configuration.

The study represents a proof of concept and a standard maze used in path planning has been used to evaluate the heuristic proposed in this research. The performance of the heuristic has been found valid and the implementation with Kubernetes can be the next step to realize the CNMA-A*PP. Our proposal is aligned with a new trends in creating self-managed micro-services in the cloud [27]. In addition, in this paper we implemented a PoC along with two heuristics for meta-planner. This meta-planner heuristic can be upgraded to more advanced techniques such as the collision model that is proposed in [28]. The proposed solution in this paper is also aligned with the idea of Cloud4IoT which is containerizing IoT functions and optimize their placement and on the edge of network through fog computing [29].

6 Acknowledgement

This research has been funded by the Spanish Ministry of Science and Innovation, under project MINECO-TIN2017-84804-R, and by the Grant FC-GRUPIN-IDI/2018/000226 project from the Asturias Regional Government.

References

[1] T. T. Mac, C. Copot, D. T. Tran, and R. De Keyser, "Heuristic approaches in robot path planning: A survey," *Rob. Auton. Syst.*, vol. 86, pp. 13–28, 2016.

[2] L. Xie, S. Xue, J. Zhang, M. Zhang, W. Tian, and S. Haugen, "A path planning approach based on multi-direction A* algorithm for ships navigating within wind farm waters," *Ocean Eng.*, vol. 184, no. June, pp. 311–322, 2019.

[3] L. C. Wang, L. S. Yong, and M. H. Ang, "Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment," in *IEEE International Symposium on Intelligent Control - Proceedings*, 2002.

[4] S. D. Han and I. I. P. Reliminaries, "Effective Heuristics for Multi-Robot Path Planning in Warehouse Environments," *2nd IEEE Int. Symp. Multi-Robot Multi-Agent Syst.*, pp. 1–3, 2019.

[5] M. M. I, N. Wehner, and X. Yu, "Ant Colony Optimization Algorithm for Robot Path Planning," vol. 3, no. 30, p. 30, 2010.

[6] E. Erdem, D. G. Kisa, U. Oztok, and P. Schüller, "A general formal framework for pathfinding problems with multiple agents," *Proc. 27th AAAI Conf. Artif. Intell. AAAI 2013*, pp. 290–296, 2013.

[7] Surynek P., "Towards Optimal Cooperative Path Planning in Hard Setups through Satisfiability Solving.," *PRICAI 2012 Trends Artif. Intell. PRICAI 2012.*, p. pp 564-576, 2012.

[8] J. Yu and S. M. LaValle, "Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics," *IEEE Trans. Robot.*, vol. 32, no. 5, pp. 1163–1177, 2016.

[9] A. M. Andrew, "Modern heuristic search methods," *Kybernetes*, vol. 27, no. 5, pp. 582–585, 1998.

[10] I. Noreen, A. Khan, K. Asghar, and Z. Habib, "A Path-Planning Performance Comparison of RRT*-AB with MEA* in a 2-Dimensional Environment," *Symmetry (Basel).*, vol. 11, no. 7, p. 945, 2019.

[11] X. Yu, W. N. Chen, T. Gu, H. Yuan, H. Zhang, and J. Zhang, "ACO-A∗: Ant Colony Optimization plus A∗ for 3-D Traveling in Environments with Dense Obstacles," *IEEE Trans. Evol. Comput.*, vol. 23, no. 4, pp. 617–631, 2019.

[12] Q. Xu, "Collision avoidance strategy optimization based on danger immune algorithm," *Comput. Ind. Eng.*, vol. 76, pp. 268–279, Oct. 2014.

[13] B. Gravell and T. Summers, "Concurrent Goal Assignment and Collision-Free Trajectory Generation for Multiple Aerial Robots," *IFAC-PapersOnLine*, vol. 51, no. 12, pp. 75–81, Jan. 2018.

[14] 2011 Bruce, *Heuristic Search Applications*, vol. 53, no. 9. 2013.

[15] Russell Stuart; Norvig Peter, *Artificial Intelligence: A Modern Approach, Global Edition*. 2011.

[16] C. Https://github.com/simpleai-team/simpleai/graphs/contributors, "simpleai-team/ simpleai." .

[17] F. Yang and N. Chakraborty, "Multirobot simultaneous path planning and task assignment on graphs with stochastic costs," *Proc. IEEE MRS*, pp. 1–3, 2019.

[18] G. E. Mathew and G. Malathy, "Direction based heuristic for pathfinding in video

games," *2nd Int. Conf. Electron. Commun. Syst. ICECS 2015*, vol. 47, pp. 1651–1657, 2015.

[19] M. Kiadi, Q. Tan, and J. R. Villar, "Optimized Path Planning in Reinforcement Learning by Backtracking," pp. 80–90, 2019.

[20] "(McGraw-Hill computer science series) Nils J Nilsson - Problem-solving methods in artificial intelligence-New York, McGraw-Hill (1971).pdf." .

[21] Y. Zhang, J. Zhou, and J. Sun, "Scheduling bag-of-tasks applications on hybrid clouds under due date constraints," *J. Syst. Archit.*, vol. 101, no. July, p. 101654, 2019.

[22] P. Dziurzanski, S. Zhao, M. Przewozniczek, M. Komarnicki, and L. S. Indrusiak, "Scalable distributed evolutionary algorithm orchestration using Docker containers," *J. Comput. Sci.*, vol. 40, p. 101069, 2020.

[23] B. Soltani, A. Ghenai, and N. Zeghib, "Towards Distributed Containerized Serverless Architecture in Multi Cloud Environment," *Procedia Comput. Sci.*, vol. 134, pp. 121–128, 2018.

[24] N. Kratzke and P. C. Quint, "Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study," *J. Syst. Softw.*, vol. 126, pp. 1–16, 2017.

[25] E. H. Durfee, "Distributed Problem Solving and the DVMT," pp. 27–44, 1988.

[26] J. Lewis, M. Fowler, "Microservices: a definition of this new architectural term." .

[27] G. Toffetti, S. Brunner, M. Blöchlinger, J. Spillner, and T. M. Bohnert, "Self-managing cloud-native applications: Design, implementation, and experience," *Futur. Gener. Comput. Syst.*, vol. 72, pp. 165–179, 2017.

[28] S. J. You and S. H. Ji, "Design of a multi-robot bin packing system in an automatic warehouse," *ICINCO 2014 - Proc. 11th Int. Conf. Informatics Control. Autom. Robot.*, vol. 2, pp. 533–538, 2014.

[29] C. Dupont, R. Giaffreda, and L. Capra, "Edge computing in IoT context: Horizontal and vertical Linux container migration," *GIoTS 2017 - Glob. Internet Things Summit, Proc.*, pp. 2–5, 2017.